Programming Assignment 2
CS 425/ECE 428 sections T4 and V4 Spring 2018

Due by 5 p.m. on April 1, 2018

Key-Value Store Consistency

This assignment should be performed by a team consisting of two students; exceptions require the instructor's approval.

Please use C, C++, Java, or Python programming languages.

The goal of this assignment is to implement distributed key-value stores that implement ***linearizability*** and ***eventual consistency*** models.

Linearizability: Refer to course notes for the definition of linearizability.

Eventual consistency: For eventual consistency, each replica should accept parameters that we will refer as 'W' and 'R'. When a write operation is issued, the new value of the specified key must be written in at least ‘W’ replicas before the write operation is considered complete. The written value should be tagged with a timestamp indicating when the write was issued. When read is issued, value of the specified key must be retrieved from at least ‘R’ replicas; the value that has the most recent timestamp among these copies should be returned in the response for the read operation. Additionally, each replica should implement the "last writer wins" rule using the timestamps.

The consistency model to be used in an execution, including any parameters (such as ‘W’ and ‘R’ for eventual consistency), should be specified as command line arguments for the replicas (you may choose a suitable command line format).

System Architecture

We will view the system as consisting of several nodes, with each node implementing the functionality of a replica server and a client both. In the implementation, for instance, you may implement each node using a single process, with different threads being used to implement the replica and client functionalities. The replicas communicate with each other using sockets. The replica and client at any given node may communicate using any method of your choice. A client always sends its put/get requests to the replica at the same node.

Please reuse the config file of MP1 to set channel delays. You may reuse code from your MP1. Your system must be able to support at least 7 nodes (i.e., 7 replicas).

Log file

Each client should produce a log file. Log file produced by client at node 1 should be named log1.txt. Name output files of other clients similarly, using their identifiers.

Each client operation (i.e., put or get) will result in two lines in the log file: one line when the operation is invoked by a client (field 6 below will be req), and another line when a client receives a response from the corresponding replica (field 6 below will be resp). Each line in the log file has 7 fields, separated by commas, as described below.

1. Unused field – pick any positive integer constant, and use that for field 1 in each line
2. Integer identifier of the client that made the request
3. get or put (depending on the operation performed)
4. key name (use a single character to name a key)
5. timestamp (an integer)
6. req (for request) and resp (for response)
7. value for req/resp for put, and value in response for get; empty otherwise)

Some examples:

```
555,2,get,y,1456940000000,req,
555,2,get,y,1456940000400,resp,4
555,2,put,x,1456940006000,req,7
555,2,put,x,1456940006250,resp,7
```

Note that there are no spaces before or after the commas in the above format. 555 above is an arbitrary positive integer.

The first line above is for invocation of a get(y) operation performed by client 2, and second line is for the corresponding response. The third line is for invocation of a put(x,7) operation by client 2, and the fourth line is for the corresponding response.

Assume that key names are a single, lowercase letter (i.e., a-z), and assume that key value is a single-digit integer in the range 0-9.

Client Operation

The client should accept on stdin the following put/get commands, and invoke the corresponding operation:

```
put keyname value
get keyname
```

In addition, the client should also accept the following commands on stdin:

```
delay millisecond
dump
```

delay command should pause the client's execution for the duration specified in millisecond. The dump command should instruct the corresponding replica to print its local values of all the keys to stdout (with one line for each “key value”). Here is an example sequence of commands issued to a client:

```
get y
get x
put x 2
delay 400
get x
dump
put z 5
delay 300
dump
```

Submission

Submit a compressed archive of your code via Compass. Include in your submission a PDF report with an overview of the code organization (at most 1.5 page), and a brief description of the functionality that each partner implemented (at most half page).

Grading rubric

60% for correct implementation of the two consistency models (30% for each model)
10% report
10% output log file format
10% for good comments and code organization
10% answers to questions during demo of the project