ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

# PA4: Threading and Synchronization

Maria Dmitrievskaia

UIN: 927009911

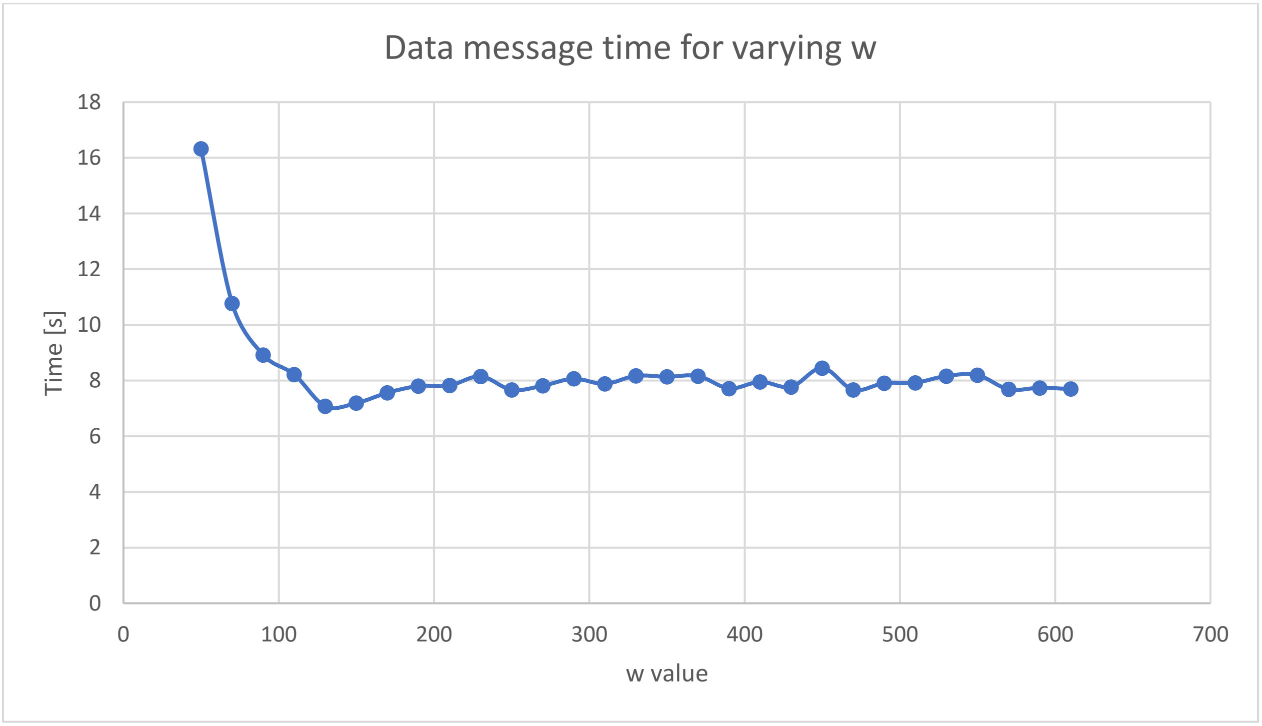
CSCE 313 – 512

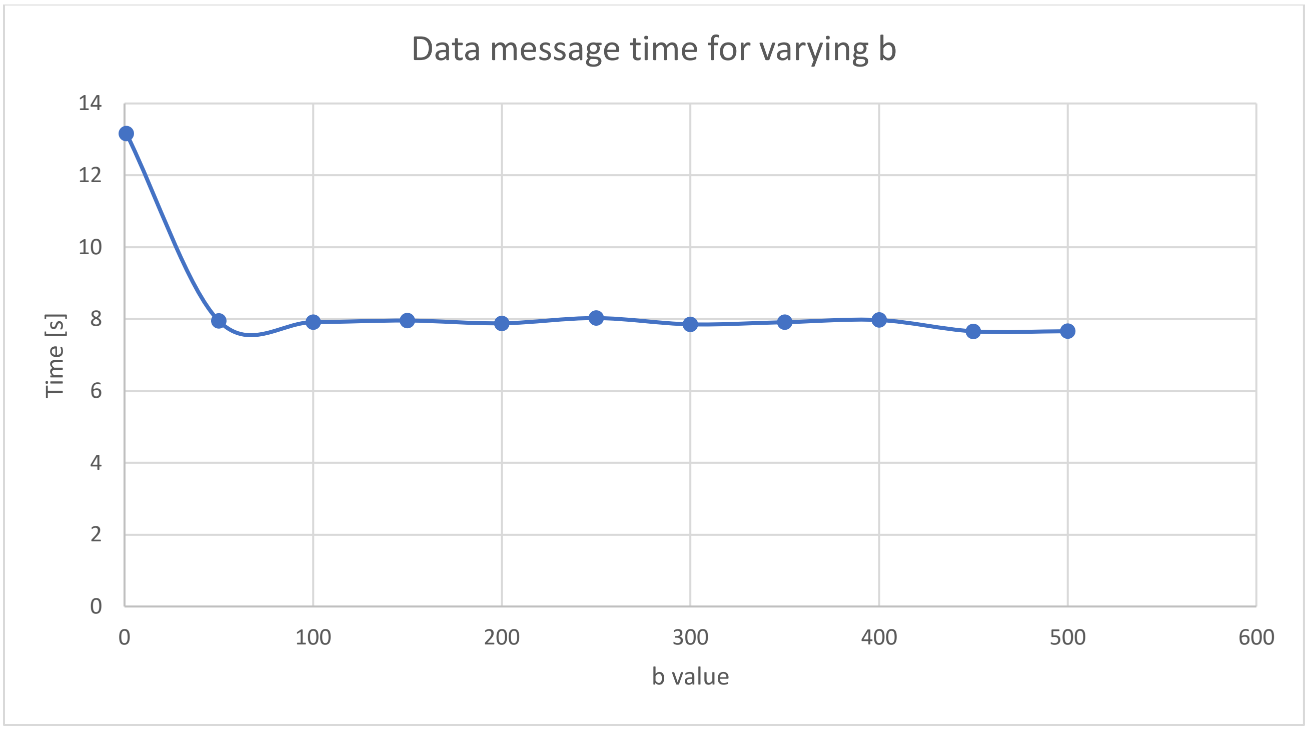
Due Date: October 19th , 2020

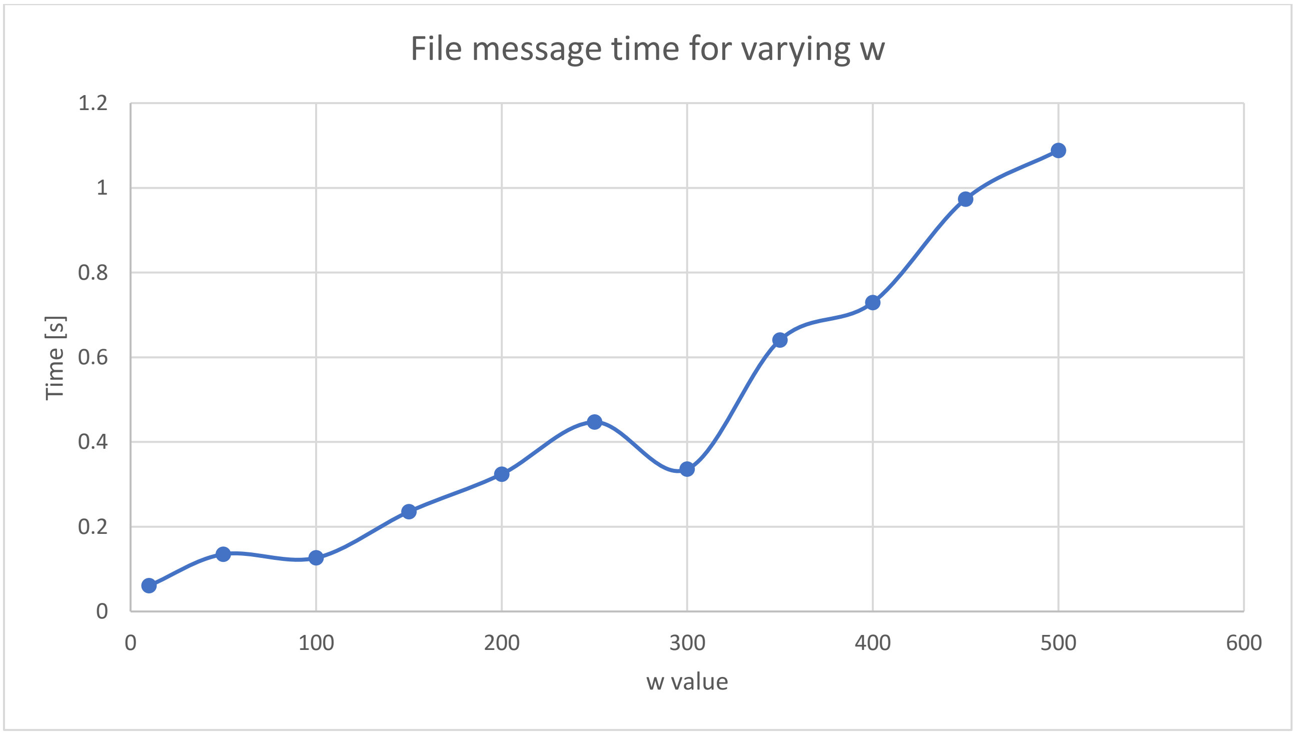
**Design:**

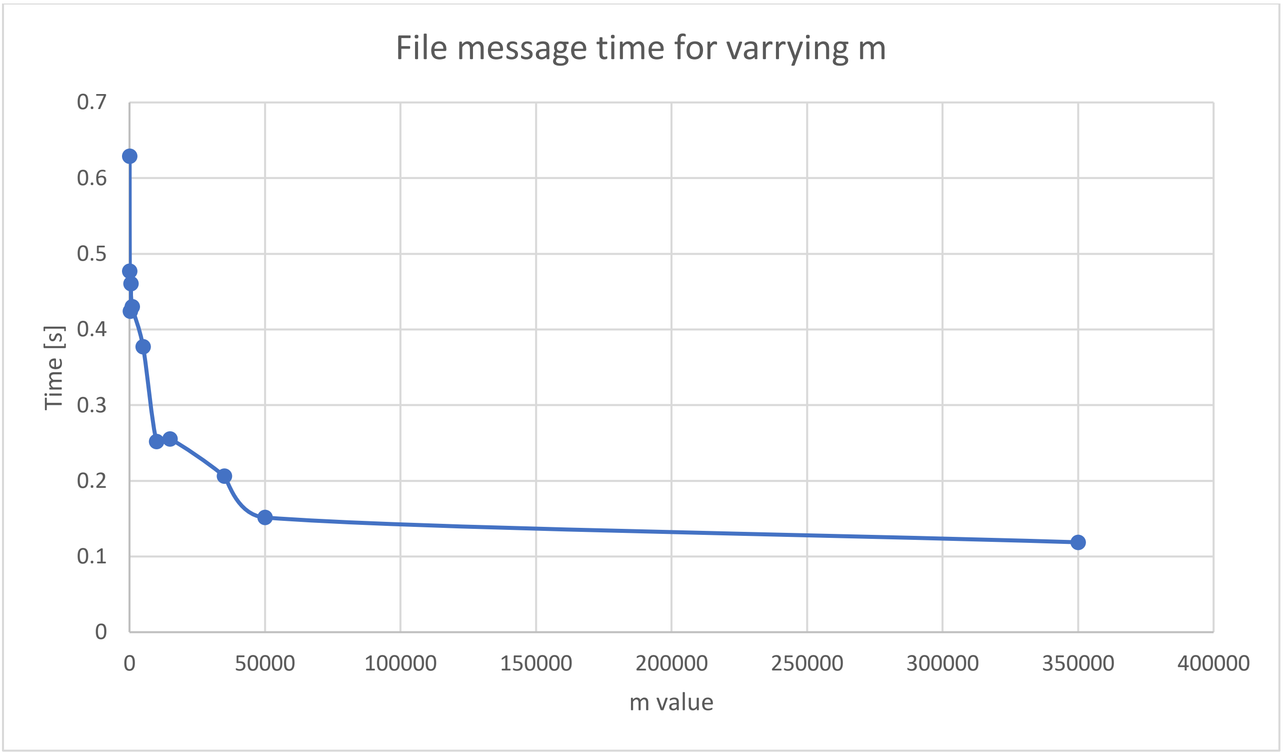
The provided code includes files from PA1, as well as an implemented Histogram and Histogram Collection class. For each patient, we create a Histogram object, thus resulting in *p* number of histograms. All of these histograms are added to the HistogramCollection which maintains the list and prints out the histogram values. The following modifications were made to the PA1 code in order to increase the efficiency through using threads. The client is able to accept different parameters ( n, p , w, b, m, f) and is able to chose to request data or to request a file. The patient and worker threads are started and later joined when they are finished. For data requests the client program prints the histogram collection and shows *n* for each person. The BoundedBuffer class was modified so that it is thread-safe and is protected against overflow and underflow conditions. Also, the user can change the buffer capacity with the -m argument, and the new buffer capacity is passed along to the server.

**Time Graphs:**




Data message time for varying b
Time [s]
b value
0
2
4
6
8
10
12
14
0
100
200
300
400
500
600


File message time for varying w
Time [s]
w value
0
0.2
0.4
0.6
0.8
1
1.2
0
100
200
300
400
500
600



1. Data Requests: Make two graphs for the performance of your client program with varying numbers of worker threads and varying size of request buffer (i.e. different values of w and b) for n = 15K. Discuss how performance changes (or fails to change) with each of them, and offer explanations for both. Do we see linear scaling on any of the parameters?

Due to the machine I am running my code on, the graphs I obtained are not necessarily accurate. The cut off point for varrying the worker threads for data requests that I obtained was around 130 (although when I ran it a few days before it was aroun 500). After this point we hit the point of diminishing return at which increasing the worker threads does not decrease the running time. Before hitting the point of diminishing return, the runtime decreases linearly with increasing worker threeads. This is because adding more threads sqeezes more CPU activity while the IPC is keeping the CPU idle for one or more threads. Changing the request buffer capacity did not significantly change the run time and it stayed around 8 seconds.

2. File Request: Make two graphs for the performance of your client program with varying numbers of worker threads and varying buffer capacity (i.e. different values of w and m). Discuss how performance changes (or fails to change) with each of them, and offer explanations for both. Do we see a linear scaling? Why or why not?

Due to the machine I am running my code on, the graphs I obtained are not necessarily accurate. For the file requests with varying worker threads the run time should stay relatively the same or decrease if the file is fully cached in memory. This should happen because there is not much CPU activity for a file transfer. The main activity involves reading from the disk and writing to the same disk, thus there is not a great advantage in increasing threads. My graphs shows an increase in runtime with increasing the worker threads which is due to the machine I am running my code on. With varying the buffer capacity we see a decrease in the file transfer runtime because a bigger part of the file is being grabbed each time.

**YouTube Video Link:**

https://youtu.be/i3WKmtTywrs