Jackson State University
Department of Computer Science
CSC 435 Computer Networks, Spring 2019
Instructor: Dr. Natarajan Meghanathan
Project # 4: Large Scale Data Processing: Map Reduce Paradigm and Multicast Sockets

Due: April 1, 2019 (4 PM) **Max. Points:** 100

Map Reduce is a parallel processing paradigm, wherein a large dataset is processed in parallel, by several processes. The dataset is assumed to be indexed so that each process can extract its portions from the dataset (map phase) and process them. After processing the data, the process sends the computation results as <key, value> pair(s) to other processes that aggregate the results (reduce phase) and finalize the results of the computation. The key(s) sent by a process are unique to that process based on the index values of the dataset to which the process is mapped to.

Synopsis: In this project, you are going to find the sum of an array of integers generated and stored (as an ArrayList object) in a file using the Map Reduce paradigm and communication implemented using multicast sockets. You will write a multicast sender receiver program that will first read the ArrayList object from the file and store it as a local object. There are *n* processes in the multicast group and each process is assigned a unique ID (0, 1, 2, ..., *n* -1). A multicast process only reads the ArrayList data whose indices are (ID + *x***n*), where *x* is an integer starting from 0 and *n* is the number of processes. This would continue as long as the index value is within the ArrayList boundary. For example, if the ArrayList data has 10^5 integers and if there are $n = 5$ processes in the multicast group, then process with ID 2 reads the data at indices (2 + 0*5), (2 + 1*5), (2 + 2*5), ... 99997. A process sums up the integers it has read from the data set and ***sends*** a multicast message <*key*, *value*> pair to all processes (including itself), where the key is the process ID and the value is the partial sum of the integers read. The receiver thread of a process runs in a loop, waiting to receive the <*key*, *value*> pairs from all the processes including itself. The receiver thread stores the partial sum values received for each key (i.e., process ID) in a sum array. The indices of the sum array are the key values (process ID values) that range from 0 to *n*-1, where *n* is the number of processes. After receiving all the <key, value> pairs, the read thread computes the sum of the partial sum values in the sum array and prints the overall sum. The read thread should also find the sum array index (i.e., the key or the process ID) that sent the maximum of the partial sum values and print the index value along with its corresponding partial sum value.

In this project, you are going to find the sum of an array of 10^5 integers generated and stored as an ArrayList object in a file. You will write a multicast sender receiver program that will first read the ArrayList object containing the 10^5 integers from the file, and store it as a local list object. You will run 5 processes as part of the multicast group. Each process is assigned a unique ID (0 through 4). A process with a particular ID reads the data from the ArrayList mapped to its ID (as explained above), sums them up and multicasts the sum as a <key, value> pair: <ID, partial sum value> to all the 5 processes, including itself. The read thread at each multicast process stores the partial sum values received in a sum array, indexed with the key/process ID. After receiving the <key, value> pairs from all the 5 processes, the read thread of each process should be able to locally compute the overall sum value of all the elements in the ArrayList data set (by summing the partial sum values in the sum array) as well as print the process ID whose partial sum value is the maximum.

First Step: ***Data Generator Program*****:** I have given you the Java program to generate the data set of 10^5 integers and store them in an ArrayList object that is saved to a file. See the Project section in the course

website for the program. As a first step, you should run the program and save the data in a file named '**data**'.

Guidelines to the design of your multicast sender/receiver program: You write one program that has a class with the main function to read the input values, read the integers from the data (ArrayList object) stored in a file and store them as a local ArrayList object, determine the partial sum value (as mapped to the process ID, explained above) and send the partial sum value to all the processes. Make arrangements to launch the read thread in the main function itself as shown in the sample programs of the programming manual. The program should also have a class for the read thread and the run() method that basically implements the receiving of the partial sum values as <key, value> pairs and storing them in a sum array; after receiving the <key, value> pairs from all the processes, the read thread should compute the overall sum value and the process ID whose partial sum value is the maximum. Set the multicast port to be the last 4 digits of your J# and if it is less than 1000, add a 10000 to it.

trim( ) method and StringTokenizer: You could send the Process ID and the partial sum value as a string (concatenate the Process ID and the partial sum value, separated by a blank space). In the read thread, first use the trim( ) method on the string extracted from the byte array, before passing the string to the StringTokenizer to extract the process ID and the partial sum value. Refer to the example (Java code and video explanation: http://www.youtube.com/watch?v=jMbbvMeZuDc) posted along with this project description to know more about the use of StringTokenizer.

Inputs to your multicast program:
* Number of processes
* Process ID (ID of the particular process)
    make sure the process ID is unique and lies between 0 to (number of process - 1)
* Number of integers in the data set (array size)
* File name in which the data set is stored

Outputs:
Your multicast process should output the following:
* The local partial sum value computed by reading the data elements with indices mapped to its process ID
* The partial sum values received from every process (including itself) displayed along with the process ID
* The overall sum value computed based on all the partial sum values received
* The maximum of the partial sum values and the ID of the corresponding process that sent the maximum partial sum value.

Sample programs on ArrayList
You are given two programs on using ArrayList. The ArrayListWriter.java program creates an Array List object of size and maximum value input by the user. The ArrayList is filled up using randomly generated integers. The program then saves the ArrayList object to a file named **data.** The ArrayListReader.java program would first read the **data** file, extract the ArrayList object from the file and store the contents as a local ArrayList object. The program will then loop through the ArrayList, extract its individual values and sum them up. The overall sum is then printed.

What to Submit:
(1) **A video file** (either one of these formats: .mp4, .wmv, .avi) that is generated by desktop recording your explanation of the working of your program and the logic/approach you took to implement the main method of the multicastSenderReceiver class and the readThread class, along with the run method to satisfy the design requirements. You should display the program(s) on the desktop and walkover the different sections of your code as well as explain the execution flow of the program. You should also record demonstrating the working of your program**.**

Note that the contents of the desktop/programs captured through your video should be clearly readable.

Submit the video through Google Drive (using your JSU email address) and send the link via email to natarajan.meghanathan@jsums.edu

You could try using one of the **desktop recording software** (or anything of your choice):
CamStudio: http://sourceforge.net/projects/camstudio/files/legacy/
Debut: http://www.nchsoftware.com/capture/index.html

(2) A **report** featuring your code for the multicastSenderReceiver.java program and snapshots of the five multicast processes with the local partial sum value and the partial sum values received from all the processes, including itself, displayed as a <key, value> pair: where key is the process ID and value is the partial sum value received from the processes. The processes should then output the overall sum value and the maximum of the partial sum values and the process ID that sent the maximum partial sum value.

Submission:
(1) Video file shared (to: natarajan.meghanathan@jsums.edu) via Google Drive
(2) Hardcopy of the report, submit in class