Demo 3 - Hadoop Shuffle & TOS

MIDS w261: Machine Learning at Scale | UC Berkeley School of Information | Fall 2022

Designing MapReduce algorithms involves two kinds of planning. First we have to figure out which parts of a calculation can be performed in parallel and which can't. Then we have to figure out how to put those pieces together so that the right information ends up in the right place at the right time. That's what the Hadoop shuffle is all about. Today we'll talk about a few techniques to optimize your Hadoop jobs. By the end of this demo you should be able to:

- ... **identify** what makes the Hadoop Shuffle potentially costly.
- ... **define** local aggregation & combiners.
- ... **implement** partial, unordered, and total order sort.
- ... **create** custom counters for your Hadoop Jobs.
- ... **describe** the order inversion pattern & when to use it (ie: relative frequencies).

Note: Hadoop Streaming syntax is very particular. Make sure to test your python scripts before passing them to the Hadoop job and pay careful attention to the order in which Hadoop job parameters are specified.

In [1]:
```
!hadoop version
```

```
Hadoop 3.2.3
Source code repository https://bigdataoss-internal.googlesource.com/third_party/apache/hadoop -r b5a2a261f1b443f04dd9bf6
b461544880064b3cd
Compiled by bigtop on 2023-01-03T22:17Z
Compiled with protoc 2.5.0
From source with checksum 20d2ce35888d70e98df0e9781ff3cbed
This command was run using /usr/lib/hadoop/hadoop-common-3.2.3.jar
```

Hadoop Streaming Docs

https://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html

Notebook Set-Up

In [2]:
```
!pwd
```

/

In [58]:
```
# ADMIN
#!mkdir  /media/notebooks/LiveSessionMaterials/wk03Demo_HadoopShuffle/tmp_jgs
#%cd /media/notebooks/LiveSessionMaterials/wk03Demo_HadoopShuffle/tmp_jgs
#!gsutil -m cp -r \
#  "gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/wk03
#  .
```

```
mkdir: cannot create directory ‘/media/notebooks/LiveSessionMaterials/wk03Demo_HadoopShuffle/tmp_jgs’: File exists
/media/notebooks/LiveSessionMaterials/wk03Demo_HadoopShuffle/tmp_jgs
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/DITP_fig2-4.png...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/DITP_fig2-5,6.png...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/Frequencies/combiner.py...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/Frequencies/mapper.py...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/TotalOrderSort/mapper.py...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/Frequencies/reducer.py...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/HDG_fig7-4-annotated.png...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/PartitionSort/mapper.py...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/TotalOrderSort/TOS_mapper.py...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/MultiPart/reducer.py...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/TotalOrderSort/reducer.py...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/MultiPart/mapper.py...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/HDG_fig7-4.png...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/demo3_workbook.ipynb...
Copying gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/w
k03Demo_HadoopShuffle/master/MultiPart/combiner.py...
/ [15/15 files][  1.1 MiB/  1.1 MiB] 100% Done
Operation completed over 15 objects/1.1 MiB.
```

In [ ]:
```
cd
```

In [ ]:
```
gsutil -m cp -r \
  "gs://dataproc-staging-us-central1-913378501339-ychjbide/notebooks/jupyter/ucb-w261-master/LiveSessionMaterials/wk03D
  .
```

In [4]:
```
!pwd
```

```
/root/shared/Dropbox/Projects/Courses/w261_master_jimi_Notes_2021_01_Savannah/src/Instructors-master 3-2023-01-17/LiveSe
ssionMaterials/wk03Demo_HadoopShuffle/master
```

In [2]:
```
%cd  /media/notebooks/LiveSessionMaterials/wk03Demo_HadoopShuffle/master/
```

```
/media/notebooks/LiveSessionMaterials/wk03Demo_HadoopShuffle/master
```

In [3]:
```
# imports & magic commands
import sys
%reload_ext autoreload
%autoreload 2
```

In [4]:
```
!ls /usr/lib/hadoop
```

```
bin                             hadoop-extras-3.2.3.jar
client                          hadoop-extras.jar
etc                             hadoop-fs2img-3.2.3.jar
hadoop-aliyun-3.2.3.jar         hadoop-fs2img.jar
hadoop-aliyun.jar               hadoop-gridmix-3.2.3.jar
hadoop-annotations-3.2.3.jar    hadoop-gridmix.jar
hadoop-annotations.jar          hadoop-kafka-3.2.3.jar
hadoop-archive-logs-3.2.3.jar   hadoop-kafka.jar
hadoop-archive-logs.jar         hadoop-kms-3.2.3.jar
hadoop-archives-3.2.3.jar       hadoop-kms.jar
hadoop-archives.jar             hadoop-nfs-3.2.3.jar
hadoop-auth-3.2.3.jar           hadoop-nfs.jar
hadoop-auth.jar                 hadoop-openstack-3.2.3.jar
hadoop-aws-3.2.3.jar            hadoop-openstack.jar
hadoop-aws.jar                  hadoop-resourceestimator-3.2.3.jar
hadoop-azure-3.2.3.jar          hadoop-resourceestimator.jar
hadoop-azure-datalake-3.2.3.jar hadoop-rumen-3.2.3.jar
```

```
hadoop-azure-datalake.jar        hadoop-rumen.jar
hadoop-azure.jar                 hadoop-sls-3.2.3.jar
hadoop-common-3.2.3-tests.jar    hadoop-sls.jar
hadoop-common-3.2.3.jar          hadoop-streaming-3.2.3.jar
hadoop-common.jar                hadoop-streaming.jar
hadoop-datajoin-3.2.3.jar        lib
hadoop-datajoin.jar              libexec
hadoop-distcp-3.2.3.jar          sbin
hadoop-distcp.jar
```

In [5]:
```
# globals
JAR_FILE = "/usr/lib/hadoop/hadoop-streaming.jar"
DEMO_DIR = "/media/notebooks/LiveSessionMaterials/wk03Demo_HadoopShuffle/"
HDFS_DIR = "/user/root/demo3"
!hdfs dfs -mkdir -p {HDFS_DIR}

# globals for the master dir
JAR_FILE = "/usr/lib/hadoop/hadoop-streaming.jar"
DEMO_DIR = "/media/notebooks/LiveSessionMaterials/wk03Demo_HadoopShuffle/master/"
HDFS_DIR = "/user/root/demo3"
!hdfs dfs -mkdir -p {HDFS_DIR}
```

In [6]:
```
# <--- SOLUTION --->
# ... for instructors ...
DEMO_DIR = "/media/notebooks/LiveSessionMaterials/wk03Demo_HadoopShuffle/master/"
```

In [7]:
```
# get path to notebook
PWD = !pwd
PWD = PWD[0]
```

In [8]:
```
# store notebook environment path
from os import environ
PATH  = environ['PATH']
```

REMINDER: If you are running this notebook from the course Docker container you can track your Hadoop Jobs using the UI at:
http://localhost:19888/jobhistory/

Load the Data

In this notebook, we'll continue working with the *Alice in Wonderland* text file from HW1 and the test file we created for debugging. Run the following cell to confirm that you have access to these files and save their location to a global variable to use in your Hadoop Streaming jobs.

In [9]:
```
# make a data subfolder - RUN THIS CELL AS IS
!mkdir -p data
```

In [10]:
```
# (Re)Download Alice Full text from Project Gutenberg - RUN THIS CELL AS IS
# NOTE: feel free to replace 'curl' with 'wget' or equivalent command of your choice.
!gsutil cp gs://w261-hw-data/main/Assignments/HW1/data/alice.txt data/alice.txt
```

```
Copying gs://w261-hw-data/main/Assignments/HW1/data/alice.txt...
/ [1 files][170.2 KiB/170.2 KiB]
Operation completed over 1 objects/170.2 KiB.
```

In [11]:
```
%%writefile data/alice_test.txt
This is a small test file. This file is for a test.
This small test file has two small lines.
```

```
Overwriting data/alice_test.txt
```

In [12]:
```
# save the paths - RUN THIS CELL AS IS (if Option 1 failed)
ALICE_TXT = PWD + "/data/alice.txt"
TEST_TXT = PWD + "/data/alice_test.txt"
```

In [13]:
```
# confirm the files are there - RUN THIS CELL AS IS
!echo "######### alice.txt #########"
!head -n 6 {ALICE_TXT}
!echo "######### alice_test.txt #########"
!cat {TEST_TXT}
```

```
######### alice.txt #########
The Project Gutenberg eBook of Alice’s Adventures in Wonderland, by Lewis Carroll

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this eBook or online at
######### alice_test.txt #########
```

```
This is a small test file. This file is for a test.
This small test file has two small lines.
```

Load the file into HDFS for easy access

See the docs for the difference between `-put` and `-copyFromLocal`:

https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-common/FileSystemShell.html

In [14]:
```
# load the input files into HDFS (RUN THIS CELL AS IS)
!hdfs dfs -put {TEST_TXT} {ALICE_TXT} {HDFS_DIR}
```

```
put: `/user/root/demo3/alice_test.txt': File exists
put: `/user/root/demo3/alice.txt': File exists
```

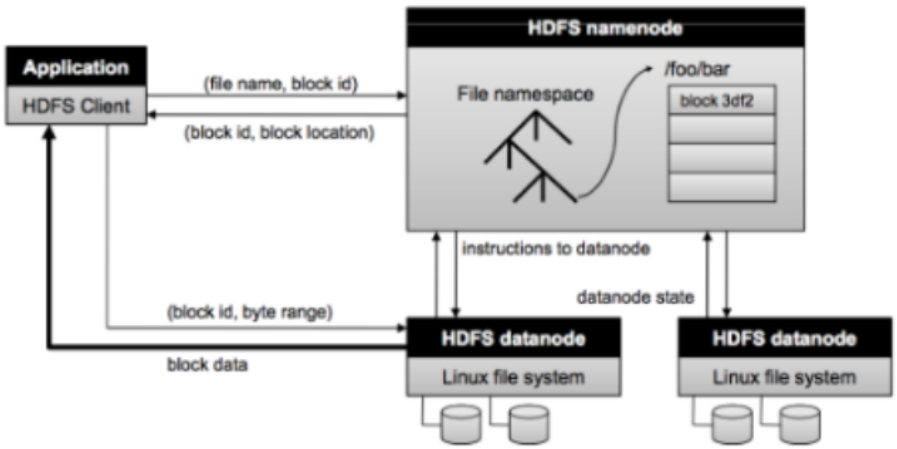
Content Overview: Hadoop Shuffle

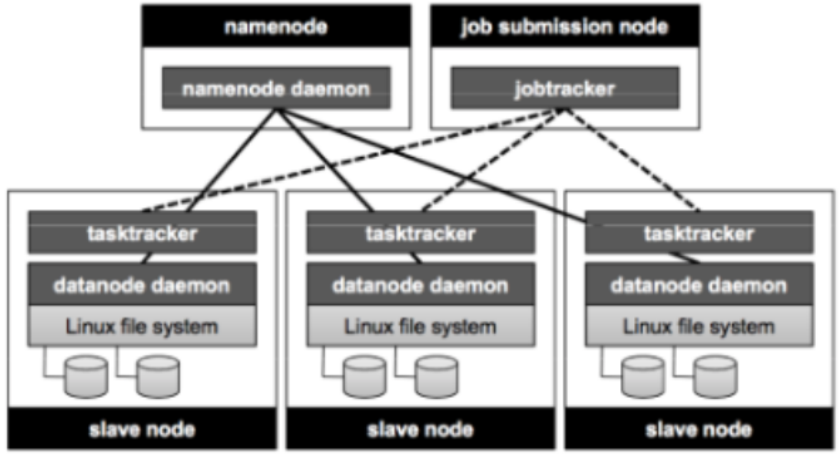
The week 3 reading from Chapter 3 of *Data Intensive Text Processing with Map Reduce* by Lin and Dyer discusses 5 key techniques for controlling execution and managing the flow of data in MapReduce:

1. using complex data structures to communicate partial results
2. user-specified initialization/termination code before/after each map/reduce task
3. preserving state across multiple keys
4. controling the sort order of intermediate keys
5. specifying the partitioning of the key space

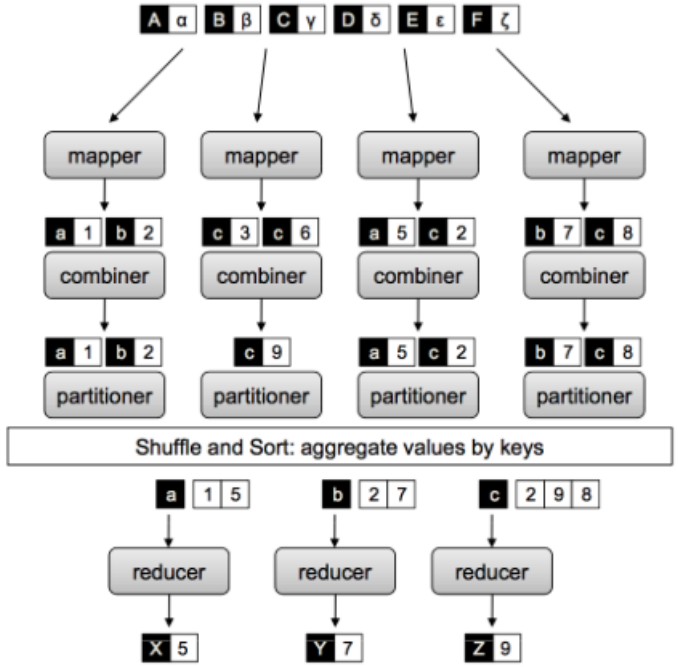
Our goal in employing these techniques is to minimize the amount we have to move the data. This involves keeping track of what data is stored where at each stage in our job (*DITP figure 2.5 and 2.6*):

Recall that a Hadoop cluster stores our data on datanotes and ships the programmer's map and reduce code to those nodes to perform those transformations in place.

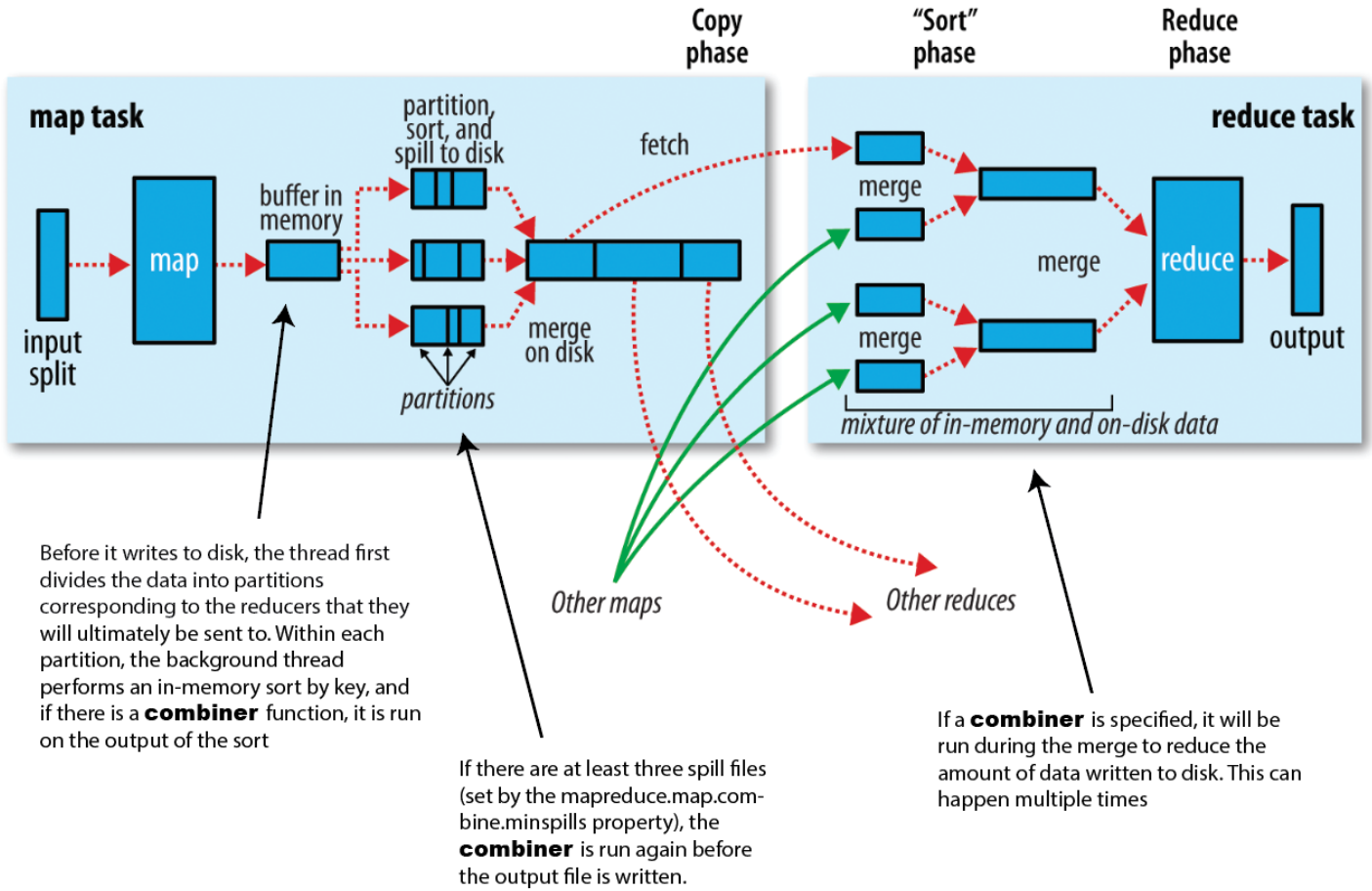




Though it isn't always possible to do so, ideally we'd like to design an implementation that retrieves the result in a single MapReduce job (*DITP figure 2.4*):

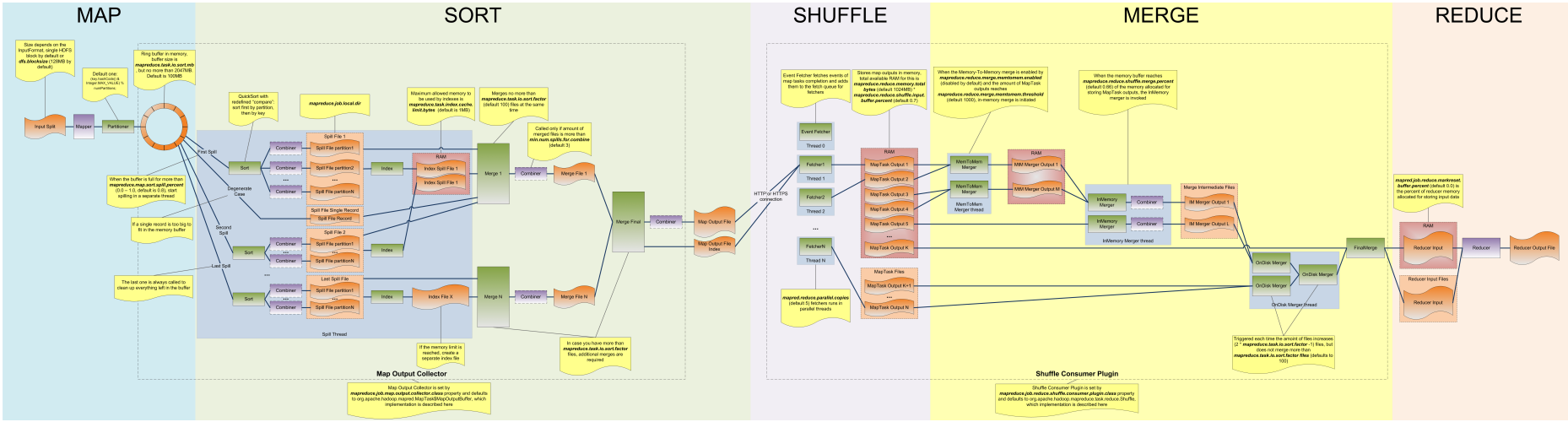


Shuffle & Sort Detail (*Hadoop, The Definitve Guide, by Tom White; fig 7-4*):



Shuffle & Sort Detail in extreme detail:

Click here to ZOOM on this diagram



> **DISCUSSION QUESTIONS:**
>
> - What work does your Hadoop cluster have to do at the shuffle stage?
> - What determines the time complexity of this work?
> - Compare the Lyn & Dyer diagram with the om White diagram. How might the Lyn & Dyer diagram be misleading?
> - What is a combiner, how does it impact the shuffle? 'where' does the combining happen?
> - What is local aggregation? Is a combiner the only way to do it?
> - What is a partitioner and how does it impact the shuffle? How/where can we specify custom paritioning behavior? If we don't specify a partitioner, will Hadoop still partition the data? How?
> - Can you think of an example of a task that can't be accomplished in a single MapReduce job? Explain.

**A note on the Lin and Dyer reading (and other readings you may encounter):*

> In MapReduce, the programmer defines a mapper and a reducer with the following signatures:
> map: (k1; v1) -> [(k2; v2)]
> reduce: (k2; [v2]) -> [(k3; v3)]
> where [..] denotes a list

It is important to note that the MapReduce framework ensures that the keys are ordered, so we know that if a key is different from the previous one, we have moved into a new key group. In contrast to the Java API, where you are provided an iterator over each key group (as per above Lin

and Dyer version), in Streaming you have to find key group boundaries in your program. (Page 39, Hadoop - The Definitive Guide)

<--- SOLUTION --->

INSTRUCTOR TALKING POINTS

- What work does your Hadoop cluster have to do at the shuffle stage? What determines the time complexity of this work?

> Short digression: there is something slightly deceptive about this kind of MapReduce diagram -- the arrows make it look like data is flowing from step to step down the chart. Of course that is not the case physically. The shuffle stage is where the data gets moved. It doesn't linearly follow the 'paritioning' but includes all the work done by the framework after the mapper emits the data and before the reducers start. The work involved is costly in two ways: first in order to perform the sort Hadoop has to look at which keys are present on each data node & make a bunch of comparisions to sort these keys and plan where records with those keys are going to end up, then the data has to be transfered. The complexity is a function of the number of records output by our mappers (NOTE: this is different than the number of records we started with.)

- Compare the Lyn & Dyer diagram with the om White diagram. How might the Lyn & Dyer diagram be misleading?

> In the Lyn & Dyer diagram combiners appear to only be executed at the map phase.

- What is a combiner, how does it impact the shuffle? 'where' does the combining happen?

> A combiner is an aggregation script specfied by the programmer that will take mapper output records with the same key and turn them into a single, combined, record. Usually this is desireable because fewer records = a faster shuffle. However its important to note that Hadoop uses the combiners strategically -- sometimes records are combined before leaving the mapper node, sometimes after arriving at a reducer node (this is related to the 'circular buffer' that you may remember discussed in the async). There are two key takeaways here 1) that Hadoop doesn't guarantee that it will perform the combining on any or all records and 2) Hadoop makes smart choice about when to hold data in memory/combine/spill to disk... saving a lot of work for the programmer.

- What is local aggregation? Is a combiner the only way to do it?

> Local aggregation is basically what we described a minute ago: reducing the number of records that need to be transfered over the network in the shuffle state *OR* between two sequential MapReduce jobs. Combiners are *NOT* the only way to perform local aggregation, we could also use an in memory form of aggregation eg. a python dicitonary. Of course there are

> risks to this approach if your mappers are emitting more records than the ones they read in. (*NOTE:* local aggregation is one example of the 2nd technique that Lin & Dyer list: "preserving state across multiple keys").

- What is a partitioner and how does it impact the shuffle? How/where can we specify custom partitioning behavior? If we don't specify a partitioner, will Hadoop still partition the data? How?

> The 'partitioner' parameter in our Hadoop jobs (we saw this last week in live session) simply tells Hadoop how to determine which keys end up together on the same reducer node. Custom paritioning is just a matter of manipulating the key format/data structure -- this would happen in a mapper script there is no 'paritioner' script. Anytime the programmer specifies more than one reducer then Hadoop will perform partitioning... if we don't tell it how to partittion explicitly then Hadoop will make its own decision about how to split up the key space.

- Can you think of an example of a task that can't be accomplished in a single MapReduce job? Explain.

> Students should be able to come up with sorted word count (challenge question from the end of live session in week2) --> can't sort until after reducing so you need a second job. We'll look at an example below of calculating relative frequencies... it would be good to guide them to consider the key challenge in that task: you can't divide until you have a total but what if you don't want to hold all of the counts in memory? (we'll teach order inversion below, no need to mention that here).

Exercise 1: Relative Frequencies with one reducer (using order inversion)

In last week's live session and HW1 we used Word Count as a cannonical example of an embarassingly parallel task. At first glance, computing relative frequencies (`word count / total count`) seems like it would be just as easy to implement -- after all it's just word count with an extra division at the end. However this task actually presents a small design challenge that is perfect to illustrate a few of the techniques that Lin & Dyer talk about.

The Order Inversion (OI) design pattern can be used to control the order of reducer values in the MapReduce framework (which is useful because some computations require data in a partiticular sequence). In other words, the OI pattern is to properly sequence data (key-value records) presented to the reducer. A simple ilustrative example of the OI pattern in action is the calculation of the relative frequency of words (in the word count example), where two pieces of the data are required to calculate the word relative frequency. We can simply adapt our existing word count Hadoop map-reduce job to keep track of the total word count processed by each mapper task and ship that information to the reducer(assume one reducer for now) such that those intermediate total records arrive first at the reducer and can be tallied up before any word count records are processed. Then that total word count information can be used to normalize each word count frequency.

DISCUSSION:

> - Talk through what the MapReduce job would look like? What is the challenge here?
> - Is it possible to compute relative frequencies in a single MapReduce job?
> - Is it possible to compute relative frequencies in a single MapReduce job with multiple reducers?

<--- SOLUTION --->

INSTRUCTOR TALKING POINTS (before exercise 1)

- Talk through what the MapReduce job would look like? What is the challenge here?

> The map would work the same as in word count, then the reducer would aggregate counts and also add up a total word count so that it can divide each word's count by the total. The challenge is that we won't know the total until after reducing. In word count we count in parallel and then add at the end. To compute frequencies we can still count in parallel but then we need to add both individual words and the total number of words before we can divide.

- Is it possible to compute relative frequencies in a single MapReduce job?

> Yes, we could hold all the word counts in memory and then divide at the end... but that's not scalable.

- Is it possible to compute relative frequencies in a single MapReduce job with multiple reducers?

> No, if we use multiple reducers we'd have no way to get the total-- we'd just have partial totals on each reducer node. We'd need a second job to get the full total & perform the division.

.... OK *actually* there IS a scalable a way to do this. Its called the order inversion pattern. Lets take a look:

Exercise 1 Tasks:

- **a) read provided code:** We've provided a naive interpretation to get you started. Take a look at `Frequencies/mapper.py`, `Frequencies/combiner.py` and `Frequencies/reducer.py`.
- **b) discuss:** How does it resolve the challenge of computing the total? Uncomment line 22 in the mapper and lines 26-27 in the reducer to take advantage of this 'solution', then run the unit tests below, to confirm that you understand what's going on in each of these scripts.

Despite solving the problem of computing the total in a single map-reduce job, what is wrong with this approach?

- **c) fix the problem:** To fix the problem, all we need to do is make a small change to the key used to emit the total counts (you need to do this in both the mapper and reducer). Think about Hadoop's default sorting. What keys arrive first? What key could you assign to the total that would be guaranteed to arrive first? Once you are satisfied that your solution works, run the provided Hadoop job on the test file & then the full `alice` text. (note it has a single reducer for now).

- **d) discuss:** Now, re-run the job with 4 reducers, what happens to the results? What would we have to do to fix the problem?

In [15]:
```
# part b - make sure scripts are executable (RUN THIS CELL AS IS)
!chmod a+x Frequencies/mapper.py
!chmod a+x Frequencies/combiner.py
!chmod a+x Frequencies/reducer.py
```

Frequencies/mapper.py

```
#!/usr/bin/env python
"""
Mapper script to tokenize words from a line of text.
INPUT:
    a text file
OUTPUT:
    word \t partialCount

NOTE: Uncomment line 22 before running.
"""
import re
import sys

# read from standard input
for line in sys.stdin:
    line = line.strip()
    # tokenize
    words = re.findall(r'[a-z]+', line.lower())
    # emit words and count of 1 plus total counter
    for word in words:
        print(f'{word}\t{1}')
        #print(f'total\t{1}')  # part b/c - UNCOMMENT & MAKE YOUR CHANGE HERE
        print(f'!total\t{1}')      # <--- SOLUTION --->
```

Frequencies/reducer.py

```
#!/usr/bin/env python
"""
Reducer script to add counts with the same key and
divide by total count to get relative frequency.
INPUT:
    word \t partialCount
OUTPUT:
    word \t totalCount
"""
import sys

# initialize trackers
cur_word = None
cur_count = 0
total = 0

# read input key-value pairs from standard input
for line in sys.stdin:
    key, value = line.split()
    # tally counts from current key
    if key == cur_word:
        cur_count += int(value)
    # OR ...
    else:
        # store word count total
        #if cur_word == 'total':   # part b/c - UNCOMMENT & MAKE YOUR CHANGE HERE
        if cur_word == '!total':         # <--- SOLUTION --->
            total = float(cur_count)
        # emit realtive frequency
        if cur_word:
            print(f'{cur_word}\t{cur_count/total}')
        # and start a new tally
        cur_word, cur_count  = key, int(value)

# don't forget the last record!
print(f'{cur_word}\t{cur_count/total}')
```

Frequencies/combiner.py

```
#!/usr/bin/env python
"""
Combiner script to add counts with the same key.
INPUT:
    word \t partialCount
OUTPUT:
    word \t totalCount
"""
import sys

# initialize trackers
cur_word = None
cur_count = 0

# read input key-value pairs from standard input
for line in sys.stdin:
    key, value = line.split()
    # tally counts from current key
    if key == cur_word:
        cur_count += int(value)
    # OR emit current total and start a new tally
    else:
        if cur_word:
            print(f'{cur_word}\t{cur_count}')
        cur_word, cur_count  = key, int(value)

# don't forget the last record!
print(f'{cur_word}\t{cur_count}')
```

In [16]:

```
# part b - unit test mapper script
!echo "foo foo quux labs foo bar quux" | Frequencies/mapper.py
```

```
foo     1
!total  1
foo     1
!total  1
quux    1
!total  1
```

```
labs    1
!total  1
foo     1
!total  1
bar     1
!total  1
quux    1
!total  1
```

In [17]:

```
# part b - unit test map-combine (sort mimics shuffle) (RUN THIS CELL AS IS)
!echo "foo foo quux labs foo bar quux" | Frequencies/mapper.py | sort -k1,1 | Frequencies/combiner.py
```

```
!total  7
bar     1
foo     3
labs    1
quux    2
```

In [18]:

```
# part b - unit test map-combine-reduce (sort mimics shuffle) (RUN THIS CELL AS IS)
!echo "foo foo quux labs foo bar quux" | Frequencies/mapper.py | sort -k1,1 | Frequencies/combiner.py | Frequencies/redu
```

```
!total  1.0
bar     0.14285714285714285
foo     0.42857142857142855
labs    0.14285714285714285
quux    0.2857142857142857
```

In [19]:

```
# parts c - clear the output directory (RUN THIS CELL AS IS)
!hdfs dfs -rm -r {HDFS_DIR}/frequencies-output
# NOTE: this directory won't exist unless you are re-running a job, that's fine.
```

```
Deleted /user/root/demo3/frequencies-output
```

In [20]:

```
# parts c - Hadoop streaming job (RUN THIS CELL AS IS FIRST, then make your modification)
!hadoop jar {JAR_FILE} \
  -files Frequencies/reducer.py,Frequencies/mapper.py,Frequencies/combiner.py \
  -mapper mapper.py \
  -combiner combiner.py \
  -reducer reducer.py \
  -input {HDFS_DIR}/alice_test.txt \
  -output {HDFS_DIR}/frequencies-output \
  -cmdenv PATH={PATH} \
  -numReduceTasks 1
```

```
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar] /tmp/streamjob1744408072977128658.jar tmpDir=null
2023-01-24 23:15:05,229 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.55:8032
2023-01-24 23:15:05,445 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.55:10200
2023-01-24 23:15:05,989 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.55:8032
2023-01-24 23:15:05,990 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.55:10200
2023-01-24 23:15:06,189 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
root/.staging/job_1674579266007_0006
2023-01-24 23:15:06,636 INFO mapred.FileInputFormat: Total input files to process : 1
2023-01-24 23:15:07,099 INFO mapreduce.JobSubmitter: number of splits:10
2023-01-24 23:15:07,289 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1674579266007_0006
2023-01-24 23:15:07,291 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-01-24 23:15:07,480 INFO conf.Configuration: resource-types.xml not found
2023-01-24 23:15:07,480 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-01-24 23:15:07,746 INFO impl.YarnClientImpl: Submitted application application_1674579266007_0006
2023-01-24 23:15:07,783 INFO mapreduce.Job: The url to track the job: http://w261-m:8088/proxy/application_1674579266007
_0006/
2023-01-24 23:15:07,785 INFO mapreduce.Job: Running job: job_1674579266007_0006
2023-01-24 23:15:15,959 INFO mapreduce.Job: Job job_1674579266007_0006 running in uber mode : false
2023-01-24 23:15:15,960 INFO mapreduce.Job:  map 0% reduce 0%
2023-01-24 23:15:26,203 INFO mapreduce.Job:  map 30% reduce 0%
2023-01-24 23:15:31,263 INFO mapreduce.Job:  map 40% reduce 0%
2023-01-24 23:15:34,296 INFO mapreduce.Job:  map 60% reduce 0%
2023-01-24 23:15:38,324 INFO mapreduce.Job:  map 70% reduce 0%
2023-01-24 23:15:40,341 INFO mapreduce.Job:  map 80% reduce 0%
2023-01-24 23:15:41,362 INFO mapreduce.Job:  map 90% reduce 0%
2023-01-24 23:15:44,378 INFO mapreduce.Job:  map 100% reduce 0%
2023-01-24 23:15:49,402 INFO mapreduce.Job:  map 100% reduce 100%
2023-01-24 23:15:51,419 INFO mapreduce.Job: Job job_1674579266007_0006 completed successfully
2023-01-24 23:15:51,507 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=150
                FILE: Number of bytes written=2760210
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1450
                HDFS: Number of bytes written=103
                HDFS: Number of read operations=35
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=3
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=10
                Launched reduce tasks=1
                Data-local map tasks=10
                Total time spent by all maps in occupied slots (ms)=197552976
                Total time spent by all reduces in occupied slots (ms)=9853032
```

```
                Total time spent by all map tasks (ms)=62596
                Total time spent by all reduce tasks (ms)=3122
                Total vcore-milliseconds taken by all map tasks=62596
                Total vcore-milliseconds taken by all reduce tasks=3122
                Total megabyte-milliseconds taken by all map tasks=197552976
                Total megabyte-milliseconds taken by all reduce tasks=9853032
        Map-Reduce Framework
                Map input records=2
                Map output records=40
                Map output bytes=311
                Map output materialized bytes=204
                Input split bytes=960
                Combine input records=40
                Combine output records=16
                Reduce input groups=11
                Reduce shuffle bytes=204
                Reduce input records=16
                Reduce output records=11
                Spilled Records=32
                Shuffled Maps =10
                Failed Shuffles=0
                Merged Map outputs=10
                GC time elapsed (ms)=1602
                CPU time spent (ms)=15710
                Physical memory (bytes) snapshot=6253367296
                Virtual memory (bytes) snapshot=48999034880
                Total committed heap usage (bytes)=5561647104
                Peak Map Physical memory (bytes)=624218112
                Peak Map Virtual memory (bytes)=4514324480
                Peak Reduce Physical memory (bytes)=289214464
                Peak Reduce Virtual memory (bytes)=4451692544
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=490
        File Output Format Counters
                Bytes Written=103
2023-01-24 23:15:51,507 INFO streaming.StreamJob: Output directory: /user/root/demo3/frequencies-output
```

In [34]:

```
# take a look at the first few results (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/frequencies-output/part-00000 | head -n 10
```

```
!total  1.0
a       0.1
file    0.15
for     0.05
has     0.05
is      0.1
lines   0.05
small   0.15
test    0.15
this    0.15
```

In [35]:

```
# <--- SOLUTION --->
# parts c - Hadoop streaming job (RUN THIS CELL AS IS FIRST, then make your modification)
!hdfs dfs -rm -r {HDFS_DIR}/frequencies-output
!hadoop jar {JAR_FILE} \
  -files Frequencies/reducer.py,Frequencies/mapper.py,Frequencies/combiner.py \
  -mapper mapper.py \
  -reducer reducer.py \
  -input {HDFS_DIR}/alice.txt \
  -output {HDFS_DIR}/frequencies-output \
  -cmdenv PATH={PATH} \
  -numReduceTasks 1
!hdfs dfs -cat {HDFS_DIR}/frequencies-output/part-00000 | head -n 10
```

```
Deleted /user/root/demo3/frequencies-output
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar] /tmp/streamjob6699798022098232158.jar tmpDir=null
2022-09-06 18:48:56,484 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.42:8032
2022-09-06 18:48:56,720 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.42:10200
2022-09-06 18:48:57,227 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.42:8032
2022-09-06 18:48:57,227 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.42:10200
2022-09-06 18:48:57,436 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
root/.staging/job_1662400986413_0002
2022-09-06 18:48:57,887 INFO mapred.FileInputFormat: Total input files to process : 1
2022-09-06 18:48:58,008 INFO mapreduce.JobSubmitter: number of splits:9
2022-09-06 18:48:58,237 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1662400986413_0002
2022-09-06 18:48:58,239 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-09-06 18:48:58,453 INFO conf.Configuration: resource-types.xml not found
2022-09-06 18:48:58,454 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-09-06 18:48:58,528 INFO impl.YarnClientImpl: Submitted application application_1662400986413_0002
2022-09-06 18:48:58,575 INFO mapreduce.Job: The url to track the job: http://w261-m:8088/proxy/application_1662400986413
_0002/
2022-09-06 18:48:58,577 INFO mapreduce.Job: Running job: job_1662400986413_0002
2022-09-06 18:49:07,698 INFO mapreduce.Job: Job job_1662400986413_0002 running in uber mode : false
2022-09-06 18:49:07,700 INFO mapreduce.Job:  map 0% reduce 0%
2022-09-06 18:49:16,846 INFO mapreduce.Job:  map 33% reduce 0%
2022-09-06 18:49:23,929 INFO mapreduce.Job:  map 44% reduce 0%
```

```
2022-09-06 18:49:24,937 INFO mapreduce.Job:  map 56% reduce 0%
2022-09-06 18:49:25,944 INFO mapreduce.Job:  map 67% reduce 0%
2022-09-06 18:49:30,990 INFO mapreduce.Job:  map 78% reduce 0%
2022-09-06 18:49:32,004 INFO mapreduce.Job:  map 89% reduce 0%
2022-09-06 18:49:33,012 INFO mapreduce.Job:  map 100% reduce 0%
2022-09-06 18:49:39,047 INFO mapreduce.Job:  map 100% reduce 100%
2022-09-06 18:49:41,065 INFO mapreduce.Job: Job job_1662400986413_0002 completed successfully
2022-09-06 18:49:41,156 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=612636
                FILE: Number of bytes written=3730535
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=207900
                HDFS: Number of bytes written=89865
                HDFS: Number of read operations=32
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=3
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=1
                Launched map tasks=9
                Launched reduce tasks=1
                Data-local map tasks=9
                Total time spent by all maps in occupied slots (ms)=192863160
                Total time spent by all reduces in occupied slots (ms)=11336352
                Total time spent by all map tasks (ms)=61110
                Total time spent by all reduce tasks (ms)=3592
                Total vcore-milliseconds taken by all map tasks=61110
                Total vcore-milliseconds taken by all reduce tasks=3592
                Total megabyte-milliseconds taken by all map tasks=192863160
                Total megabyte-milliseconds taken by all reduce tasks=11336352
        Map-Reduce Framework
                Map input records=3761
                Map output records=61128
                Map output bytes=490374
                Map output materialized bytes=612684
                Input split bytes=819
                Combine input records=0
                Combine output records=0
                Reduce input groups=3007
                Reduce shuffle bytes=612684
                Reduce input records=61128
                Reduce output records=3007
                Spilled Records=122256
                Shuffled Maps =9
                Failed Shuffles=0
```

```
                Merged Map outputs=9
                GC time elapsed (ms)=1626
                CPU time spent (ms)=18090
                Physical memory (bytes) snapshot=5699006464
                Virtual memory (bytes) snapshot=44508078080
                Total committed heap usage (bytes)=4613210112
                Peak Map Physical memory (bytes)=613621760
                Peak Map Virtual memory (bytes)=4473020416
                Peak Reduce Physical memory (bytes)=316416000
                Peak Reduce Virtual memory (bytes)=4451037184
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=207081
        File Output Format Counters
                Bytes Written=89865
2022-09-06 18:49:41,156 INFO streaming.StreamJob: Output directory: /user/root/demo3/frequencies-output
!total  1.0
a       0.022739170265672033
abide   6.543646119617851e-05
able    3.2718230598089256e-05
about   0.003337259521005104
above   9.815469179426777e-05
absence 3.2718230598089256e-05
absurd  6.543646119617851e-05
accept  3.2718230598089256e-05
acceptance      3.2718230598089256e-05
cat: Unable to write to output stream.
```

In [ ]:

Expected Results:

part c (test)	part c (full)
!total 1.0	!total 1.0
a 0.1	a 0.022728759238668322
file 0.15	abide 6.540650140623977e-05
for 0.05	able 3.270325070311989e-05
has 0.05	about 0.003335731571718229
is 0.1	above 9.810975210935967e-05

```
lines   0.05  absence 3.270325070311989e-05
small   0.15  absurd  6.540650140623977e-05
test    0.15  accept  3.270325070311989e-05
this    0.15  acceptance  3.270325070311989e-05
```

DISCUSSION:

> - Was the original implementation scalable? How does it resolve the challenge of computing the total?
> - Despite solving that problem, what is wrong with this approach?
> - Did you come up with a new key that is guaranteed to arrive first? how?
> - What happened when you went from 1 reducer to 4, why? (HINT: Use the UI to see the error logs for the failed reduce tasks)
> - What do we need to do with the total counts when we move up to 4 reducers?

In [21]:

```
# <--- SOLUTION --->
# parts c - Hadoop streaming job (RUN THIS CELL AS IS FIRST, then make your modification)
!hdfs dfs -rm -r {HDFS_DIR}/frequencies-output
!hadoop jar {JAR_FILE} \
  -files Frequencies/reducer.py,Frequencies/mapper.py,Frequencies/combiner.py \
  -mapper mapper.py \
  -reducer reducer.py \
  -input {HDFS_DIR}/alice.txt \
  -output {HDFS_DIR}/frequencies-output \
  -cmdenv PATH={PATH} \
  -numReduceTasks 4
!hdfs dfs -cat {HDFS_DIR}/frequencies-output/part-00000 | head -n 10
```

```
Deleted /user/root/demo3/frequencies-output
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar] /tmp/streamjob8650819795652039399.jar tmpDir=null
2023-01-25 00:52:03,473 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.55:8032
2023-01-25 00:52:03,685 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.55:10200
2023-01-25 00:52:04,193 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.55:8032
2023-01-25 00:52:04,194 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.55:10200
2023-01-25 00:52:04,413 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
root/.staging/job_1674579266007_0007
2023-01-25 00:52:04,850 INFO mapred.FileInputFormat: Total input files to process : 1
2023-01-25 00:52:04,957 INFO mapreduce.JobSubmitter: number of splits:9
2023-01-25 00:52:05,162 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1674579266007_0007
2023-01-25 00:52:05,163 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-01-25 00:52:05,350 INFO conf.Configuration: resource-types.xml not found
2023-01-25 00:52:05,350 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-01-25 00:52:05,412 INFO impl.YarnClientImpl: Submitted application application_1674579266007_0007
2023-01-25 00:52:05,449 INFO mapreduce.Job: The url to track the job: http://w261-m:8088/proxy/application_1674579266007
```

```
_0007/
2023-01-25 00:52:05,450 INFO mapreduce.Job: Running job: job_1674579266007_0007
2023-01-25 00:52:14,553 INFO mapreduce.Job: Job job_1674579266007_0007 running in uber mode : false
2023-01-25 00:52:14,554 INFO mapreduce.Job:  map 0% reduce 0%
2023-01-25 00:52:24,684 INFO mapreduce.Job:  map 33% reduce 0%
2023-01-25 00:52:31,792 INFO mapreduce.Job:  map 56% reduce 0%
2023-01-25 00:52:33,804 INFO mapreduce.Job:  map 67% reduce 0%
2023-01-25 00:52:38,846 INFO mapreduce.Job:  map 89% reduce 0%
2023-01-25 00:52:39,856 INFO mapreduce.Job:  map 100% reduce 0%
2023-01-25 00:52:47,919 INFO mapreduce.Job:  map 100% reduce 25%
2023-01-25 00:52:47,922 INFO mapreduce.Job: Task Id : attempt_1674579266007_0007_r_000001_0, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.close(PipeReducer.java:134)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:454)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:393)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:174)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1762)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:168)

2023-01-25 00:52:48,988 INFO mapreduce.Job: Task Id : attempt_1674579266007_0007_r_000002_0, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.close(PipeReducer.java:134)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:454)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:393)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:174)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1762)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:168)

2023-01-25 00:52:52,009 INFO mapreduce.Job: Task Id : attempt_1674579266007_0007_r_000003_0, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.close(PipeReducer.java:134)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:454)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:393)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:174)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1762)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:168)
```

```
2023-01-25 00:52:55,032 INFO mapreduce.Job: Task Id : attempt_1674579266007_0007_r_000001_1, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.close(PipeReducer.java:134)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:454)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:393)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:174)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1762)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:168)

2023-01-25 00:52:57,053 INFO mapreduce.Job: Task Id : attempt_1674579266007_0007_r_000002_1, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.close(PipeReducer.java:134)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:454)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:393)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:174)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1762)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:168)

2023-01-25 00:53:00,075 INFO mapreduce.Job: Task Id : attempt_1674579266007_0007_r_000003_1, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.close(PipeReducer.java:134)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:454)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:393)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:174)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1762)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:168)

2023-01-25 00:53:03,108 INFO mapreduce.Job: Task Id : attempt_1674579266007_0007_r_000001_3, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.close(PipeReducer.java:134)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:454)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:393)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:174)
```

```
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1762)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:168)

2023-01-25 00:53:05,119 INFO mapreduce.Job:  map 100% reduce 100%
2023-01-25 00:53:06,131 INFO mapreduce.Job: Job job_1674579266007_0007 failed with state FAILED due to: Task failed task
_1674579266007_0007_r_000001
Job failed as tasks failed. failedMaps:0 failedReduces:1 killedMaps:0 killedReduces: 0

2023-01-25 00:53:06,226 INFO mapreduce.Job: Counters: 57
        File System Counters
                FILE: Number of bytes read=436860
                FILE: Number of bytes written=3555649
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=207900
                HDFS: Number of bytes written=23854
                HDFS: Number of read operations=32
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=3
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Failed reduce tasks=8
                Killed map tasks=1
                Killed reduce tasks=2
                Launched map tasks=9
                Launched reduce tasks=10
                Data-local map tasks=9
                Total time spent by all maps in occupied slots (ms)=201207624
                Total time spent by all reduces in occupied slots (ms)=161041212
                Total time spent by all map tasks (ms)=63754
                Total time spent by all reduce tasks (ms)=51027
                Total vcore-milliseconds taken by all map tasks=63754
                Total vcore-milliseconds taken by all reduce tasks=51027
                Total megabyte-milliseconds taken by all map tasks=201207624
                Total megabyte-milliseconds taken by all reduce tasks=161041212
        Map-Reduce Framework
                Map input records=3761
                Map output records=61128
                Map output bytes=490374
                Map output materialized bytes=612846
                Input split bytes=819
                Combine input records=0
                Combine output records=0
                Reduce input groups=797
                Reduce shuffle bytes=436908
```

```
                Reduce input records=42419
                Reduce output records=797
                Spilled Records=103547
                Shuffled Maps =9
                Failed Shuffles=0
                Merged Map outputs=9
                GC time elapsed (ms)=1719
                CPU time spent (ms)=20560
                Physical memory (bytes) snapshot=5712138240
                Virtual memory (bytes) snapshot=44627369984
                Total committed heap usage (bytes)=4842848256
                Peak Map Physical memory (bytes)=636485632
                Peak Map Virtual memory (bytes)=4513640448
                Peak Reduce Physical memory (bytes)=304222208
                Peak Reduce Virtual memory (bytes)=4452143104
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=207081
        File Output Format Counters
                Bytes Written=23854
2023-01-25 00:53:06,227 ERROR streaming.StreamJob: Job not successful!
Streaming Command Failed!
!total  1.0
a       0.022739170265672033
abide   6.543646119617851e-05
about   0.003337259521005104
above   9.815469179426777e-05
absence 3.2718230598089256e-05
acceptance      3.2718230598089256e-05
accepted        6.543646119617851e-05
accessed        3.2718230598089256e-05
accident        6.543646119617851e-05
cat: Unable to write to output stream.
```

<--- SOLUTION --->

INSTRUCTOR TALKING POINTS (after exercise 1)

- Was the original implementation scalable? How does it resolve the challenge of computing the total?

> Yes, instead of holding records in memory it emits a second record allowing us to count the totals without relying on the word counts.

- Despite solving that problem, what is wrong with this approach?

> The Hadoop Shuffle ensures that the key "Total" arrives in alphabetical order... after most of our word counts have already been added up. We need the total to arrive first.

- Did you come up with a new key that is guaranteed to arrive first? how?

> *Total, !Total etc

- What happened when you went from 1 reducer to 4, why?

> The Hadoop job fails due to a zero division error (ask them if they used the UI to see the error logs for the failed reduce tasks). This happens because the total key only gets sent to one of the 4 reducers.

- What do we need to do with the total counts when we move up to 4 reducers?

> We need a way to make the total available to each reducer node (partition)

Exercise 2: Sorting and Custom Partitioning

Examples of Different Sort Types (in context of Hadoop and HDFS)

Demonstrated below is an example dataset in text format on HDFS. It includes three partitions in an HDFS directory. Each partition stores records in the format of `{Integer} [TAB] {English Word}` .

```
files in hdfs directory
2016-07-20 22:04:56          0 _SUCCESS
2016-07-20 22:04:45    2392650 part-00000
2016-07-20 22:04:44    2368850 part-00001
2016-07-20 22:04:45    2304038 part-00002
```

◉ Partial Sort ○ Total Sort (Unorderd Partitions) ○ Total Sort (Ordered Partitions)

Partial Sort

file: part-00000		file: part-00001		file: part-00002	
27	driver	30	do	26	descent
27	creating	28	dataset	26	def
27	experiements	15	computing	25	compute
19	consists	15	document	24	done
19	evaluate	15	computational	24	code
17	drivers	14	center	23	descent
10	clustering	5	distributed	22	corresponding
9	during	4	develop	13	efficient
9	change	3	different	1	cell
7	contour	2	cluster	0	current

Keys are assigned to buckets without any ordering. Keys are sorted within each bucket (the key is the the number in the first column rendered in red).

Total Sort (Unorderd Partitions)

file: part-00000		file: part-00001		file: part-00002	
				30	do
		10	clustering	28	dataset
19	consists	9	during	27	driver
19	evaluate	9	change	27	creating
17	drivers	7	contour	27	experiements
15	computing	5	distributed	26	descent
15	document	4	develop	26	def
15	computational	3	different	25	compute
14	center	2	cluster	24	done
13	efficient	1	cell	24	code
		0	current	23	descent
				22	corresponding

Keys are assigned to buckets according to their numeric value. The result is that all keys between 20-30 end up in one bucket, keys between 10-20 end up in another bucket, and keys 0-10 end up in another bucket. Keys are sorted within each bucket. Partitions are not assigned in sorted order.

Total Sort (Ordered Partitions)

file: part-00000		file: part-00001		file: part-00002	
30	do				
28	dataset				
27	creating	19	evaluate	9	during
27	driver	19	consists	9	change
27	experiements	17	drivers	7	contour
26	def	15	document	5	distributed
26	descent	15	computing	4	develop
25	compute	15	computational	3	different
24	code	14	center	2	cluster
24	done	13	efficient	1	cell
23	descent	10	clustering	0	current
22	corresponding				

Keys are assigned to buckets according to their numeric value. The result is that all keys between 20-30 end up in one bucket, keys between 10-20 end up in another bucket, and keys 0-10 end up in another bucket. Keys are sorted within each bucket. Here, partitions are assigned in sorted order, such that keys between 20-30 end up in the first bucket, keys between 10-20 end up in the second bucket, and keys 0-10 end up in the third bucket. We use the term buckets and partitions interchageably.

Exercise 2: Custom Partitioning for sorting

Last week in Breakout 4 you learned out to implement a secondary sort -- that is, you learned how to tell Hadoop to order key-value pairs within each partition based on the value. However you also saw the limitation of this simple secondary sort: namely that sorting within a partition is not very useful if you need to use multiple reducers because, for example, the top word could end up in any one of the partitions and the next highest might end up in a totally different partition. Of course post processing your partially sorted partition files (eg. using mergesort) might solve this problem, but if your data is too large to fit on a single machine that is not a viable solution.

Luckily, Hadoop provides a way to partition the data across reducers in a user-defined way. This is done telling Hadoop to use all or part of the composite key as a partitioning key. All lines with the same partition key are guaranteed to be processed by the same reducer. This is similar to the sort key, but allows for control at a higher level. This "custom partitioning" will both solve our sorting troubles from last week and solve the

problem you saw when using multiple reducers in Exercise 1 this week. To use custom partitioning, there are 2 more parameters we need to add to our Hadoop Streaming Command:

`-D mapreduce.partition.keypartitioner.options="-k1,1"` : tells Hadoop that we want to use the first key field as a partition key. Note: just like the `keycomparator`, `keypartitioner` must be used in conjuction with `stream.num.map.output.key.fields`.

`-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner` : tells Hadoop that we want to partition the data in a custom way. Note about partitioner: this line **MUST** appear after all of the `-D` options or it will be ignored.

> **DISCUSSION QUESTIONS (before exercise 2):**
>
> - Quick review: What is a composite key?
> - Quick review: Practically speaking, what is a 'partition'? How is this concept related to the HDFS output of a job?
> - What is the difference between the partition key & the 'sort' key?

Review

In Hadoop a composite key can be used when we want to control which sets of records are shuffled (partition the data) to the same reducer node. For example, the following specifies a composite key that is used to partition mapper records such that records in the same partition are sent to the same reducer:

```
-D stream.num.map.output.key.fields=4 \
    -D map.output.key.field.separator=. \
    -D mapreduce.partition.keypartitioner.options=-k1,2
```

In Hadoop a composite key can be used to determine the sort order of records arriving at a reducer. For example, this can be accomplished as follows:

```
-D stream.num.map.output.key.fields=4 \
    -D mapreduce.map.output.key.field.separator=. \
    -D mapreduce.partition.keycomparator.options= -k3n,3 -k2,2   \
```

<--- SOLUTION --->

INSTRUCTOR TALKING POINTS (before exercise 2)

- Quick review: What is a composite key?

> We created a composite key in Breakout 4 -- that was when we told Hadoop to treat the first two fields together as the key. We did this above for sorting purposes. Another way to create a composite key would have been to simply make a string joinin the two fields with a comma or dash (anything other than the Hadoop field delimiter).

- Quick review: Practically speaking, what is a 'partition'? How is this concept related to the HDFS output of a job?

> Partition = data split / guaranteed co-location of records for processing in reduce tasks. Number of partitions = number of reduce tasks = number of files in the HDFS output directory.

- What is the difference between the partition key & the 'sort' key?

> Normally Hadoop's only guarantee is that records with the same key get processed on the same reducer node. However, if we wanted to process multiple keys together on a given reducer, we could use a partition key which determines which reducer note handles a record... but still allows us to sort on something more granular (eg. another field).

In [ ]:

Exercise 2 Tasks:

- **a) read provided code:** In **`PartitionSort/mapper.py`** we've provided a function that will assign a custom partition key (just a letter) to each word. We're going to use this mapper to sort our sample file with 3 partitions. Read this script.
- **b) discus:** How does the mapper decide which partition to assign each record? When you print out the results in what order do you expect to see the records?
- **c) Hadoop job:** Add the required parameters to complete the Hadoop Streaming code below. Your job should partition based on the newly added first key field, and sort alphabetically by word. Run your job. [**`Hint:`** Don't forget to specify the number of fields!]
- **d) discuss:** Examine the output from your job. Compare it to the partitioning function we used. Are the words sorted alphabetically? What is suprising about the partition key that ended up in `part-00000` ?
- **e) code:** If time permits, modify your job so that it sorts the words by *count* instead (still using 3 partitions). To do this you will need to change the partition function in **`PartitionSort/mapper.py`** so that it partitions based on count instead of the first letter of the word. Use 4 and 8 as your cut-points. You will also need to modify one of the Hadoop parameters (which one? why?). Run your job. Are you able to get a total sort? Why/why not?

In [7]:
```
%%writefile PartitionSort/sample.txt
foo     5
quux    9
labs    100
bar     5
qi      1
```

```
Overwriting PartitionSort/sample.txt
```

In [ ]:
```
#### PartitionSort/mapper.txt

* CASE 1:  produces PARTIAL order sorted files Part-0002, Part-0000, Part-0001
* CASE 2:  produces PARTIAL order sorted files Part-0002, Part-0000, Part-0001

``` python
#!/usr/bin/env python
"""
Mapper partitions based on first letter in word.
INPUT:
    word \t count
OUTPUT:
    partitionKey \t word \t count
"""
import re
import sys

def getPartitionKey(word,count):
    """
    First letter of the word
    Helper function to assign partition key ('A', 'B', or 'C').
    Args:  word (str) ; count (int)
    """
    ############ YOUR CODE HERE ##########
    # provided implementation: (run this first, then make your changes in part e)
    # CASE1:  First letter of the word:
    #.        produces PARTIAL order sorted files Part-0002 (keys < h) , Part-0000, Part-0001
    if word[0] < 'h':
        return  'A'
    elif word[0] < 'p':
        return  'B'
    else:
```
```

```
        return 'C'
    ############ (END) YOUR CODE #########

# read from standard input
for line in sys.stdin:
    word, count = line.strip().split()
    count = int(count)
    partitionKey = getPartitionKey(word, count)
    print(f"{partitionKey}\t{word}\t{count}")
```

PartitionSort/mapper.txt

- CASE 1: produces PARTIAL order sorted files Part-0002, Part-0000, Part-0001
- CASE 2 produces a total order sorted files Part-0000 (most freq words), Part-0001, Part-0002 (least frequent words)
- CASE 3: produces a total order sorted files Part-0002 (most freq words), Part-0000, Part-0001 (least frequent words)

```
#!/usr/bin/env python
"""
Mapper partitions based on first letter in word.
INPUT:
    word \t count
OUTPUT:
    partitionKey \t word \t count
"""
import re
import sys

def getPartitionKey(word,count):
    """
    Helper function to assign partition key ('A', 'B', or 'C').
    Args:  word (str) ; count (int)
    """
    ############ YOUR CODE HERE ##########
    # partition based on frequency
    # CASE 2: produces a total order sorted files Part-0000 (most freq words), Part-0001, Part-0002 (least
frequent words)

    if count > 8:                           # <--- SOLUTION --->
        return  'B'                         # <--- SOLUTION --->
```

```
    elif count > 4:                          # <--- SOLUTION --->
        return  'C'                          # <--- SOLUTION --->
    else:    # low freq                      # <--- SOLUTION --->
        return 'A'                           # <--- SOLUTION --->
    #never gets here
    # provided implementation: (run this first, then make your changes in part e)
    # CASE1:  produces PARTIAL order sorted files Part-0002, Part-0000, Part-0001
    if word[0] < 'h':
        return  'A'
    elif word[0] < 'p':
        return  'B'
    else:
        return 'C'
    ############ (END) YOUR CODE #########

# read from standard input
for line in sys.stdin:
    word, count = line.strip().split()
    count = int(count)
    partitionKey = getPartitionKey(word, count)
    print(f"{partitionKey}\t{word}\t{count}")
```

In [ ]:

```
#### PartitionSort/mapper.txt

* CASE 1: produces PARTIAL order sorted files Part-0002, Part-0000, Part-0001
* CASE 2  produces a total order sorted files Part-0000 (most freq words), Part-0001, Part-0002 (least frequent words)
* CASE 3:  produces a total order sorted files Part-0002 (most freq words), Part-0000, Part-0001 (least frequent words)

``` python
#!/usr/bin/env python
"""
Mapper partitions based on first letter in word.
INPUT:
    word \t count
OUTPUT:
    partitionKey \t word \t count
"""
import re
import sys

def getPartitionKey(word,count):
```
```

```
    """
    Helper function to assign partition key ('A', 'B', or 'C').
    Args:  word (str) ; count (int)
    """
    ############ YOUR CODE HERE ##########
    # partition based on frequency
    # CASE 3: produces a total order sorted files Part-0000 (mid freq words), Part-0001, Part-0002 (most frequent words
    # PLEASE verify.... as I may have made a mistake!!

    if count > 8:                           # <--- SOLUTION --->
        return  'A'                         # <--- SOLUTION --->
    elif count > 4:                         # <--- SOLUTION --->
        return  'B'                         # <--- SOLUTION --->
    else:    # low freq                     # <--- SOLUTION --->
        return 'C'                          # <--- SOLUTION --->
    #never gets here
    # CASE 2: produces a total order sorted files Part-0000 (most freq words), Part-0001, Part-0002 (least frequent wor

    if count > 8:                           # <--- SOLUTION --->
        return  'B'                         # <--- SOLUTION --->
    elif count > 4:                         # <--- SOLUTION --->
        return  'C'                         # <--- SOLUTION --->
    else:    # low freq                     # <--- SOLUTION --->
        return 'A'                          # <--- SOLUTION --->
    #never gets here
    # provided implementation: (run this first, then make your changes in part e)
    # CASE1:  produces PARTIAL order sorted files Part-0002, Part-0000, Part-0001
    if word[0] < 'h':
        return  'A'
    elif word[0] < 'p':
        return  'B'
    else:
        return 'C'
    ############ (END) YOUR CODE #########

# read from standard input
for line in sys.stdin:
    word, count = line.strip().split()
    count = int(count)
    partitionKey = getPartitionKey(word, count)
    print(f"{partitionKey}\t{word}\t{count}")
```

In [5]:
```
# part a - complete your work above then RUN THIS CELL AS IS
!chmod a+x PartitionSort/mapper.py
```

In [13]:
```
# unit test
! PartitionSort/mapper.py < PartitionSort/sample.txt
```

```
C       foo     5
B       quux    9
B       labs    100
C       bar     5
A       qi      1
```

In [37]:
```
# load sample file into HDFS (RUN THIS CELL AS IS)
!hdfs dfs -copyFromLocal PartitionSort/sample.txt {HDFS_DIR}
```

In [39]:
```
# parts a - clear output directory (RUN THIS CELL AS IS)
!hdfs dfs -rm -r {HDFS_DIR}/psort-output
```

```
rm: `/user/root/demo3/psort-output': No such file or directory
```

In [47]:
```
# part a - Hadoop streaming command - ADD SORT and PARTITION PARAMETERS HERE
!hadoop jar {JAR_FILE} \




  -files PartitionSort/mapper.py \
  -mapper mapper.py \
  -reducer /bin/cat \

  -input {HDFS_DIR}/sample.txt \
  -output {HDFS_DIR}/psort-output \
  -cmdenv PATH={PATH}\
  -numReduceTasks 3
```

In [40]:
```
# <--- SOLUTION --->
# part a - Hadoop streaming command
!hdfs dfs -rm -r {HDFS_DIR}/psort-output
```

```
!hadoop jar {JAR_FILE} \
  -D stream.num.map.output.key.fields=3 \
  -D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
  -D mapreduce.partition.keycomparator.options="-k3,3nr" \
  -D mapreduce.partition.keypartitioner.options="-k1,1"  \
  -files PartitionSort/mapper.py \
  -mapper mapper.py \
  -reducer /bin/cat \
  -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner \
  -input {HDFS_DIR}/sample.txt \
  -output {HDFS_DIR}/psort-output \
  -cmdenv PATH={PATH}\
  -numReduceTasks 3
```

```
rm: `/user/root/demo3/psort-output': No such file or directory
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar] /tmp/streamjob1257317400051182282.jar tmpDir=null
2022-09-06 18:51:59,702 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.42:8032
2022-09-06 18:51:59,939 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.42:10200
2022-09-06 18:52:00,437 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.42:8032
2022-09-06 18:52:00,438 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.42:10200
2022-09-06 18:52:00,641 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
root/.staging/job_1662400986413_0003
2022-09-06 18:52:01,021 INFO mapred.FileInputFormat: Total input files to process : 1
2022-09-06 18:52:01,026 WARN concurrent.ExecutorHelper: Thread (Thread[GetFileInfo #1,5,main]) interrupted:
java.lang.InterruptedException
        at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:510)
        at com.google.common.util.concurrent.FluentFuture$TrustedFuture.get(FluentFuture.java:88)
        at org.apache.hadoop.util.concurrent.ExecutorHelper.logThrowableFromAfterExecute(ExecutorHelper.java:48)
        at org.apache.hadoop.util.concurrent.HadoopThreadPoolExecutor.afterExecute(HadoopThreadPoolExecutor.java:90)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1157)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:750)
2022-09-06 18:52:01,116 INFO mapreduce.JobSubmitter: number of splits:11
2022-09-06 18:52:01,361 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1662400986413_0003
2022-09-06 18:52:01,362 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-09-06 18:52:01,557 INFO conf.Configuration: resource-types.xml not found
2022-09-06 18:52:01,558 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-09-06 18:52:01,640 INFO impl.YarnClientImpl: Submitted application application_1662400986413_0003
2022-09-06 18:52:01,680 INFO mapreduce.Job: The url to track the job: http://w261-m:8088/proxy/application_1662400986413
_0003/
2022-09-06 18:52:01,682 INFO mapreduce.Job: Running job: job_1662400986413_0003
2022-09-06 18:52:09,856 INFO mapreduce.Job: Job job_1662400986413_0003 running in uber mode : false
2022-09-06 18:52:09,857 INFO mapreduce.Job:  map 0% reduce 0%
2022-09-06 18:52:19,083 INFO mapreduce.Job:  map 18% reduce 0%
2022-09-06 18:52:20,099 INFO mapreduce.Job:  map 27% reduce 0%
2022-09-06 18:52:27,182 INFO mapreduce.Job:  map 45% reduce 0%
```

```
2022-09-06 18:52:28,195 INFO mapreduce.Job:  map 55% reduce 0%
2022-09-06 18:52:34,244 INFO mapreduce.Job:  map 64% reduce 0%
2022-09-06 18:52:35,256 INFO mapreduce.Job:  map 82% reduce 0%
2022-09-06 18:52:40,297 INFO mapreduce.Job:  map 100% reduce 0%
2022-09-06 18:52:49,357 INFO mapreduce.Job:  map 100% reduce 33%
2022-09-06 18:52:50,363 INFO mapreduce.Job:  map 100% reduce 67%
2022-09-06 18:52:51,368 INFO mapreduce.Job:  map 100% reduce 100%
2022-09-06 18:52:52,379 INFO mapreduce.Job: Job job_1662400986413_0003 completed successfully
2022-09-06 18:52:52,472 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=76
                FILE: Number of bytes written=3511924
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1210
                HDFS: Number of bytes written=48
                HDFS: Number of read operations=48
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=9
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed reduce tasks=1
                Launched map tasks=11
                Launched reduce tasks=3
                Data-local map tasks=11
                Total time spent by all maps in occupied slots (ms)=220623336
                Total time spent by all reduces in occupied slots (ms)=51587976
                Total time spent by all map tasks (ms)=69906
                Total time spent by all reduce tasks (ms)=16346
                Total vcore-milliseconds taken by all map tasks=69906
                Total vcore-milliseconds taken by all reduce tasks=16346
                Total megabyte-milliseconds taken by all map tasks=220623336
                Total megabyte-milliseconds taken by all reduce tasks=51587976
        Map-Reduce Framework
                Map input records=5
                Map output records=5
                Map output bytes=48
                Map output materialized bytes=256
                Input split bytes=1012
                Combine input records=0
                Combine output records=0
                Reduce input groups=5
                Reduce shuffle bytes=256
                Reduce input records=5
                Reduce output records=5
                Spilled Records=10
                Shuffled Maps =33
```

```
                Failed Shuffles=0
                Merged Map outputs=33
                GC time elapsed (ms)=2007
                CPU time spent (ms)=18430
                Physical memory (bytes) snapshot=7354474496
                Virtual memory (bytes) snapshot=62204211200
                Total committed heap usage (bytes)=6158286848
                Peak Map Physical memory (bytes)=604532736
                Peak Map Virtual memory (bytes)=4461973504
                Peak Reduce Physical memory (bytes)=315777024
                Peak Reduce Virtual memory (bytes)=4447133696
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=198
        File Output Format Counters
                Bytes Written=48
2022-09-06 18:52:52,473 INFO streaming.StreamJob: Output directory: /user/root/demo3/psort-output
```

In [41]:

```
# part a - Save results locally (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/psort-output/part-0000* > PartitionSort/results.txt
```

In [42]:

```
# part a - view results (RUN THIS CELL AS IS)
!head PartitionSort/results.txt
```

```
B       labs    100
B       quux    9
C       foo     5
C       bar     5
A       qi      1
```

In [43]:

```
# part a - look at first partition (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/psort-output/part-00000
```

```
B       labs    100
B       quux    9
```

In [44]:
```
# part a - look at second partition (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/psort-output/part-00001
```

```
C	foo	5
C	bar	5
```

In [45]:
```
# part a - look at third partition (RUN THIS CELL AS IS)
!hdfs dfs -cat {HDFS_DIR}/psort-output/part-00002
```

```
A	qi	1
```

Expected Output:

part c			part e		
B	labs	100	B	labs	100
C	qi 1		B	quux	9
C	quux	9	C	foo 5	
A	bar 5		C	bar 5	
A	foo 5		A	qi 1	

> **DISCUSSION QUESTIONS (exercise 2 debrief):**
>
> - In the provided implementation how did we assign records to partitions?
> - In part c, why didn't this partitioning result in alphabetically sorted words?
> - Given what you saw in part c, how did you 'trick' Hadoop into doing a full sort (by count) in part e?
> - If you didn't achive the full sort in e why not? On a larger data set what postprocessing would you have to do in this kind of scenario? Is this postprocessing non-trivial?
> - In addition to changing the partition function what other Hadoop parameter did you have to change for part e ?
> - In the real world you wouldn't want your partition key as part of your output. What would we do to avoid this?
> - Does anyone need any additional clarification about any of the Hadoop Streaming?

<--- SOLUTION --->

INSTRUCTOR TALKING POINTS (exercise 2 debrief)

- In the provided implementation how did we assign records to partitions?

> according to the first letter in the word... a-h when in partition 'A', j-o in partition 'B' and p=z in partition 'C'

- In part c, why didn't this partitioning result in alphabetically sorted words?

> We assigned the top(alphabetical) words to partition A, but when Hadoop concatenated the files, it put partition A last. This is not by chance. Hadoop uses a hash function to assign our human readible partition keys to an ordered list of partitions. The ordering that is logical to us, may or may not get preserved in the process. However, its worth noting that this hash function is consistent. i.e. we just say Hadoop order B-C-A, that means it will always order those three keys that way. We can take advantage of this fact to perform a Total Order Sort -- i.e. a sort with multiple paritions that doestn' require any post-processing. Implementing this will be one of the most important question on HW2.

- Given what you saw in part c, how did you 'trick' Hadoop into doing a full sort (by count) in part e?

> Since we knew partition B was going to go first, we put the highest counts there, followed by the 5-9 count words in 'C' and the less than 5 count words in 'A'.

- If you didn't achive the full sort in `e` why not? On a larger data set what postprocessing would you have to do in this kind of scenario? Is this postprocessing non-trivial?

> Not knowing about the hash function you may have simply assigned the top counts to A.. in that case we still have the counts segemented by top/middle/bottom we'd just need to figure out in what order the partitions should be concatenated. This is a lot less work than the merge-sort we needed for postprocessing our secondary sort in break out 4... but with a really large corpus this could still be in-ideal because it requires checking and comparing each partition.

- In addition to changing the partition function what other Hadoop parameter did you have to change for part `e` ?

> the `keycomparator` -- so that it reverse numerical sorts on the 3rd field within each partition

- Does anyone need any additional clarification about any of the Hadoop Streaming parameters -- you will need to be fluent in these for HW2.

Exercise 3: Relative Frequencies Revisited with multiple reducers broadcast total counts to reducers: How?) use COUNTERS to verify

(+ intro to counters)

Now that you know how to use custom partition keys, let's use this techinique to fix the problem from part d in Exercise 1.

First a brief digression... A common challenge when implementing custom partitioning at scale is load balancing: *how can we ensure that each reducer node is doing approximately the same amount of work?* Hadoop's option to define custom counters can be a useful way to monitor this kind of detail. Just like the built in counters that you learned about last week, custom counters are a slight departure from statelessness. Normally we wouldn't want to share a mutable variable across multiple nodes, however in this case the framework manages so that you can increment them from any node without causing a race condition.

To use a custom counter you'll just write an appropriately formatted line to the standard error stream. For example the following line would **increment a counter called 'counter-name' in group 'MyWordCounters' by 10**:

```
sys.stderr.write("reporter:counter:MyWordCounters,counter-name,10\n")
```

This line can be added to your mapper, combiner or reducer scripts and wraped in `if` clauses to increment only under certain conditions. If a counter with that name/group doesn't exsit yet the framework will create one. The values of your custom counters will be printed to the console in their respective groups just like the built in Job Tracking counters. Counters can only be incremented using integers (ie, floats are not supported).

Look in the job output for the counter information:

```
rm: `/user/root/demo3/multipart-output': No such file or directory
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar] /tmp/streamjob4427879317707976565.jar
tmpDir=null
2022-09-06 19:05:39,586 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.42:8032
2022-09-06 19:05:39,799 INFO client.AHSProxy: Connecting to Application History server at w261-
m/10.128.0.42:10200
..........
2022-09-06 19:05:41,463 INFO mapreduce.Job: The url to track the job: http://w261-
m:8088/proxy/application_1662400986413_0004/
2022-09-06 19:05:41,465 INFO mapreduce.Job: Running job: job_1662400986413_0004
2022-09-06 19:05:50,590 INFO mapreduce.Job: Job job_1662400986413_0004 running in uber mode : false
2022-09-06 19:05:50,591 INFO mapreduce.Job:  map 0% reduce 0%
2022-09-06 19:05:59,737 INFO mapreduce.Job:  map 22% reduce 0%
2022-09-06 19:06:00,748 INFO mapreduce.Job:  map 33% reduce 0%
......
```

```
2022-09-06 19:06:30,011 INFO mapreduce.Job: Job job_1662400986413_0004 completed successfully
2022-09-06 19:06:30,117 INFO mapreduce.Job: Counters: 59
    File System Counters
        FILE: Number of bytes read=368754
        FILE: Number of bytes written=4007173
.....
Job Counters
        Killed map tasks=1
        Launched map tasks=9
        Launched reduce tasks=4
...
Total megabyte-milliseconds taken by all reduce tasks=68327400
    Map-Reduce Framework
        Map input records=3761
        Map output records=30600
.......

Peak Reduce Physical memory (bytes)=320868352
        Peak Reduce Virtual memory (bytes)=4448096256
    MyCounters  # MYCOUNTERS!!!!!
        A=8393
        B=6830
        C=6825
        D=8552
......
File Input Format Counters
        Bytes Read=207081
    File Output Format Counters
        Bytes Written=89854
2022-09-06 19:06:30,118 INFO streaming.StreamJob: Output directory: /user/root/demo3/multipart-output
```

Ok, armed with this new tool let's return to the relative frequencies task.

DISCUSSION:

> - Remember the problem we encountered at the end of Exercise 1? How would a custom partition key help us make sure the total counts get sent to each reducer node?

> - What does this do to the number of records being shuffled? Is there a more and a less efficient way to implement it from the perspective of amount of data shuffled?
> - Where would you implement the custom partition key?
> - How would you partition the records? (i.e what criteria would you use to assign keys)

<--- SOLUTION --->

INSTRUCTOR TALKING POINTS (before exercise 2)

- How would a custom partition key help us make sure the total counts get sent to each reducer node?

  > If each partition has a specfic key then we can explicitly send the totals to each one.

- What does this do to the number of records being shuffled?

  > We will be emitting alot more records - x4 for four reducers. But if we are using local aggregation this needn't increase the number of records being shuffled unduly. This would be a great situaion in which to use an in-mapper aggregator.

- Where would you implement the custom partition key?

  > In the first mapper.

- How would you partition the records? (i.e what criteria would you use to assign keys)

  > It doesn't really matter since we don't actually care where the records end up... however it would be smart to try and balance out the amount of work done on each reducer so in this case we could use the alphabet or some other EDA to write a partition function based on the words. Since we're going to use 4 reducers maybe you'd do something like a-g, h-n, o-t,u-z.

Exercise 3 Tasks:

UPDATE Jan 23, 2021 - Combiner has been removed from this exercise. Combiners do not play nice with secondary sort. https://issues.apache.org/jira/browse/MAPREDUCE-3310

- **a) add a custom partitioner:** In `MultiPart/mapper.py` , and `MultiPart/reducer.py` we've copied the base code from the frequencies job. Complete the code in this mapper so that it adds a custom partition key to each record and emits the total subcounts *once for each partition*. Assume we'll be using 4 partitions. Then make any required adjustments to your reducer and combiner to accomodate the new record format (your final output should not include the partition key, though you may want it there initially for debugging purposes).

- **b) discuss:** Keep in mind that each partition still needs the total counts to arrive before the individual word counts. However since you've added a custom partition key to each record, the order inversion trick of adding a special character isn't going to work with Hadoop's default sorting behavior any more. Why not? What will you need to specify in your Hadoop job to make sure that the totals still arrive first?

- **c) unit test & run your job:** Write a few unit tests to debug your scripts. When you have them working as expected run the Hadoop job. [`NOTE` We've provided a few tests to get you started but you'll need to add a unix sort to mimic the sorting you discussed in part 'b'].

- **d) custom counters:** Add custom counter(s) to your reducer code so that you can count how many records are processed by each reduce task. [`TIP` use the partition key as the counter name and python3 string formatting to do this efficiently].

- **e) discuss:** Rerun your Hadoop job and take a look at the custom counter values. What do they tell you? Are these counts the same as the number of keys in the result of each partition? Try changing the partitioning criteria in your mapper... what partitioning results in a really uneven split? what partitioning results in the most even split?

In [46]:
```
# part c - make sure scripts are executable (RUN THIS CELL AS IS)
!chmod a+x MultiPart/mapper.py
!chmod a+x MultiPart/reducer.py
```

MultiPart/mapper.py

```
#!/usr/bin/env python
"""
Mapper script to tokenize words from a line of text.
INPUT:
    a text file
OUTPUT:
    word \t partialCount
"""
import re
import sys
```

```
def getPartitionKey(word):
    "Helper function to assign partition key alphabetically."
    ############## YOUR CODE HERE ##############
    if word[0] < 'g':                                  # <--- SOLUTION --->
        return 'A'                                     # <--- SOLUTION --->
    elif word[0] < 'n':                                # <--- SOLUTION --->
        return 'B'                                     # <--- SOLUTION --->
    elif word[0] <'t':                                 # <--- SOLUTION --->
        return 'C'                                     # <--- SOLUTION --->
    else:                                              # <--- SOLUTION --->
        return 'D'                                     # <--- SOLUTION --->
    ############## (END) YOUR CODE ##############

# initialize local aggregator for total word count
TOTAL_WORDS = 0
# read from standard input
for line in sys.stdin:
    line = line.strip()
    # tokenize
    words = re.findall(r'[a-z]+', line.lower())
    # emit words and increment total counter
    for word in words:
        TOTAL_WORDS += 1
        pkey = getPartitionKey(word)
        print(f'{pkey}\t{word}\t{1}')

# emit total count to each partition (note this is a partial total)
############## YOUR CODE HERE ##############
for pkey in ['A','B','C','D']:                         # <--- SOLUTION --->
        print(f'{pkey}\t!total\t{TOTAL_WORDS}')        # <--- SOLUTION --->
############## (END) YOUR CODE ##############
```

MultiPart/reducer.py

```
#!/usr/bin/env python
"""
Reducer script to add counts with the same key and
divide by total count to get relative frequency.
```

```
INPUT:
    word \t partialCount
OUTPUT:
    word \t totalCount
"""
import sys

# initialize trackers
cur_word = None
cur_count = 0
total = 0

# read input key-value pairs from standard input
for line in sys.stdin:
    ########## UNCOMMENT & MODIFY AS NEEDED BELOW #########
#    key, value = line.split()
#    # tally counts from current key
#    if key == cur_word:
#        cur_count += int(value)
#    # OR ...
#    else:
#        # store word count total
#        if cur_word == '!total':
#            total = float(cur_count)
#        # emit realtive frequency
#        if cur_word:
#            print(f'{cur_word}\t{cur_count/total}')
#        # and start a new tally
#        cur_word, cur_count  = key, int(value)
#
## don't forget the last record!
#print(f'{cur_word}\t{cur_count/total}')

    part, key, value = line.split()                # <--- SOLUTION --->
    # USE counters to see if everything is tallied correctly
    #     wehn you run the Hadoop, look at the outpur from the job for the COUNTER info
    sys.stderr.write(f'reporter:counter:MyCounters,{part},1\n')  # <--- SOLUTION --->
    # tally counts from current key                # <--- SOLUTION --->
    if key == cur_word:                            # <--- SOLUTION --->
```

```
            cur_count += int(value)                    # <--- SOLUTION --->
        # OR ...                                       # <--- SOLUTION --->
        else:                                          # <--- SOLUTION --->
            # store word count total                   # <--- SOLUTION --->
            if cur_word == '!total':                   # <--- SOLUTION --->
                total = float(cur_count)               # <--- SOLUTION --->
            # emit realtive frequency                  # <--- SOLUTION --->
            if cur_word and cur_word != '!total':      # <--- SOLUTION --->
                print(f'{cur_word}\t{cur_count/total}') # <--- SOLUTION --->
            # and start a new tally                    # <--- SOLUTION --->
            cur_word, cur_count  = key, int(value)     # <--- SOLUTION --->
                                                       # <--- SOLUTION --->
    ## don't forget the last record!                   # <--- SOLUTION --->
    print(f'{cur_word}\t{cur_count/total}')            # <--- SOLUTION --->
```

In [47]:

```
# part c - unit test mapper script
!echo "foo foo quux labs foo bar quux" | MultiPart/mapper.py
```

```
A       foo     1
A       foo     1
C       quux    1
B       labs    1
A       foo     1
A       bar     1
C       quux    1
A       !total  7
B       !total  7
C       !total  7
D       !total  7
```

In [48]:

```
# part c - unit test map-combine-reduce (ADJUST SORT AS NEEDED)
!echo "foo foo quux labs foo bar quux" | MultiPart/mapper.py | sort -k1,1 | MultiPart/reducer.py
```

```
reporter:counter:MyCounters,A,1
reporter:counter:MyCounters,A,1
reporter:counter:MyCounters,A,1
bar     0.14285714285714285
reporter:counter:MyCounters,A,1
reporter:counter:MyCounters,A,1
reporter:counter:MyCounters,B,1
foo     0.42857142857142855
reporter:counter:MyCounters,B,1
```

```
reporter:counter:MyCounters,C,1
labs    0.14285714285714285
reporter:counter:MyCounters,C,1
reporter:counter:MyCounters,C,1
reporter:counter:MyCounters,D,1
quux    0.2857142857142857
!total  1.0
```

In [49]:

```
# parts c - clear the output directory (RUN THIS CELL AS IS)
!hdfs dfs -rm -r {HDFS_DIR}/multipart-output
# NOTE: this directory won't exist unless you are re-running a job, that's fine.
```

```
rm: `/user/root/demo3/multipart-output': No such file or directory
```

In [50]:

```
# part c - Hadoop streaming command - FILL IN HERE (don't forget to specify 4 reducers)
```

In [51]:

```
# <--- SOLUTION --->
# https://issues.apache.org/jira/browse/MAPREDUCE-3310 combiners don't play nice with secondary sort!
# part c - Hadoop streaming command
!hdfs dfs -rm -r {HDFS_DIR}/multipart-output
!hadoop jar {JAR_FILE} \
  -D stream.num.map.output.key.fields=3 \
  -D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
  -D mapreduce.partition.keycomparator.options="-k2,2" \
  -D mapreduce.partition.keypartitioner.options="-k1,1"  \
  -files MultiPart/mapper.py,MultiPart/combiner.py,MultiPart/reducer.py \
  -mapper mapper.py \
  -reducer reducer.py \
  -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner \
  -input {HDFS_DIR}/alice.txt \
  -output {HDFS_DIR}/multipart-output \
  -cmdenv PATH={PATH} \
  -numReduceTasks 4
```

```
rm: `/user/root/demo3/multipart-output': No such file or directory
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar] /tmp/streamjob4427879317707976565.jar tmpDir=null
2022-09-06 19:05:39,586 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.42:8032
2022-09-06 19:05:39,799 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.42:10200
2022-09-06 19:05:40,270 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.42:8032
2022-09-06 19:05:40,270 INFO client.AHSProxy: Connecting to Application History server at w261-m/10.128.0.42:10200
2022-09-06 19:05:40,473 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
root/.staging/job_1662400986413_0004
```

```
2022-09-06 19:05:40,857 INFO mapred.FileInputFormat: Total input files to process : 1
2022-09-06 19:05:40,927 INFO mapreduce.JobSubmitter: number of splits:9
2022-09-06 19:05:41,139 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1662400986413_0004
2022-09-06 19:05:41,140 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-09-06 19:05:41,322 INFO conf.Configuration: resource-types.xml not found
2022-09-06 19:05:41,323 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-09-06 19:05:41,408 INFO impl.YarnClientImpl: Submitted application application_1662400986413_0004
2022-09-06 19:05:41,463 INFO mapreduce.Job: The url to track the job: http://w261-m:8088/proxy/application_1662400986413
_0004/
2022-09-06 19:05:41,465 INFO mapreduce.Job: Running job: job_1662400986413_0004
2022-09-06 19:05:50,590 INFO mapreduce.Job: Job job_1662400986413_0004 running in uber mode : false
2022-09-06 19:05:50,591 INFO mapreduce.Job:  map 0% reduce 0%
2022-09-06 19:05:59,737 INFO mapreduce.Job:  map 22% reduce 0%
2022-09-06 19:06:00,748 INFO mapreduce.Job:  map 33% reduce 0%
2022-09-06 19:06:06,819 INFO mapreduce.Job:  map 56% reduce 0%
2022-09-06 19:06:08,837 INFO mapreduce.Job:  map 67% reduce 0%
2022-09-06 19:06:13,882 INFO mapreduce.Job:  map 78% reduce 0%
2022-09-06 19:06:14,891 INFO mapreduce.Job:  map 89% reduce 0%
2022-09-06 19:06:15,900 INFO mapreduce.Job:  map 100% reduce 0%
2022-09-06 19:06:22,957 INFO mapreduce.Job:  map 100% reduce 25%
2022-09-06 19:06:24,972 INFO mapreduce.Job:  map 100% reduce 50%
2022-09-06 19:06:25,976 INFO mapreduce.Job:  map 100% reduce 75%
2022-09-06 19:06:28,995 INFO mapreduce.Job:  map 100% reduce 100%
2022-09-06 19:06:30,011 INFO mapreduce.Job: Job job_1662400986413_0004 completed successfully
2022-09-06 19:06:30,117 INFO mapreduce.Job: Counters: 59
        File System Counters
                FILE: Number of bytes read=368754
                FILE: Number of bytes written=4007173
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=207900
                HDFS: Number of bytes written=89854
                HDFS: Number of read operations=47
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=12
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=1
                Launched map tasks=9
                Launched reduce tasks=4
                Data-local map tasks=9
                Total time spent by all maps in occupied slots (ms)=194011944
                Total time spent by all reduces in occupied slots (ms)=68327400
                Total time spent by all map tasks (ms)=61474
                Total time spent by all reduce tasks (ms)=21650
                Total vcore-milliseconds taken by all map tasks=61474
                Total vcore-milliseconds taken by all reduce tasks=21650
```

```
                Total megabyte-milliseconds taken by all map tasks=194011944
                Total megabyte-milliseconds taken by all reduce tasks=68327400
        Map-Reduce Framework
                Map input records=3761
                Map output records=30600
                Map output bytes=307530
                Map output materialized bytes=368946
                Input split bytes=819
                Combine input records=0
                Combine output records=0
                Reduce input groups=3042
                Reduce shuffle bytes=368946
                Reduce input records=30600
                Reduce output records=3006
                Spilled Records=61200
                Shuffled Maps =36
                Failed Shuffles=0
                Merged Map outputs=36
                GC time elapsed (ms)=1722
                CPU time spent (ms)=21830
                Physical memory (bytes) snapshot=6595514368
                Virtual memory (bytes) snapshot=57785286656
                Total committed heap usage (bytes)=5870977024
                Peak Map Physical memory (bytes)=622084096
                Peak Map Virtual memory (bytes)=4457984000
                Peak Reduce Physical memory (bytes)=320868352
                Peak Reduce Virtual memory (bytes)=4448096256
        MyCounters
                A=8393
                B=6830
                C=6825
                D=8552
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=207081
        File Output Format Counters
                Bytes Written=89854
2022-09-06 19:06:30,118 INFO streaming.StreamJob: Output directory: /user/root/demo3/multipart-output
```

In [ ]:

My counters appear below

```
rm: `/user/root/demo3/multipart-output': No such file or directory
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.2.3.jar] /tmp/streamjob4427879317707976565.jar
tmpDir=null
2022-09-06 19:05:39,586 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.42:8032
2022-09-06 19:05:39,799 INFO client.AHSProxy: Connecting to Application History server at w261-
m/10.128.0.42:10200
2022-09-06 19:05:40,270 INFO client.RMProxy: Connecting to ResourceManager at w261-m/10.128.0.42:8032
2022-09-06 19:05:40,270 INFO client.AHSProxy: Connecting to Application History server at w261-
m/10.128.0.42:10200
2022-09-06 19:05:40,473 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-
yarn/staging/root/.staging/job_1662400986413_0004
2022-09-06 19:05:40,857 INFO mapred.FileInputFormat: Total input files to process : 1
2022-09-06 19:05:40,927 INFO mapreduce.JobSubmitter: number of splits:9
2022-09-06 19:05:41,139 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1662400986413_0004
2022-09-06 19:05:41,140 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-09-06 19:05:41,322 INFO /gateway/default/node/conf?host&port.Configuration: resource-types.xml not
found
2022-09-06 19:05:41,323 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-09-06 19:05:41,408 INFO impl.YarnClientImpl: Submitted application application_1662400986413_0004
2022-09-06 19:05:41,463 INFO mapreduce.Job: The url to track the job: http://w261-
m:8088/proxy/application_1662400986413_0004/
2022-09-06 19:05:41,465 INFO mapreduce.Job: Running job: job_1662400986413_0004
2022-09-06 19:05:50,590 INFO mapreduce.Job: Job job_1662400986413_0004 running in uber mode : false
2022-09-06 19:05:50,591 INFO mapreduce.Job:  map 0% reduce 0%
2022-09-06 19:05:59,737 INFO mapreduce.Job:  map 22% reduce 0%
2022-09-06 19:06:00,748 INFO mapreduce.Job:  map 33% reduce 0%
......


2022-09-06 19:06:30,011 INFO mapreduce.Job: Job job_1662400986413_0004 completed successfully
2022-09-06 19:06:30,117 INFO mapreduce.Job: Counters: 59
    File System Counters
        FILE: Number of bytes read=368754
        FILE: Number of bytes written=4007173
.....
Job Counters
        Killed map tasks=1
```

```
        Launched map tasks=9
        Launched reduce tasks=4
...
Total megabyte-milliseconds taken by all reduce tasks=68327400
    Map-Reduce Framework
        Map input records=3761
        Map output records=30600
.......

Peak Reduce Physical memory (bytes)=320868352
        Peak Reduce Virtual memory (bytes)=4448096256
    MyCounters  # MYCOUNTERS!!!!!
        A=8393
        B=6830
        C=6825
        D=8552
    Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
        Bytes Read=207081
    File Output Format Counters
        Bytes Written=89854
2022-09-06 19:06:30,118 INFO streaming.StreamJob: Output directory: /user/root/demo3/multipart-output
```

In [52]:

```
#  part c - take a look at a few records from each partition (RUN THIS CELL AS IS)
for p in range(4):
    print('='*10,f'PARTITION{p+1}','='*10)
    !hdfs dfs -cat {HDFS_DIR}/multipart-output/part-0000{p} | head -n 5
```

```
========== PARTITION1 ==========
t       0.007132574270383458
table   0.0005889281507656066
tail    0.0002944640753828033
tails   9.815469179426777e-05
take    0.0007198010731579636
```

```
cat: Unable to write to output stream.
========== PARTITION2 ==========
a       0.022739170265672033
abide   6.543646119617851e-05
able    3.2718230598089256e-05
about   0.003337259521005104
above   9.815469179426777e-05
cat: Unable to write to output stream.
========== PARTITION3 ==========
gained  3.2718230598089256e-05
gallons 3.2718230598089256e-05
game    0.0004253369977751603
games   3.2718230598089256e-05
garden  0.0005234916895694281
cat: Unable to write to output stream.
========== PARTITION4 ==========
name    0.0003599005365789818
named   3.2718230598089256e-05
names   6.543646119617851e-05
narrow  6.543646119617851e-05
nasty   3.2718230598089256e-05
cat: Unable to write to output stream.
```

Expected Results:

NOTE: exact partition splits depend on your custom function:

Test file:

partition 1	partition 2	partition 3	partition 4
test 0.15	a 0.1	has 0.05	
this 0.15	file 0.15	is 0.1	small 0.15
two 0.05	for 0.05	lines 0.05	

Full Alice Text:

partition 1		partition 2	
t	0.007119994774315762	a	0.022699065908942453
table	0.0005878894767783657	abide	6.532105297537396e-05
tail	0.00029394473838918284	able	3.266052648768698e-05
tails	9.798157946306095e-05	about	0.003331373701744072
take	0.0007185315827291136	above	9.798157946306095e-05

partition 3	partition 4

```
gained  3.266052648768698e-05   name    0.0003592657913645568
gallons 3.266052648768698e-05   named   3.266052648768698e-05
game    0.0004245868443399308   names   6.532105297537396e-05
games   3.266052648768698e-05   narrow  6.532105297537396e-05
garden  0.0005225684238029917   nasty   3.266052648768698e-05
```

> **DISCUSSION**
>
> - After adding the custom partition key what else did you have to do so that the order inversion pattern would still work?
> - In part `d` what did you see from your custom counters? Do these numbers match the number of keys in the result of each partition? Why/why not?
> - What other partitioning cuts did you try? What partitioning results in a really uneven split? what partitioning results in the most even split?
> - Do custom counters solve the load balancing challenge?
> - If we wanted to design a subsequent job to sort these results from highest to lowest frequency what partitioning strategy would you explore? Any particular challenge there? (**`HINT:`** you may wish to consider the kinds of numbers you saw in the results from Exercise 1, how python stores floats, and what you know about word frequencies in natural language).

<--- SOLUTION --->

INSTRUCTOR TALKING POINTS (after exercise 3)

- After adding the custom partition key what else did you have to do so that the order inversion pattern would still work?

  > We need to add a secondary sort.

- In part `d` what did you see from your custom counters? Do these numbers match the number of keys in the result of each partition? Why/why not?

  > Results will vary for counts depending on the partition students choose & the ammount of combining. We saw:
  >
  > ```
  > A=1391
  > B=845
  > C=1170
  > D=633
  > ```

> Note that these are not the same as the number of resulting lines in each partition. Some words eg. 'the' will occur many more times than others, so even if we divide the alphabet evenly that may not be an even split of word's processed. Use this opportunity to reinforce the fact that Hadoop did not put partition 'A' first & remind students about the hash function. They'll implement a Total Order Sort in HW2 where they reverse engineer that hash function to pre-order the partition keys.

- What other partitioning cuts did you try? What partitioning results in a really uneven split? what partitioning results in the most even split?

> Results will vary. Give students an opportunity to share & then discuss their reasoning.

- Do custom counters solve the load balancing challenge?

> No, they can help us determine whether we have a load balancing problem in the first place, but solving that problem if it exists requires some outside understading of our data distribution. That's usually one of the key goals behind our EDA. With really large datasets we'll also often do this EDA on a random sample instead of the full data -- we'll talk more about that in week 5.

- If we wanted to design a subsequent job to sort these results from highest to lowest frequency what partitioning strategy would you explore? Any particular challenge there? (**`HINT:`** you may wish to consider the kinds of numbers you saw in the results from Exercise 1, how python stores floats, and what you know about word frequencies in natural language).

> We'd need to partition based on the relative frequencies but that is a challenge because the numbers are very very small and we're going to run into floating point errors. Another challenge is that this is going to be a inverse power law distribution --- lots of words which occur very few times & whose relative frequencies are clustered around 0. We could try a log-transformation to help us partition... but in practice we'd want to use EDA on specific words to design a partition strategy.

Exercise 4: Unordered Total Sort vs Total Order Sort (sort with multiple reducers): with precalculated partition function vs. on the fly

In HW1 we emphasized that sorting can be a useful preprocessing tool because sorted input can sometimes facilitate a more efficient algorithm design. However sorting is also often desirable in its own right. For example suppose you wanted to get the top and bottom 100 words according to their relative frequencies? The best way to get these results would be to do a Total Order Sort -- that is to sort the entire data set from highest to lowest so that you can simply read the first 100 records in the first partition and last 100 records in the last partition.

So far we've tried two different ways of sorting using the Hadoop framework and neither quite achieved this result to satisfaction. First, in last week's breakout we performed a secondary sort using the `keycomparator` and accompanying Hadoop parameters. When we used a single reducer that strategy *did* successfully result in a sort from top to bottom, but in a class all about scaling up for large data solutions that require a single reducer won't get us very far. Unfortunately when we tried using those same three sorting parameters and with multiple reducers we ended up with results sorted within each partition but not across partitions. This is what we call that a *partial sort* because records are only ordered relative to the other keys that happen to be processed by the same reducer. Importantly a partial sort is of no use at all for us if we wanted to find the top 100 and bottom 100 from our relative frequencies file.

Then in Exercise 2 of this notebook we learned how to specify a custom partition key so that we can explicitly control the partitioning. By combining the use of a custom partition key and the secondary sort from last week we were able to ensure that our results were not only sorted within their paritions but also that the paritioning grouped records with similar value ranges together. In theory this should have solved our 'total sort with multiple reducers' challenge but in practice something odd happened: the records with the top value didn't reliably end up in the first partition nor did the records with the lowest values necessarily end up in the last. The partition files themselves were out of order, making this an *unordered total sort*.

The reason that the partitions appear out of order has to do with the hash function that Hadoop applies to your custom partition keys. That hash function results in an ordering that may not match the human readible string or integer numbering you as a programmer might have intended. Luckily, this hash function is both fixed and known -- so for example, if you are using partition keys `A`, `B`, and `C` your partitions will always end up ordered `B - C - A` (as you saw in Exercise 2). That means if we know in advance how many partitions (i.e. reducers) we plan to use, we can reverse engineer the hash function to figure out how Hadoop will order those keys and plan accordingly (for example in our 3 partition case by specifying that the top values should get the partition key `B` not `A`). When your data is sorted not only within each partition but the partitions are also in the right order, you've achieved a *total order sort*.

Let's give it a try! Your job in this exercise is to write a Hadoop Job that will perform a total order sort on the output of Exercise 1 (relative frequencies) with any number of partitions.

Note: Part III.D.4 in the Total Order Sort notebook is a necessary and useful read. It located on your private storage bucket in GCS/HelpfulResources/TotalSortGuide/total-sort-guide-hadoop-streaming.ipynb. It has a much more comprehensive explanation that you may wish to reference here.

Exercise 4 Tasks

- **a) discuss:** In the `%%writefile` cell below we create a partition file (**`partitions.txt`**) which contains the cut points we'll use to partition the data. Based on your reading of this file, how many partitions will we use? Think about the kinds of values that we got in Exercise 1 ... can you see any potential problems with these cutpoints?

- **b) discuss:** Read through **`TotalOrderSort/mapper.py`** and **`TotalOrderSort/reducer.py`** . Which of these scripts makes use of our partition file? What does the mapper do? What does the reducer do?

- **c) code:** Run the provided code below to apply this mapper and reducer. Note how the Hadoop job reads directly from the output directory we created in Exercise 1 (you will need to be those results were computed inorder to run this job). Use the provided code to view the output and confirm that the current implementation performs an unordered sort. What adjustments might you want to make to the partition file? (go ahead and modify it as desired). Then modify the Hadoop job so that it uses `/bin/cat` instead of `reducer.py` ... and re-run the job this will allow you to see the order in which the partition keys were sorted.

- **d) discuss:** Read through **`TotalOrderSort/TOS_mapper.py`** . Pay particular attention to the helper function `makeKeyHash()` -- what does this function do? how is it used? how is this mapper different that the original one?

- **e) code + discussion:** Replace the original mapper with this new mapper(*don't forget to add it to the `-files` line*) and rerun the job (still with `/bin/cat` as reducer). How did the parition ordering change? What happpens if you change the partition file so that it has one extra number? Does your job work? Re-place the true reducer. Et voila, you have achieved total order sort!

<--- SOLUTION --->

INSTRUCTOR TALKING POINTS

> **a)** 5 partitions. Most relative frequencies will be very low decimals so most of the data is going to end up in the last few partitions. We can fix this by performing EDA on the input file and determining a way to balance the load across all 5 reducers. In this case we might use cutpoints more like: 0.5,0.3,0.2,0.1,0.
>
> **b)** The mapper reads the partition file and uses those cut points to assign letters for partition keys (eg. A, B, C, etc). The reducer simply removes these partition keys.
>
> **d)** `makeKeyHash` allows us to sort the parition keys in the order that Hadoop will, this different ordering of partition keys is the only difference between this mapper and the last.

In [15]:
```
%%writefile TotalOrderSort/partitions.txt
0.01,0.005,0.003,0.002,0.001,0
```

```
Writing TotalOrderSort/partitions.txt
```

In [19]:
```
import numpy as np
import sys
from operator import itemgetter
import os.path

# helper function
def getPartitionsFromFile(fpath='TotalOrderSort/partitions.txt'):
    """
    Args:   partition file path
    Returns:    partition_keys (sorted list of strings)
                partition_values (descending list of floats)

    NOTE 1: make sure the partition file gets passed into Hadoop
    """
    # load in the partition values from file
    assert os.path.isfile(fpath), 'ERROR with partition file'
    with open(fpath,'r') as f:
        vals = f.read()
    partition_cuts = sorted([float(v) for v in vals.split(',')], reverse=True)

    # use the first N uppercase letters as custom partition keys
    N = len(partition_cuts)
    KEYS = list(map(chr, range(ord('A'), ord('Z')+1)))[:N]
    partition_keys = sorted(KEYS)

    return partition_keys, partition_cuts

# call your helper function to get partition keys & cutpoints
pKeys, pCuts = getPartitionsFromFile()

list(zip(pKeys, pCuts))
```

Out[19]:
```
[('A', 0.01),
 ('B', 0.005),
 ('C', 0.003),
 ('D', 0.002),
```

```
 ('E', 0.001),
 ('F', 0.0)]
```

This looks great BUT when we hash the partition keys something goes wrong:

Using a partition keys ['A', 'B', 'C', 'D', 'E', 'F'] leads to an output directory of PART files:

- PART-000005 PART-00000.....PART-00004

I.e.,

- parition key ['A', 'B', 'C', 'D', 'E', 'F'] --> PART-00005 PART-0000.....PART-0004

In summary the following partition does not work:

```
[('A', 0.01),  --> PART-00005
 ('B', 0.005), --> PART-00000
 ('C', 0.003), --> PART-00001
 ('D', 0.002), --> PART-00002
 ('E', 0.001), --> PART-00003
 ('F', 0.0)]   --> PART-00004
```

To make the order in desceasing frequency (total order sort) we will need to provide the following partition mapping:

```
[('B', 0.01),  --> PART-00000
 ('C', 0.005), --> PART-00001
 ('D', 0.003), --> PART-00002
 ('E', 0.002), --> PART-00003
 ('F', 0.001), --> PART-00004
 ('A', 0.0)]   --> PART-00005
```

inverse hashCode function

In order to preserve partition key ordering across the output partition files, we need to use an `inverse hashCode function` , which takes as input the desired partition index and total number of partitions, and returns a sorted list partition key that correspond to the output parition files PART-00000.....PART-00005. This key, when supplied to the Hadoop framework (KeyBasedPartitioner), will hash to the correct partition index (and reducer) so that key order is preserved across all PART-000* files.

E.g., for a six-reducer job the following partitions keys will need to be used to ensure that the key order is preserved across all PART-000* files

- parition key ['B', 'C', 'D', 'E', 'F', 'A'] --> PART-00000.....PART-00005

In [22]:

```
#!/usr/bin/env python
"""
INPUT:
    count \t word
OUTPUT:
    partitionKey \t count \t word
"""

import os
import re
import sys
import numpy as np
from operator import itemgetter


N = int(os.getenv('mapreduce_job_reduces', default=1))

def makeIndex(key, num_reducers = N):
    """
    Mimic the Hadoop string-hash function.

    key             the key that will be used for partitioning
    num_reducers    the number of reducers that will be configured
    """
    byteof = lambda char: int(format(ord(char), 'b'), 2)
    current_hash = 0
    for c in key:
        current_hash = (current_hash * 31 + byteof(c))
    return current_hash % num_reducers

def makeKeyFile(num_reducers = N):
    KEYS = list(map(chr, range(ord('A'), ord('Z')+1)))[:num_reducers]
    partition_keys = sorted(KEYS, key=lambda k: makeIndex(k,num_reducers))

    return partition_keys

# call your helper function to get partition keys
pKeys = makeKeyFile(6)
pKeys  # parition key ['B', 'C', 'D', 'E', 'F', 'A'] --> PART-00000.....PART-00005
```

Out[22]: ['B', 'C', 'D', 'E', 'F', 'A']

In [ ]:

```
%%writefile multipleReducerTotalOrderSort_mapper.py
#!/usr/bin/env python
"""
INPUT:
    count \t word
OUTPUT:
    partitionKey \t count \t word
"""

import os
import re
import sys
import numpy as np
from operator import itemgetter


N = int(os.getenv('mapreduce_job_reduces', default=1))

def makeIndex(key, num_reducers = N):
    """
    Mimic the Hadoop string-hash function.

    key             the key that will be used for partitioning
    num_reducers    the number of reducers that will be configured
    """
    byteof = lambda char: int(format(ord(char), 'b'), 2)
    current_hash = 0
    for c in key:
        current_hash = (current_hash * 31 + byteof(c))
    return current_hash % num_reducers

def makeKeyFile(num_reducers = N):
    """ Keys A - Z (26 characters only"""
    KEYS = list(map(chr, range(ord('A'), ord('Z')+1)))[:num_reducers]
    partition_keys = sorted(KEYS, key=lambda k: makeIndex(k,num_reducers))

    return partition_keys


# call your helper function to get partition keys
pKeys = makeKeyFile()
```

```
def makePartitionFile():
    # returns a list of split points
    # For the sake of simplicity this is hardcoded.
    # See the sampling section below for more information.
    return [20,10,0]

pFile = makePartitionFile()

### Mapper starts on each input record
###
for line in sys.stdin:
    line = line.strip()
    key,value = line.split('\t')

    for idx in range(N):
        if float(key) > pFile[idx]:
            print(str(pKeys[idx])+"\t"+key+"\t"+value)
            break
```

In [ ]:

```
%%writefile PartitionSort/sample.txt
foo     5
quux    9
labs    100
bar     5
qi      1
```

In [54]:

```
# part c - make sure files are executable (RUN THIS CELL AS IS)
!chmod a+x TotalOrderSort/mapper.py
!chmod a+x TotalOrderSort/reducer.py
!chmod a+x TotalOrderSort/TOS_mapper.py
```

In [16]:

```
!cat TotalOrderSort/partitions.txt
```

```
0.01,0.005,0.003,0.002,0.001,0
```

In [48]:

```
# part c - hadoop job (RUN THIS CELL AS IS)
!hdfs dfs -rm -r {HDFS_DIR}/tos-output
!hadoop jar {JAR_FILE} \
```

```
  -D stream.num.map.output.key.fields=3 \
  -D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
  -D mapreduce.partition.keypartitioner.options="-k1,1" \
  -D mapreduce.partition.keycomparator.options="-k3,3nr" \
  -files TotalOrderSort/mapper.py,TotalOrderSort/reducer.py,TotalOrderSort/partitions.txt \
  -mapper mapper.py \
  -reducer reducer.py \
  -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner \
  -input {HDFS_DIR}/frequencies-output \
  -output {HDFS_DIR}/tos-output \
  -cmdenv PATH={PATH} \
  -numReduceTasks 6
```

```
Deleted /user/root/demo3/tos-output
packageJobJar: [] [/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.15.0.jar] /tmp/streamjob3305256276489392893.jar
tmpDir=null
18/09/19 07:41:23 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/09/19 07:41:23 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/09/19 07:41:24 INFO mapred.FileInputFormat: Total input paths to process : 1
18/09/19 07:41:24 INFO mapreduce.JobSubmitter: number of splits:2
18/09/19 07:41:25 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1537278954088_0020
18/09/19 07:41:25 INFO impl.YarnClientImpl: Submitted application application_1537278954088_0020
18/09/19 07:41:25 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_153727
8954088_0020/
18/09/19 07:41:25 INFO mapreduce.Job: Running job: job_1537278954088_0020
18/09/19 07:41:33 INFO mapreduce.Job: Job job_1537278954088_0020 running in uber mode : false
18/09/19 07:41:33 INFO mapreduce.Job:  map 0% reduce 0%
18/09/19 07:41:44 INFO mapreduce.Job:  map 50% reduce 0%
18/09/19 07:41:45 INFO mapreduce.Job:  map 100% reduce 0%
18/09/19 07:41:58 INFO mapreduce.Job:  map 100% reduce 17%
18/09/19 07:42:01 INFO mapreduce.Job:  map 100% reduce 33%
18/09/19 07:42:04 INFO mapreduce.Job:  map 100% reduce 50%
18/09/19 07:42:05 INFO mapreduce.Job:  map 100% reduce 67%
18/09/19 07:42:06 INFO mapreduce.Job:  map 100% reduce 100%
18/09/19 07:42:07 INFO mapreduce.Job: Job job_1537278954088_0020 completed successfully
18/09/19 07:42:07 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=103005
                FILE: Number of bytes written=1412020
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=92273
                HDFS: Number of bytes written=96949
                HDFS: Number of read operations=24
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=12
```

```
		Job Counters 
			Killed reduce tasks=1
			Launched map tasks=2
			Launched reduce tasks=6
			Data-local map tasks=2
			Total time spent by all maps in occupied slots (ms)=14063
			Total time spent by all reduces in occupied slots (ms)=87305
			Total time spent by all map tasks (ms)=14063
			Total time spent by all reduce tasks (ms)=87305
			Total vcore-milliseconds taken by all map tasks=14063
			Total vcore-milliseconds taken by all reduce tasks=87305
			Total megabyte-milliseconds taken by all map tasks=14400512
			Total megabyte-milliseconds taken by all reduce tasks=89400320
		Map-Reduce Framework
			Map input records=3010
			Map output records=3010
			Map output bytes=96949
			Map output materialized bytes=103041
			Input split bytes=258
			Combine input records=0
			Combine output records=0
			Reduce input groups=3010
			Reduce shuffle bytes=103041
			Reduce input records=3010
			Reduce output records=3010
			Spilled Records=6020
			Shuffled Maps =12
			Failed Shuffles=0
			Merged Map outputs=12
			GC time elapsed (ms)=787
			CPU time spent (ms)=14480
			Physical memory (bytes) snapshot=1715482624
			Virtual memory (bytes) snapshot=10984353792
			Total committed heap usage (bytes)=1302331392
		Shuffle Errors
			BAD_ID=0
			CONNECTION=0
			IO_ERROR=0
			WRONG_LENGTH=0
			WRONG_MAP=0
			WRONG_REDUCE=0
		File Input Format Counters 
			Bytes Read=92015
		File Output Format Counters 
			Bytes Written=96949
18/09/19 07:42:07 INFO streaming.StreamJob: Output directory: /user/root/demo3/tos-output
```

In [ ]:

```
# part c - print top words in each class (RUN THIS CELL AS IS)
for idx in range(6):
    print(f"\n============== PART-0000{idx}===============")
    numLines = !hdfs dfs -cat {HDFS_DIR}/tos-output/part-0000{idx} | wc -l
    print(f"Number of lines processed by this reducer: {numLines[0]}\n")
    !hdfs dfs -cat {HDFS_DIR}/tos-output/part-0000{idx} | head | column -t
```

Expected output

```
============== PART-00000===============
Number of lines processed by this reducer: 17

as    0.008964795183876457
her   0.008114121188326136
with  0.00745975657636435
at    0.007361601884570083
s     0.0072634471927758145
t     0.007132574270383458
on    0.006674519042010208
all   0.006543646119617851
this  0.0059219997382541556
for   0.005856563277057977

============== PART-00001===============
Number of lines processed by this reducer: 17

very    0.004744143436722942
what    0.0046459887449286745
is      0.004286088208349692
little  0.004220651747153514
he      0.004187933516555425
if      0.003860751210574532
out     0.003860751210574532
one     0.003468132443397461
up      0.0033699777516031934
down    0.0033699777516031934

============== PART-00002===============
```

```
Number of lines processed by this reducer: 31

project  0.0028792042926318543
them     0.0028792042926318543
know     0.0028792042926318543
have     0.0028464860620337653
by       0.002813767831435676
like     0.0027810496008375866
were     0.0027810496008375866
went     0.002715613139641408
would    0.002715613139641408
again    0.002715613139641408

============== PART-00003===============
Number of lines processed by this reducer: 77

don     0.0019958120664834446
turtle  0.0019630938358853552
its     0.0019630938358853552
now     0.0019630938358853552
my      0.001897657374689177
way     0.001897657374689177
began   0.001897657374689177
other   0.0018649391440910875
tm      0.0018649391440910875
ll      0.0018649391440910875

============== PART-00004===============
Number of lines processed by this reducer: 2850

toes       9.815469179426777e-05
yours      9.815469179426777e-05
yesterday  9.815469179426777e-05
wrote      9.815469179426777e-05
worse      9.815469179426777e-05
whisper    9.815469179426777e-05
whiskers   9.815469179426777e-05
whatever   9.815469179426777e-05
week       9.815469179426777e-05
```

```
watching   9.815469179426777e-05
cat: Unable to write to output stream.

============== PART-00005===============     !!!!  BIG PROBLEM !!!! not in order
Number of lines processed by this reducer: 15

!total  1.0
the     0.06016882606988614
and     0.03082057322340008
to      0.026534485015050385
a       0.022739170265672033
of      0.020874231121580943
it      0.019958120664834447
she     0.018093181520743358
i       0.017864153906556733
you     0.015901060070671377
```

part c generate partition mapper on the fly that leads to a total order sort

In [24]:
```
list(map(chr, range(ord('A'), ord('Z')+1)))
```

Out[24]:
```
['A',
 'B',
 'C',
 'D',
 'E',
 'F',
 'G',
 'H',
 'I',
 'J',
 'K',
 'L',
 'M',
 'N',
 'O',
 'P',
 'Q',
 'R',
 'S',
 'T',
 'U',
```

```
'V',
'W',
'X',
'Y',
'Z']
```

In [52]:

```
# <--- SOLUTION --->
# part c - hadoop job (RUN THIS CELL AS IS)
!hdfs dfs -rm -r {HDFS_DIR}/tos-output
!hadoop jar {JAR_FILE} \
  -D stream.num.map.output.key.fields=3 \
  -D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
  -D mapreduce.partition.keypartitioner.options="-k1,1" \
  -D mapreduce.partition.keycomparator.options="-k3,3nr" \
  -files TotalOrderSort/TOS_mapper.py,TotalOrderSort/reducer.py,TotalOrderSort/partitions.txt \
  -mapper TOS_mapper.py \
  -reducer reducer.py \
  -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner \
  -input {HDFS_DIR}/frequencies-output \
  -output {HDFS_DIR}/tos-output \
  -cmdenv PATH={PATH} \
  -numReduceTasks 6
```

```
Deleted /user/root/demo3/tos-output
packageJobJar: [] [/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.15.0.jar] /tmp/streamjob3874368387786140947.jar
tmpDir=null
18/09/19 07:50:12 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/09/19 07:50:13 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/09/19 07:50:14 INFO mapred.FileInputFormat: Total input paths to process : 1
18/09/19 07:50:14 INFO mapreduce.JobSubmitter: number of splits:2
18/09/19 07:50:14 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1537278954088_0022
18/09/19 07:50:14 INFO impl.YarnClientImpl: Submitted application application_1537278954088_0022
18/09/19 07:50:14 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_153727
8954088_0022/
18/09/19 07:50:14 INFO mapreduce.Job: Running job: job_1537278954088_0022
18/09/19 07:50:23 INFO mapreduce.Job: Job job_1537278954088_0022 running in uber mode : false
18/09/19 07:50:23 INFO mapreduce.Job:  map 0% reduce 0%
18/09/19 07:50:32 INFO mapreduce.Job:  map 50% reduce 0%
18/09/19 07:50:33 INFO mapreduce.Job:  map 100% reduce 0%
18/09/19 07:50:44 INFO mapreduce.Job:  map 100% reduce 17%
18/09/19 07:50:48 INFO mapreduce.Job:  map 100% reduce 33%
18/09/19 07:50:51 INFO mapreduce.Job:  map 100% reduce 50%
18/09/19 07:50:52 INFO mapreduce.Job:  map 100% reduce 83%
18/09/19 07:50:53 INFO mapreduce.Job:  map 100% reduce 100%
18/09/19 07:50:53 INFO mapreduce.Job: Job job_1537278954088_0022 completed successfully
```

```
18/09/19 07:50:53 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=103005
                FILE: Number of bytes written=1412116
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=92273
                HDFS: Number of bytes written=87919
                HDFS: Number of read operations=24
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=12
        Job Counters
                Killed reduce tasks=1
                Launched map tasks=2
                Launched reduce tasks=6
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=11627
                Total time spent by all reduces in occupied slots (ms)=74728
                Total time spent by all map tasks (ms)=11627
                Total time spent by all reduce tasks (ms)=74728
                Total vcore-milliseconds taken by all map tasks=11627
                Total vcore-milliseconds taken by all reduce tasks=74728
                Total megabyte-milliseconds taken by all map tasks=11906048
                Total megabyte-milliseconds taken by all reduce tasks=76521472
        Map-Reduce Framework
                Map input records=3010
                Map output records=3010
                Map output bytes=96949
                Map output materialized bytes=103041
                Input split bytes=258
                Combine input records=0
                Combine output records=0
                Reduce input groups=3010
                Reduce shuffle bytes=103041
                Reduce input records=3010
                Reduce output records=3010
                Spilled Records=6020
                Shuffled Maps =12
                Failed Shuffles=0
                Merged Map outputs=12
                GC time elapsed (ms)=539
                CPU time spent (ms)=13280
                Physical memory (bytes) snapshot=1678360576
                Virtual memory (bytes) snapshot=10929672192
                Total committed heap usage (bytes)=1217396736
        Shuffle Errors
                BAD_ID=0
```

```
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=92015
        File Output Format Counters
                Bytes Written=87919
18/09/19 07:50:53 INFO streaming.StreamJob: Output directory: /user/root/demo3/tos-output
```

In [53]:

```
# part c - print top words in each class (RUN THIS CELL AS IS)
for idx in range(6):
    print(f"\n============== PART-0000{idx}===============")
    numLines = !hdfs dfs -cat {HDFS_DIR}/tos-output/part-0000{idx} | wc -l
    print(f"Number of lines processed by this reducer: {numLines[0]}\n")
    !hdfs dfs -cat {HDFS_DIR}/tos-output/part-0000{idx} | head | column -t
```

```
============== PART-00000===============
Number of lines processed by this reducer: 15

!total  1.0
the     0.059757420372744306
and     0.03089767610031884
to      0.026591723367189297
a       0.0226802090523617
of      0.020740886829043816
it      0.020050619597015415
she     0.018177037110081187
i       0.01791407816454656
you     0.015810406600269535

============== PART-00001===============
Number of lines processed by this reducer: 16

as    0.009006343884561023
her   0.00815172731157348
with  0.00749432994773691
at    0.007461460079545081
s     0.007198501134010452
t     0.0071656312658186245
on    0.006705453111133025
all   0.0065739736383657104
this  0.005949446142720968
for   0.005883706406337311
```

```
============== PART-00002===============
Number of lines processed by this reducer: 18

so      0.004996219965157939
very    0.0047661308878151395
what    0.004667521283239654
is      0.004437432205896854
he      0.004207343128554054
little  0.004207343128554054
out     0.003878644446635769
if      0.003812904710252112
one     0.0034842060283338263
up      0.0033855964237583407

============== PART-00003===============
Number of lines processed by this reducer: 33

them     0.0028925484008809127
know     0.0028925484008809127
project  0.002859678532689084
were     0.0027939387963054267
like     0.0027939387963054267
have     0.0027939387963054267
herself  0.0027281990599217695
again    0.0027281990599217695
went     0.0027281990599217695
would    0.0027281990599217695

============== PART-00004===============
Number of lines processed by this reducer: 75

now     0.0019721920915097132
turtle  0.0019393222233178844
way     0.001906452355126056
my      0.001906452355126056
began   0.001906452355126056
tm      0.0018735824869342275
ll      0.0018735824869342275
which   0.0018407126187423989
mock    0.0018407126187423989
hatter  0.0018407126187423989

============== PART-00005===============
Number of lines processed by this reducer: 2853

succeeded  9.860960457548565e-05
moving     9.860960457548565e-05
venture    9.860960457548565e-05
```

```
mark       9.860960457548565e-05
picked     9.860960457548565e-05
pie        9.860960457548565e-05
pieces     9.860960457548565e-05
stopped    9.860960457548565e-05
yesterday  9.860960457548565e-05
pity       9.860960457548565e-05
cat: Unable to write to output stream.
```