CSE 403 Assignment #2 (25 points)
Spring 2020

Distributed Replicated Hash Table (RHT):

This programming assignment involves the implementation of a distributed hash table where each of its element is replicated on multiple nodes (RHT) . The specification intentionally leaves undefined many components and designs. Please document your decisions/ attempts/success/failures. They will be the main topic of the grading process, along with discussing the performance evaluation of your RHT.

The submission should be done through coursesite.lehigh.edu and should consist of:
- the source code of your assignment;
- a document containing the performance plots and the findings you consider worth to be shared with me (if any).

I will grade assignments during a one-on-one meeting with you. During the one-to-one meeting.

Good luck guys!

**** Submission deadline is Thursday 26th of March through coursesite.

==========
Specification:

- The RHT offers three APIs: put(K,V)/get(K)/ put(<K1,V1>,<K2,V2>,<K3,V3>). The put(K,V) API is responsible for storing the value V associated with the key K. The get(K) API is responsible for retrieving the value associated with the key K. The put(<K1,V1>,<K2,V2>,<K3,V3>) stores the three keys K1,K2,K3 in the RHT atomically.

Keys are assumed to be of type String and values can be of any type (use generics). That means, at the time the RHT is instantiated, the programmer can decide any data type for the values to be stored in the RHT. Note that we are not interested in supporting simultaneous allocation of different data types as values.

- The semantics of the RHT operations is the following:
** The put(K,V) updates the key K in the RHT if K already exists; otherwise it adds it to the RHT. In both these cases, the put returns true. If the put operation cannot be performed due to concurrency with other operations on the same K, then the put returns false. If the put returns false, the application should retry the same put operation until it succeeds.
** The put(<K1,V1>,<K2,V2>,<K3,V3>) atomically updates the keys K1, K2, and K3 in the RHT if all of them already exist. Otherwise, for

those keys that already exist in the RHT, it atomically adds them. In both these cases, the put returns true. If the put operation cannot be performed due to concurrency with other operations on the same set of keys, then the put returns false. If the put returns false, the application should retry the same put operation until it succeeds.

- Only conflicting operations should be prevented from acting in parallel (e.g., using locks). Any non-conflicting operation should be allowed to complete regardless the existence of other concurrent operations. If you decide to use a lock table, where locks are striped, then it is fine to block operations that want to access keys whose locks map to the same lock table entry.

- Each key stored in the RHT is replicated on different nodes in the system. For simplicity you can assume the range of keys is known a priori and their mapping to nodes is also known.

- It is expected that you run your RHT through a script file that starts up all the processes in your RHT. That means, you should not manually start your processes through different terminals. Since each process knows the composition of the distributed system, each process waits until all other processes are running and then it starts executing operations.

- Deploy the RHT on at least five processes. These processes can be five different nodes in sunlab, or on Amazon, or on any other platform that offers computing nodes.

- To test your RHT, write a simple application per process that activates 8 threads and generates a configurable number of operations on the RHT (e.g., 100000)in a closed-loop. Each operation accesses random keys. You should have 20% of probability to execute a put on a single key, 20% to execute a put on multiple keys, and 60% of probability to execute a get. For the purpose of testing, use replication degree 1 and 2.

- The test application should also collect performance metrics to be included in plots. Specifically, two metrics need to be collected: average throughput (i.e., the average number of operations per second performed by all processes) and average latency (i.e., the average time needed by the system to accomplish one operation). At least two plots need to be delivered in your submission: one plot should have on the y-axis the system throughout for both the replication degrees, and one plot should have on the y-axis the average latency for both the replication degrees. Both of them have the range of keys as x-axis. The ranges of keys to use are the following {10;100;1000;10000}.

- No specific programming language it is required to be used. Choose one wisely.