Anthony Rezzonico
CS 455

HW2-WC

1) What was the biggest challenge that you encountered in this assignment? [300-350 words]

   One of the biggest challenges that I faced was developing the program is such a way that partitioned functions of the jobs to meaningful subclasses that would handle tasks. Prior to submission I had a solution that had a server helper thread that was responsible for taking the selections keys from the selector and would pass these intermediate keys to the thread pool manager thread's pending task queue. The thread pool manager was responsible for taking the keys out of his pending task queue, would convert the key into a task object, add the task to the task queue, and would then notify all of the worker threads waiting on the task queue.

   Another issue that I faced with this assignment was trying to get as much activity as I can without having to check for duplicate keys in my task queue. My first idea was that I would just maintain a linked list of batches of the selection key. By doing so I would have all the information to send the hashed message back to the client without trying to manage some data structure that would hold all of the necessary information. The issue with doing this is that when the server does another iteration of the while loop to get the selection keys that contain the selection keys that have activity, the key was already present within a batch and needed to wait until the batch was processed before any new messages could be handled by the server. I was able to solve this problem by reading the message off the key as soon as a worker thread got the task to work on.

2) If you had an opportunity to redesign your implementation, how would you go about doing this and why? [300-350 words]

   If I could completely redesigned my implementation of the thread pool, I would have relied on using a semaphore instead of using my wait notify scheme to wake up my worker threads when there were available tasks. The majority of the porblem will be explained in the next question. The solution to this problem would of been to use a semaphore on the task queue that would be able to grant access to an available thread. By doing this I could only assure that one thread would be able to pull from the task queue instead of all 10 threads trying to pull from it at the same time. By doing this, I would of definitely been able to show that the server is correctly receiving the messages at the rate in which the client is sending the messages out at.

   Something else that I would of redesigned would of been to do a better job with dealing with stagnate state on the same selection key. What I mean by this is say that we have

a new client that is trying to register with the system. The server will correctly pass off the selection key to the thread pool manager thread and will then be added to the queue. Now let's say that the queue is under heavy activity, it could take a while before that task is pulled from the task queue by one of the worker threads so now a duplicate of the same key could be in the task queue to register one client. The first task resolve would go fine and the client would be able to register with the server. The second task with the same key would result in a null pointer exception because the thread would try to try to register another client but there is no activity on the channel. To resolve this I would of done a better job at managing the behavior of the program if the key already had been put into a task queue. I would of done a better job of attaching objects to achieve this.

3) How well did your program cope with increases in the number of clients? Did the throughput increase, decrease, or stay steady? What do you think is the primary reason for this? [300-350 words]

   My server was more than capable of being able to connect with more clients without a doubt. The issue would be that the mean per client throughput and the server throughput would definitely suffer.The current implementation would have all of the worker threads be synchronized waiting on a public final reference of the task queue in the thread pool manager class. These workers would 'wake up' once the thread pool manager thread had successfully added a new task from the pending task queue. The thread pool manager thread would notify all of the workers and all of the threads would race to retrieve an item out of the task queue. This implementation of waking up the worker threads worked but it did not work to the best it could of. With this, let's say that there is one task in the queue and the manager thread wants to wake up the worker threads. All of the worker threads will then race to see if they were able to retrieve a task from the queue. The issue is in this case is that all of the threads waste their execution time in trying to determine whether or not they were able to successfully retrieve a task.

4) Consider the case where the server is required to send each client the number of messages it has received from that particular client so far. It sends this message at fixed intervals of 3 seconds. However, since each client has joined the system at different times, the times at which these messages are sent by the server would be different. For example, if client A joins the system at time T0 it will receive these messages at {T0 +3, T0 +6, T0 + 9, …} and if client B joins the system at time T1 it will receive these messages at {T1 + 3, T1 + 6, T1 + 9, …} How will you change your design so that you can achieve this? [300-400 words]

   This was originally going to be my implementation for the assignment prior to further clarification provided by the teaching assistants. The implementation can be easily done with the current design. What would need to happen is that instead of managing a linked list of batches that contain a linked list of messages is that I would maintain a linked list that will contain client node heads. Once a new client is registered with the

selector, the thread pool manager would need to add a new client node head to the linked list of registered clients. The client node head would contain the IP address of the client, the socket channel associated with the client, and a linked list of batched read messages.

Once a task has reached a worker thread, the worker thread would look at the key in the task and would be able to determine which of the clients that the message would originate from. From there the worker thread could then obtain a lock on that client's message batch and would then appropriately handle inserting the message into linked list of messages. This would almost be better than grouping all of the clients messages together because then more of the worker threads could process messages instead of waiting to acquire a lock for the messages

Prior to the worker thread adding the next message into the linked list, it would first check whether or not the batch was ready to dispatch either by checking the time or seeing if the batch has reached the max batch size. Once it was true, the worker thread would then iterate over all of the messages in the batch and using the client node head would use the socket channel and send all of the hashed messages back to the client.

5) Suppose you are planning to upgrade (or completely redesign) the overlay that you designed in the previous assignment. This new overlay must support 10,000 clients and the requirement is also that the maximum number of hops (a link in the overlay corresponds to a hop) that a packet traverses is not more than 4. Assume that you are upgrading your overlay messaging nodes using the knowledge that you have accrued in the current programming assignment; however, you are still restricted to a maximum of 10 threads in your thread-pool and 100 concurrent connections. What this means is that your messaging nodes are now servers (with thread pools) to which clients can connect. Also, the messaging nodes will now route packets produced by the clients