Recitation 3

Task1

```
> bin/zookeeper-server-start.sh config/zookeeper.properties

> bin/kafka-server-start.sh config/server.properties

> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-
factor 1 --partitions 1 --topic test

> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
test --from-beginning

Then you can send the msg from the producer to the consumer.

> $SPARK_HOME/bin/spark-submit
$SPARK_HOME/examples/src/main/python/streaming/network_wordcount.py
localhost 9999

> nc -lk 9999
hello world, this is weijia sun from ece 579 cloud compuing
```

Task2

Q1: What does the below part of the makefile do?

The below part of code lists the required files for MapReduce mapper, combiner and reducer and set `mmm-map.py, mmm-combiner.py` , `mmm-reduce.py` as the input file for mapper, combiner and reduce, and then set the input directory and output directory as defined at the begining. At last the commands print the content of mmm file to show the result.

Q2: What is the output mmm?

The output mmm file stores the min, max and mean key-value pairs as the result of MapReduce.

Q3: What does mmm-map do?

`mmm-map`turn every input content into three key-value pairs as min x.xxx, max x.xxx and mean x.xxx.

Q4: What does the for-loop in mmm-combiner do?

mmm-combiner updates the xmax and xmin from key max and min, and add the sum of every content as well as count the number of total content for futher calculation of mean, output processed key-value pairs as min, max and mean.

mmm-reducer continue finish the update task which may not be finished by combiner, and then calculate the mean according to xsum and count.

Q5: Can the mmm-combiner code be replaced with the code in mmm-reducer? Why?

mmm-reducer continue finish the update task which may not be finished by combiner, and then calculate the mean according to xsum and count.

I don't think combiner's code can be replaced by reducer's because combiner is not guaranteed to launch as job is busy. If replaced and one of the combiner did't work, the calculated value of mean would be wrong. A combiner should function as a optimizer to save the network transmission and shall not impact the final output.

hist-map.py

```
#!/usr/bin/env python
import sys
import numpy

### Prevents pipe IO errors for large files.
from signal import signal, SIGPIPE, SIG_DFL
signal(SIGPIPE,SIG_DFL)
###

# read parameters from distributed cache - nbins, minmeanmax
f = open('nbins','r') # read nbins file
params = f.readline().strip().split() # read nbins file
nbins = int(params[0]) # set maximum number of bins
f.close()

f = open('mmm','r') # open mmm file

'''
Read mmm file and assign min, max and mean values.
File read operation in similar to above reading procedure.
'''
params =  f.readline().strip().split()
xmax = float(params[1])
params = f.readline().strip().split()
xmean = float(params[1])
params = f.readline().strip().split()
xmin = float(params[1])
f.close()
```

```
dx = (xmax-xmin)/nbins # bin width calculation


# compute bin centre
#### complete this code
for line in sys.stdin:
    line = line.strip()

    words = line.split()
    x = float(words[0])
    if xmax-x<=0 :
        bn = int((x-xmin)/dx)-1
    else:
        bn = int((x-xmin)/dx)
    bc = bn * dx + xmin +dx/2
    strbc = str(bc)


# process input data
#### complete this code
    print '%s\t%d' %(strbc,1);
```

hist-combine-reduce.py

```
#!/usr/bin/env python
import sys
from signal import signal, SIGPIPE, SIG_DFL
signal(SIGPIPE,SIG_DFL)

count = 0
currentKey = None

#### Complete the rest of the code

def updateResults(value, count):
    scount = value
    newcount = int(scount)
    count += newcount
    return (count)

def printResults(key, count):
    if key:
        print '%s\t%d' % (key, count);


for line in sys.stdin:
    line = line.strip()

    key, value = line.split('\t',1)
    if currentKey == key:
        count = updateResults(value, count)
       # count += int(value);
    else:
        printResults(currentKey, count)
        currentKey = key;
       # count = int(value);
        count = updateResults(value, 0)

printResults(currentKey, count)
```

Task 3

Code

```
import math
import numpy as np
from pyspark import SparkContext

def Simulation(sc, ncells, nsteps, nprocs, leftX=-10.,
rightX=+10.,sigma=3., ao=1., coeff=. 375):
    def I(x):
        return ao*math.exp(-(x**2/(2*sigma**2)))

    x = np.linspace(-9.5, 10.5, ncells) # Create an array with ncells
items from -9.5 to 10.5


    # Create an array with nsteps+1 items from 0th run # to nstpes th run.
    # For our assignment: We should get totally 21 time steps from 0th to
20th
    t = np.linspace(0, nsteps, nsteps+1)

    u = np.zeros(ncells) u_1 = np.zeros(ncells)
    for i in range(0, ncells):
        # Calculate the initial time value
        u_1[i] = I(x[i])

    # Update the value of n th run
    for n in range(1, nsteps+1):
        # Update the value of all cells except the bound points
        for i in range(1, ncells-1):
            u[i] = u_1[i] + coeff*(u_1[i-1] - 2*u_1[i] + u_1[i+1])
        u_1[:]= u
        u[0] = I(x[0]) # Keep the bound points unchanged #Print the result
        u[ncells-1] = I(x[ncells-1])
    for i in range(0,ncells):
        print 'x',i,': ',u[i]
```

```
if __name__ == "__main__":
    sc = SparkContext(appName="SparkDiffusion")
    Simulation(sc, 100, 20, 4)
```

output

plot

The change pattern is like a linear growing because the i point is always decided by the value of last time. It's seems like a O(t) time grows. The change is slow because the third point is too closed to the boundary point.

Task 4

code

```
import re
import datetime
from operator import add
from pyspark import SparkContext


# Loads in input file. It should be in format of:
#     URL         neighbor URL
#     URL         neighbor URL
#     URL         neighbor URL
#     ...
def computeContribs(urls, rank):
    num_urls = len(urls)
    for url in urls:
        yield (url, rank / num_urls)


def parseNeighbors(urls):
    parts = re.split(r'\s+', urls)
    return parts[0], parts[1]


sc = SparkContext(appName="PageRank")
startTime = datetime.datetime.now()
lines = sc.textFile('pagerank.txt')
links = lines.map(lambda urls:
parseNeighbors(urls)).distinct().groupByKey().cache()

ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))
for iteration in range(50):
    # Calculates URL contributions to the rank of other URLs.
    contribs = links.join(ranks).flatMap(
        lambda url_urls_rank: computeContribs(url_urls_rank[1][0],
url_urls_rank[1][1]))
    # Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(add).mapValues(lambda rank: rank * 0.85 +
0.15)
endTime = datetime.datetime.now()
for (link, rank) in ranks.collect():
    print("%s has rank: %s." % (link, rank))
interval = (endTime - startTime)
print("Total time: %s " % interval)
sc.stop()
```

Q1

Didn't find the example for this question

Q2

After increasing the number of iterations from 5 to 50, we cannot observe any convergence or other phenomenon. The time it takes doesn’t increase with the number of iteration.

Q3