**Recitation 3: Spark (and Hadoop) Rutgers Fall 2018, Instructor: Maria Striki**

**Date Issued**: 11-16-18

**File Sources:** Please download from sakai/Recitation/Rec3 to your pyspark local directory the text and csv related files

**ATTENTION**: UPDATE TO MAKE FILES: mkdir –p (**MAY NOT** have been fixed for all Makefiles)

**TASK1: Introduction to Spark Streaming (INSTALLATION RECITATION PROBLEM)**

One of the major advantages of using Spark Streaming is that we can receive multiple data streams from input sources, process them on a cluster, push out dashboards/databases. A **Discretized Stream (DStream)**, the basic abstraction in Spark Streaming, is a continuous sequence of RDDs (of the same type) representing a continuous stream of data. DStreams can either be created from live data (such as, data from HDFS, Kafka or Flume) or they can be generated by transforming existing DStreams using operations such as map and window.

Apache Kafka is a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system for real time data stream processing. Common use cases include: Stream Processing, Website Activity Tracking, Metrics Collection and Monitoring, Log Aggregation.

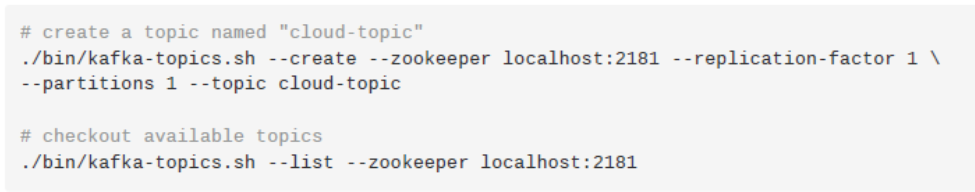
**Official Kafka installation and guidelines web-sites:**

<https://www.confluent.io/blog/stream-data-platform-1/>

**Download Apache Kafka from the following URL:**

<https://kafka.apache.org/downloads>

We will now setup a “topic” for the Kafka stream and run a server:



Apache Kafka comes with two shell scripts: kafka-console-producer.sh to send messages and kafka-console-consumer.sh to receive messages. They both use the console (stdin) as input and output.



Now publish messages by typing in the producer window. The processed stream will be returned by consumer in its console.

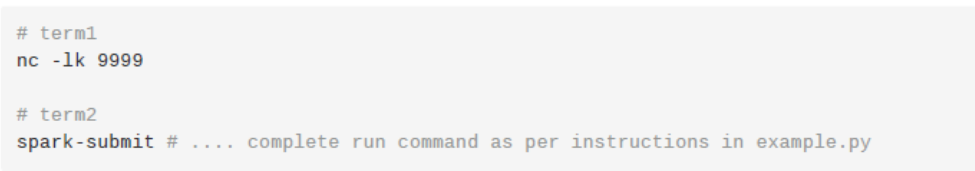
We will now execute a task which processes the input stream from a port.



Now enter sentences in term2. This input Dstream will be processed and the resultant word count output can be seen on term1.

**Spark Streaming Pipeline:**

We will now setup a basic spark streaming pipeline. Complete the program example.py as per instructions. Our task is to process the input Dstream and check whether it is in positive, negative or zero (states). We will also maintain a counter for each type of state and update it realtime. Spark streaming is a powerful tool used to analyze Big data efficiently in real time. After completing the program open two console windows.



Now provide the input using term1 for your Spark streaming job. The output will be displayed in term2.

**Your Solution:**

**TASK2: Histogram of Data in Python**

In this exercise we will use Hadoop to build a histogram of input data. Histogram is a graphical representation of frequencies of samples. The provided example reads an input and finds the min, max, and mean of the input first. It then uses the below equations to tag each input with a bin in the histogram.

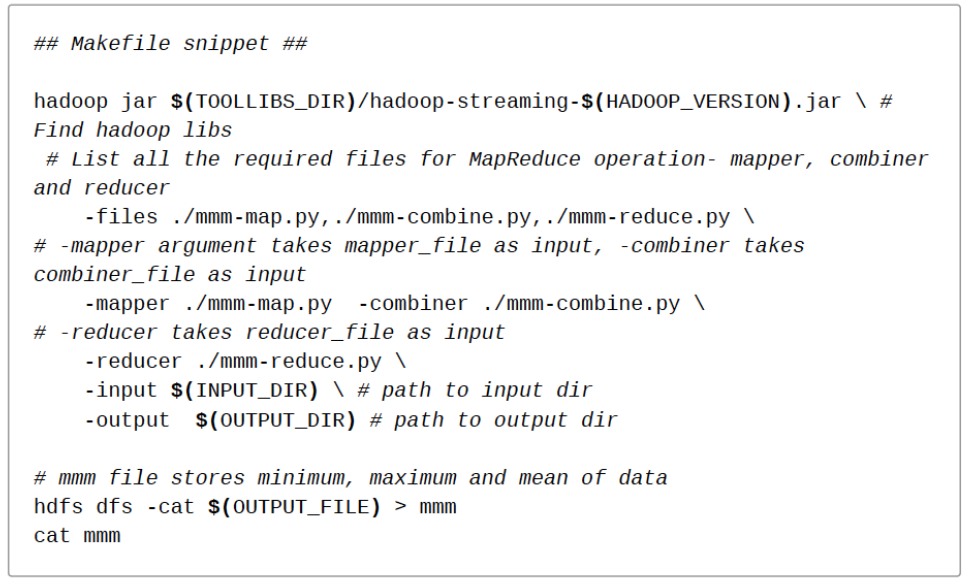
𝑏𝑖𝑛𝑊𝑖𝑑𝑡ℎ = (𝑥MAX – 𝑥MIN)/𝑛𝑢𝑚𝑏𝑒𝑟𝑜𝑓𝑏𝑖𝑛𝑠

𝑏𝑖𝑛𝑁𝑢𝑚𝑏𝑒𝑟 = (𝑥i − 𝑥MIN)/𝑏𝑖𝑛𝑊𝑖𝑑𝑡ℎ

𝑏𝑖𝑛𝐶𝑒𝑛𝑡𝑒𝑟 = 𝑏𝑖𝑛𝑁𝑢𝑚𝑏𝑒𝑟 ∗ 𝑏𝑖𝑛𝑊𝑖𝑑𝑡ℎ + 𝑥MIN + 𝑏𝑖𝑛𝑊𝑖𝑑𝑡ℎ/2

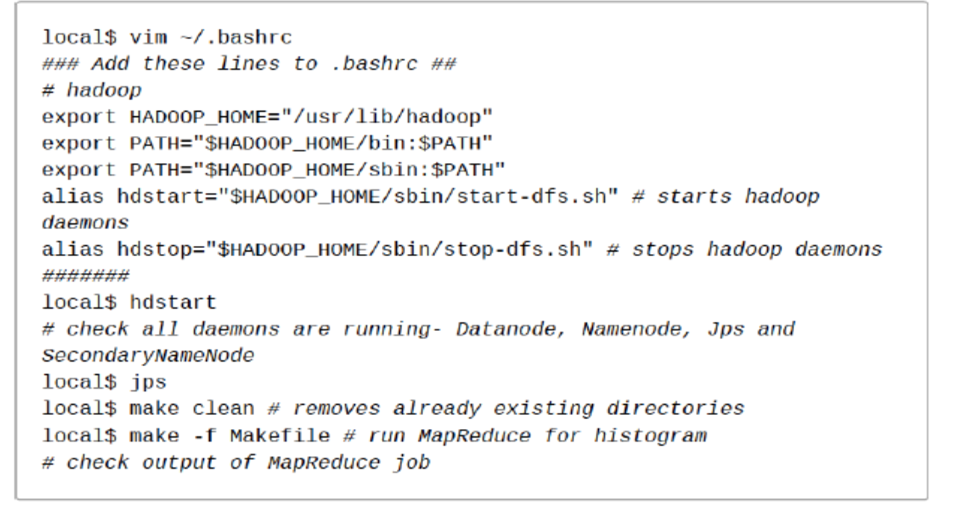
Use the provided folder: prob\_code\_hist to download the code required to work on this problem. There are a couple of places that you need to write your own Hadoop code to make your Histogram work: hist-combine-reduce and hist-map.

**Question 1:** Now read the provided code. What does the below part of the makefile do? What is the output mmm? What does mmm-map do? What does the for-loop in mmm-combiner do? Can the mmm-combiner code be replaced with the code in mmm-reducer? Why?



**Question2:** Now that we have found the mean, min, and max of data we want to build a histogram of the original inputs. Complete the hist-map and hist-combine-reduce files to show the number of total data points in each bincenter.

Hints for running the code: in your console follow the below:



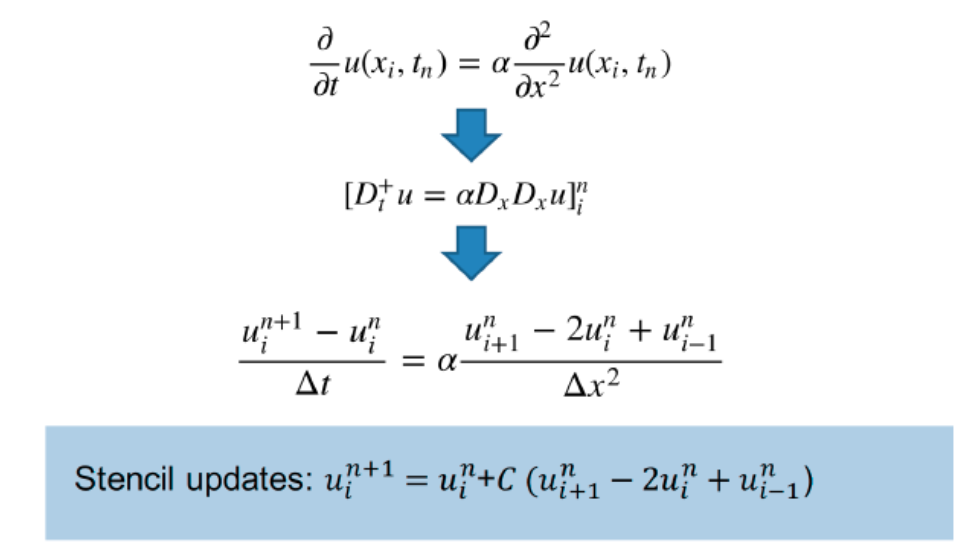
**Your Solution:**

**Task 3:**

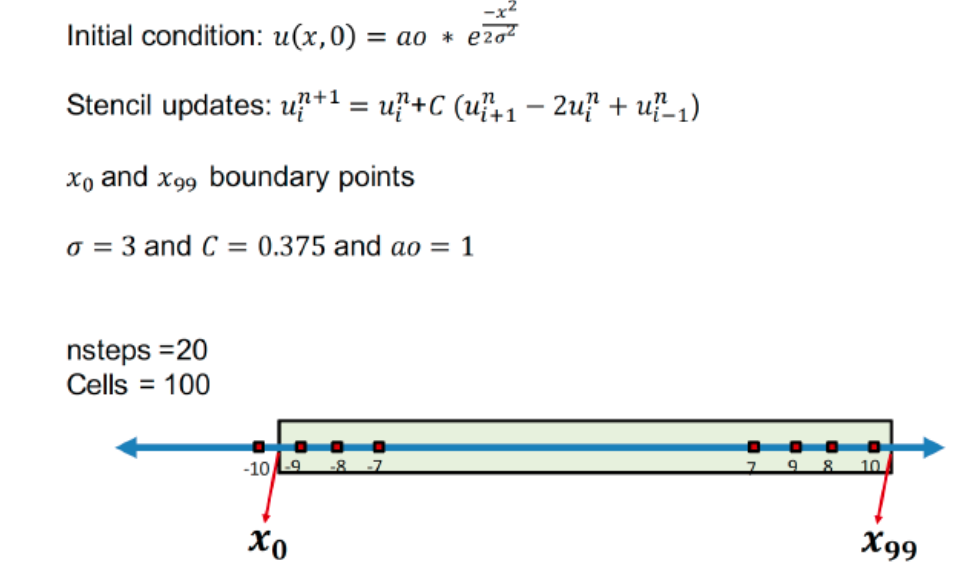
In this assignment you will write a code that computes a 1D diffusion problem using a finite difference (structured grid) approximation. Read the first three sections of the following link:

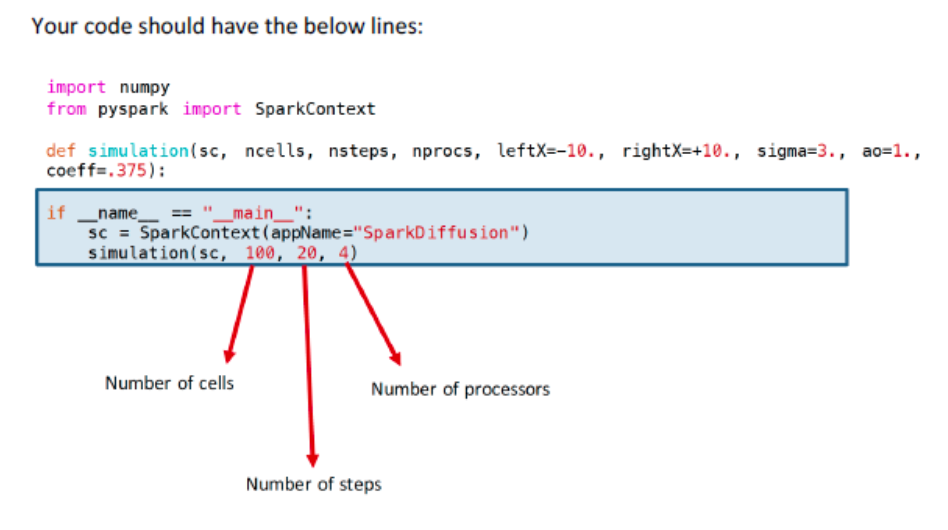
*http://hplgit.github.io/num-methods-for-PDEs/doc/pub/diffu/sphinx/.\_main\_diffu001.html*

A brief overview of the document is given in the below where a differential equation is approximated using a 1-point finite-difference problem:

****

Now solve the below diffusion problem using the finite-difference (stencil) approximation illustrated in the previous page. The boundary points x0 and x99 are located in points -9.5 and +10.5 in the x-axis, n represents the number of time-steps the algorithm runs to convergence, number of cells refers to the number of data points between boundaries, and *i* shows the point being operated on.





Items to do:

1. Print your code.

2. Show the output in which the final values are printed for each point in the x-axis (you have 100 data points!).

3. Plot the value of the third data point (*i=3*) for number of steps = 2, 4, 10, 20, 100, 200. Is there a pattern to the changes with increasing number of steps? Why?

**Your Solution:**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**2-3 MORE TASKS WILL FOLLOW – CONTINUE AS HOMEWORK 3**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***