CE6378 Spring 2014 Project 2 Report

# Project team members:

# Jun Liang ([JXL130131@utdallas.edu](mailto:JXL130131@utdallas.edu))

# Xue Ning ([xxn120130@utdallas.edu](mailto:xxn120130@utdallas.edu))

# Introduction

To validate the correctness of our design and implementation, we performed various tests under different conditions. The methods and experiment results are in following sections.

# Methods:

The project has following requirements for correctness:

1. Random read from any non-crashed server should give meaningful result.
2. Two or more servers should agree upon any write initiated from client.
3. If a write request cannot be processed by at least two servers, then the write will fail.
4. Concurrent write requests should be performed by all the participating servers, and in the same order. No write request should be skipped in the process, even though skipping this request might not change the end value of the object.

To prove the our implementation can handle all these requirements, we designed and performed following experiments:

## Random Read/Write:

For this experiment, we use the interactive client to randomly insert/update object value from the object’s primary, secondary, and tertiary servers. After the insert/update operation finishes, we read the object’s value randomly from the primary, secondary, or tertiary server and check if the value is same as what we put in.

## Write Quorum:

For this experiment, we also use the interactive client to write an object to one of its quorum servers, after the write finishes, we will read the same object from another server in the quorum, and make sure the value is same.

## Partition of servers, majority write rule:

For this test case, we use the control client to partition servers into two clusters. Servers will be able to talk to each other in the same cluster, but will not be able to reach servers outside its cluster.

We carefully setup this cluster so we know for object with key X, its quorum servers will be in two different clusters. For example, we can assume primary and secondary are in the same cluster, and tertiary is in a different cluster.

We then use the interactive client to write an object to either the primary or secondary server and the write should succeed. We then write this object via its tertiary server and now the write operation will fail. If we read from the primary or secondary, the new object value will be returned. If we read from the tertiary server, the old value will be returned.

## Concurrent Writes:

To test concurrent writes can be processed in an identical order among all servers in the object’s quorum, we created an automated test client, which will spawn off three threads, and try to write the same object via its 3 quorum servers concurrently for 100 times.

After the client run finishes, we will check in the server log files to show the order of the object values been written, and on all three servers, the value sequence should be same.

We even deliberately partitioned the quorum server into 2 clusters so one writing thread will always fail, and the other two will always succeed. After the simulation run, we compared the results on the succeeding server, and make sure the write happens in the same order.

# Results: