Lab 01 Specification – A Hands-on Exercise to get started in Cloud &
practice implementing scalable programs.
50 points

Laboratory Session Goals

- Quick review of GitHub and git commands.
- Refresh your ability to write Java code.
- Think about how to divide problems into multiple segments.
- Learn worker based program implementation to achieve scalability at the computational level.
- Practice MultiThread programming.

Learning Assignment

If not done previously, it is strongly recommended to read all of the relevant "GitHub Guides", available at the following website:
`https://guides.github.com/`
that explains how to use many of the features that GitHub provides. This reading assignment is useful to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, it is also recommended to do the reading assignment outlined below:

- **MultiThreading reading material in the class repository.**

Assignment Details

Now that we have discussed some basic principles & did a practical session to practice some basic large scale computational tasks, it is now time to experience more in-depth. In this lab, students will practice developing advanced scalable computational program(s) to retain the knowledge from the class discussions so far. This also includes developing one or more code files to implement the worker based formalizations using a programming language such as Java.

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other's program(s) is strictly not allowed. Please note that all work done during laboratory sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as lab assignments, skill tests, projects, etc.

At any duration during and/or after the lab session, students are recommended to team up with the Professor to clarify if there is any confusion related to the items in the lab sheet and/or class materials.

Section 1: Scalable Prime Number Program Implementation

This section is worth 25 points. The points breakdown is provided below:

- Task 2 = 5 points
- Task 3 = 5 points
- Task 4 = 5 points
- Task 5 = 10 points

We implemented a prime number program (to compute the total number of prime numbers between 0 and a given number) using multiple workers, during our class discussions and practicals. Additionally, we discussed how to implement a program using multiple threads during our most recent class discussions. To add further scalability & solidity to the program and to practice scalable implementation, it is required to complete the tasks listed below:

1. **Task 1:** Read through the scalable prime number computation program provided in the starter-code repository with a file named **Prime.java**. This code is a variation of the initial version of this program that we did in class and practicals. Here we assume the worker id's start with 0. There are three classes in the program. The first class named Report is used to synchronize the result produced by each worker thread. To do such a result synchronization, the doSync method is invoked by all the workers. The second class named RPrime is used to implement the Runnable class and set up all the required methods (such as compute and run) to execute each worker thread and produce the corresponding result, which is stored in the variable myshare. Here the word myshare represents the share of the computational result by each worker thread. The third class named Prime is used to orchestrate the complete program by communicating with the other two classes and achieve the goal of the program, to find the number of primes between 0 and a given number. The Prime class has the method doWork, which internally uses the ExecutorService thread pool feature to run the prime compute program using multiple threads concurrently.

2. **Task 2:** In line no 53, the variable myshare is set to 0. This results in the program not fully functional. Add the required logic to this line of code, to integrate the right snippet of code, to get the myshare code populated correctly. To do this part, it is required to read through the code thoroughly and get a complete understanding of the program. And think through, how to do the right method call(s) to get the right result populated. It may also need some debugging to verify the correctness of the integration of the new code.

3. **Task 3:** At this point, after completing the previous steps, all your local results (from the workers) are expected to be correctly populated. However, the final result may not show up. To do this part, it is required to add the required logic in the doSync method in the Report class. Add the necessary line(s) of code inside the try block, after the comment, in line no 7. It may be worth refreshing your memory on the try-catch block from class discussions in CMPSC 201. Reading the following online material may help you to do a quick refresh of your memory on this concept learned in the 201 course.
   `https://www.geeksforgeeks.org/flow-control-in-try-catch-finally-in-java/`

4. **Task 4:** At this point, after completing the previous steps, the program is expected to be fully functional when alpha % beta is equal to 0, that is when the proportions are equal. However, if alpha % beta is not equal to 0, that is when the proportions are unequal the ranges may not be accurate. This inaccuracy may lead to both myshare and total result to produce incorrect results. This can be verified by setting alpha is equal to 98 and beta is equal to 5. The output results may not be accurate while executing. To avoid this, we are using a variable named excess. This variable is used to calculate the excess data resulted due to unequal proportions. The excess data is then distributed to the workers. Between lines 47 and 50, add the required logic to update the start and end variables. By doing this update, the program is expected to produce the correct range for each worker, if the proportions are unequal. It may also need some debugging to verify the correctness of the integration of the new code.

5. **Task 5:** At this point, after completing the previous steps, the program should be solid and fully functional. If you find any issues, please report your experience to the Professor. After doing a series of steps, to practice the design and development issues in implementing the prime number program, let us now shift our focus to the experimental side of this program. To showcase our experimental result, we are required to use the charting feature in LATEX. For simplicity, a starter code with a file named `charts.tex` is provided in the lab repository. The starter code has an example formal chart. In the Department of Computer Science, all thesis chapters are written and developed using LATEX code files. This lab presents an opportunity for students to learn this important skill. Please note that it is important to learn implementation using LATEX, to be proficient in developing professional documents in your career either in industry or academia. A latex file may be developed using the Overleaf website link provided below:
`https://www.overleaf.com/login`
Please note: The Overleaf website may prompt to register and log in before providing the options to compile latex files and generate pdf files. This should be a straight forward process. If there are any questions, students are encouraged to ask the Professor. After doing, the setup, do the following modifications to the code and record the running time for each run.

   - Modify lines 86 and 87, by setting the alpha value as 1000000 and the beta value as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Record the running time for each change in beta value and report your results using the charts.
   - Modify lines 86 and 87, by setting the alpha value as 2000000 and the beta value as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Record the running time for each change in beta value and report your results using the charts.
   - Modify lines 86 and 87, by setting the alpha value as 3000000 and the beta value as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Record the running time for each change in beta value and report your results using the charts.
   - Modify lines 86 and 87, by setting the alpha value as 4000000 and the beta value as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Record the running time for each change in beta value and report your results using the charts.
   - Modify lines 86 and 87, by setting the alpha value as 5000000 and the beta value as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Record the running time for each change in beta value and report your results using the charts.
   - Note, the program running time changes from machine to machine. Although the alpha values chosen are an optimum one across all machines, this may not be entirely true. So, please feel free to increase or decrease the alpha values, accordingly, after discussing it with the Professor.

   Finally, create 5 different charts using the template code provided in `charts.tex` file to report all your experimental results. Create a new file named `charts1.tex` to report the experimental results for this part. In lines 85, and [89-91], the program timing component (time taken to run the program) is implemented using the nano clock units. Although not required, it is optional, for students to create more charts for much larger alpha values and different combinations of beta values. Make sure to push both the `charts1.tex` and the `charts1.pdf` files to the repository. Additionally, create a reflection markdown file to report your understanding of the experimental results. Make sure to push the reflection file to the repository. You may use one reflection file for the entire lab.

Section 2: Scalable Twitter Analytics Implementation.

This section is worth 25 points. The points breakdown is provided below:

- Task 7 = 15 points
- Task 8 = 10 points

During this unprecedented time, public view on politics is changing every day, sometimes every few hours in a day. Social media has become a major platform for a common man to share their views publicly these days. One such social media platform is Twitter. Twitter provides a platform for people around the world to share their mind out using short messages called Tweets. These Tweets may often be categorized as Neutral, Positive, and Negative, based on the topic discussed. This categorization holds good for all controversial topics. These topics often lead to a debate on Twitter, and the categorization of the final verdict. In this section of the lab, we will use a synthetic data loader to load the scores for each tweet. A tweet with neutral categorization is represented by a score of 0. Tweet with positive categorization is represented by a score 1 and tweet with negative categorization is represented by a score 2. Although we don't use the real data here, the simulated data loaded in the array is well equipped to foster our learning goals for the lab, which is to learn scalable computations. To make things simple to understand, a starter code with the file name **Tweets.java** is provided in the repository. To practice some more scalable implementation techniques, it is required to complete the tasks listed below:

1. **Task 6:** Read through the scalable tweets analytics program provided in the starter-code repository with a file named **Tweets.java**. Although entirely different, this code has several code-repetitions from that of the **Prime.java**. The experience from section 1 should help understand this code better. Here we assume the worker id's start with 0 and always equal proportions, that is tweets_per_day % no_of_days is always 0. By doing this assumption, there is no need to take care of the excess for this program. The first class named RTweets is used to implement the Runnable class and set up all the required methods (such as run and constructor) to execute each worker thread and produce the corresponding result, which is stored in the array `tweets_populated`. This program does not require synchronization to be done. Each worker thread (that is, for each day) will take their portion of the array as input and update the final verdict of the day in the output array static array `tweets_processed`. The second class named `Tweets`, initially loads the data into the `tweets_populated` array with the scores for each tweet. As described above the scores are categorized as neutral, positive, and negative. After loading the data, the `startProcess` method is invoked. This method will then execute the run method in the `RTweets` class for each worker thread concurrently.
2. **Task 7:** At this point, the starter-code is not fully functional. It is required to complete the code in the run method in RTweets class, between lines 17 and 24. The sub-tasks required to be completed here is outlined below:

- Set up the appropriate code to read the array of tweets distributedly. That is, each worker thread should read their portion of data. To do this, it is required to think through and formalize how to break the array into segments so that each thread can read their portion. It is worth noting that, we have access to wid, tweets_per_day, and tweets_populated inside the RTweets class. A similar partitioning technique like the Prime.java program is required to be implemented for this part.
- The number of threads is equivalent to the number of days. Each thread is supposed to read their segment from the entire array and generate the final verdict score for the respective day.
- Compute the total number of neutral, positive, and negative tweets per thread. It is worth noting that, the number of times each thread will iterate is equivalent to the number of tweets per day (tweets_per_day).
- Find the maximum from the total number of neutral, positive, and negative tweets. This should indicate the final verdict for the respective days. If the maximum number is the total number of neutral tweets, then simply insert the value 0, to the (tweets_processed) array for the corresponding day. If the maximum number is the total number of positive tweets, then simply insert the value 1, to the (tweets_processed) array for the corresponding day. If the maximum number is the total number of negative tweets, then simply insert the value 2, to the (tweets_processed) array for the corresponding day.
- If the previous steps are correctly completed, the final verdict for each day is added to the tweets_processed array, using multiple threads. Again, each day represents a thread.

Task 8: At this point, after completing the previous steps, the program should be solid and fully functional. If you find any issues, please report your experience to the Professor. After doing a series of steps, to practice the design and development issues in implementing the tweets analytics program, let us now shift our focus to the experimental side of this program. To showcase our experimental result, we are required to use the charting feature in LATEX. For simplicity, a starter code with a file named `charts.tex` is provided in the lab repository. The starter code has an example formal chart. After doing, the setup, do the following modifications to the code and record the running time for each run.

- Modify lines 72 and 73, by setting the tweets_per_day value as 1000000 and the no_of_days value as 1, 2, 5, 10. Record the running time for each change in beta value and report your results using the charts.
- Modify lines 72 and 73, by setting the tweets_per_day value as 2000000 and the no_of_days value as 1, 2, 5, 10. Record the running time for each change in beta value and report your results using the charts.
- Modify lines 72 and 73, by setting the tweets_per_day value as 3000000 and the no_of_days value as 1, 2, 5, 10. Record the running time for each change in beta value and report your results using the charts.
- Modify lines 72 and 73, by setting the tweets_per_day value as 4000000 and the no_of_days value as 1, 2, 5, 10. Record the running time for each change in beta value and report your results using the charts.
- Modify lines 72 and 73, by setting the tweets_per_day value as 5000000 and the no_of_days value as 1, 2, 5, 10. Record the running time for each change in beta value and report your results using the charts.
- Note, the program running time changes from machine to machine. Although the values chosen are an optimum one across all machines, this may not be entirely true. So, please feel free to increase or decrease the tweets_per_day values, accordingly, after discussing it with the Professor.

Finally, create 5 different charts using the template code provided in `charts.tex` file to report all your experimental results. Create a new file named `charts2.tex` to report the experimental results for this part. In lines 71, and [90-92], the program timing component (time taken to run the program) is implemented using the nano clock units. Although not required, it is optional, for students to create more charts for much larger tweets_per_day values and different combinations of no_of_days values. Make sure to push both the `charts2.tex` and the `charts2.pdf` files to the repository. Additionally, create a reflection markdown file to report your understanding of the experimental results. Make sure to push the reflection file to the repository. You may use one reflection file for the entire lab.

Submission Details

For this assignment, please submit the following to your GitHub lab repository.

1. **reflection** markdown file
2. updated version of **Prime.java** file
3. updated version of **Tweets.java** file
4. **charts1.tex** and **charts1.pdf**
5. **charts2.tex** and **charts2.pdf**
6. It is highly important, for you to meet the honor code standards provided by the college and to ensure that the submission is made before the deadline. The honor code policy can be accessed through the course syllabus. Make sure to add the statement ”This work is mine unless otherwise cited.” in all your deliverables such as source code and PDF files. In your summary file, please make sure to include your name in all submissions.

Grading Rubric

1. Details including the points breakdown are provided in the individual sections above.
2. If a student needs any clarification on their lab credits, it is strongly recommended to talk to the Professor. The lab credits may be changed if deemed appropriate.