CE6378 Spring 2014 Project 2 Report

# Project team members:

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

# Xue Ning ([xxn120030@utdallas.edu](mailto:xxn120030@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.

## Simulate Failure:

To test the implementation can handle server crash failure, we implemented a control utility that can selectively turn off servers to simulate crash. When a server becomes unavailable, it will not process any request from any other servers or clients. In this case, the other server needs still to follow the majority rule for write request.

In our experiment, we first turn off one server, and let this server be in the write quorum of object X. We will use one client to write object X initiated from any of the remaining two servers. In this case, the write will still succeed, as there is a 2-server quorum available. The client can read from any of these 2 servers and get the correct value back.

We will then turn off one more server in object X’s quorum, and then the write via the remaining server will fail, as there is no majority agreement can be reached anymore.

## Command Description:

We provide interactive user input at the client side, the commands supported by user input command line are listed as follows:

**1) Put object command:**

Command format: Server\_id, put, object\_key, object\_value

Server\_id: a represents the primary server, b represents the secondary server,

c represents the tertiary server.

Example: a, put, 1, 10

Implement put object: object\_key = 1, object value =10 by accessing primary

server.

**2) Get object command:**

Command format: Server\_id, get, object\_key

Server\_id: a represents the primary server, b represents the secondary server, c represent the tertiary server.

Example: a, get, 1

Implement get object: object\_key = 1 by accessing primary server.

**3) Concurrent write command:**

Command format: Server\_id, wreq, object\_key, total\_num\_of\_request

Server\_id: a represents the primary server, b represents the secondary server, c represent the tertiary server.

total\_num\_of\_request: total number of generating put request.

Example: a, wreq,1,10

Implement 10 times of put operation with given object\_key = 1 by accessing primary server.

# Results:

Below tables lists the detailed information of test cases we have implemented to verity our design satisfying the requirements.

## Result of Read/Write /Write Quorum Operations:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Item** | **Test Case Description** | **Expected Result** | **Test Result** |
| read\_write\_001 | A client puts an object given key = 0,value = 5 to primary server:  Command is "a,put,0,5" | put operation is successful | pass |
| read\_write\_002 | A client gets an object given key = 0 from primary server Command is "a,get,0" | Server returns value = 5 for object with key = 0 | pass |
| read\_write\_003 | A client gets an object given key = 0 from secondary server  Command is "b,get,0" | Server returns value = 5 for object with key = 0 | pass |
| read\_write\_004 | A client gets an object given key = 0 from tertiary server Command is "c,get,0" | Server returns value = 5 for object with key = 0 | pass |
|  |  |  |  |
| read\_write\_005 | A client puts an object given key = 0,value =6 to secondary server:  Command is "a,put,0,6" | put operation is successful | pass |
| read\_write\_006 | A client gets an object given key = 0 from primary server Command is "a,get,0" | Server returns value = 6 for object with key = 0 | pass |
| read\_write\_007 | A client gets an object given key = 0 from secondary server  Command is "b,get,0" | Server returns value = 6 for object with key = 0 | pass |
| read\_write\_008 | A client gets an object given key = 0 from tertiary server Command is "c,get,0" | Server returns value = 6 for object with key = 0 | pass |
|  |  |  |  |
| read\_write\_009 | A client puts an object given key = 0,value = 7 to secondary server:  Command is "a,put,0,7" | put operation is successful | pass |
| read\_write\_010 | A client gets an object given key = 0 from primary server Command is "a,get,0" | Server returns value = 7 for object with key = 0 | pass |
| read\_write\_011 | A client gets an object given key = 0 from secondary server  Command is "b,get,0" | Server returns value = 7 for object with key = 0 | pass |
| read\_write\_012 | A client gets an object given key = 0 from tertiary server Command is "c,get,0" | Server returns value = 7 for object with key = 0 | pass |
|  |  |  |  |
| read\_write\_013 | A client puts an object given key = 1,value = 10 to primary server:  Command is "a,put,1,10" | put operation is successful | pass |
| read\_write\_014 | A client gets an object given key = 1 from primary server Command is "a,get,1" | Server returns value = 10 for object with key = 1 | pass |
| read\_write\_015 | A client gets an object given key = 1 from secondary server  Command is "b,get,1" | Server returns value = 10 for object with key =1 | pass |
| read\_write\_016 | A client gets an object given key = 1 from tertiary server Command is "c,get,1" | Server returns value = 10 for object with key =1 | pass |
|  |  |  |  |
| read\_write\_017 | A client puts an object given key = 1,value =11 to secondary server:  Command is "a,put,1,11" | put operation is successful | pass |
| read\_write\_018 | A client gets an object given key = 1 from primary server Command is "a,get,1" | Server returns value = 11 for object with key = 1 | pass |
| read\_write\_019 | A client gets an object given key = 0 from secondary server  Command is "b,get,1" | Server returns value = 11 for object with key = 1 | pass |
| read\_write\_020 | A client gets an object given key = 0 from tertiary server Command is "c,get,1" | Server returns value = 11 for object with key = 1 | pass |
|  |  |  |  |
| read\_write\_021 | A client puts an object given key = 0,value = 12 to secondary server:  Command is "a,put,1,12" | put operation is successful | pass |
| read\_write\_022 | A client gets an object given key = 0 from primary server Command is "a,get,1" | Server returns value = 12 for object with key =1 | pass |
| read\_write\_023 | A client gets an object given key = 0 from secondary server  Command is "b,get,1" | Server returns value = 12 for object with key = 1 | pass |
| read\_write\_024 | A client gets an object given key = 0 from tertiary server Command is "c,get,1" | Server returns value = 12 for object with key =1 | pass |
|  |  |  |  |
| read\_write\_025 | A client puts an object given key = 2,value = 20 to primary server:  Command is "a,put,2,20" | put operation is successful | pass |
| read\_write\_026 | A client gets an object given key = 2 from primary server Command is "a,get,2" | Server returns value = 20 for object with key =2 | pass |
| read\_write\_027 | A client gets an object given key = 2 from secondary server  Command is "b,get,2" | Server returns value = 20 for object with key = 2 | pass |
| read\_write\_028 | A client gets an object given key = 2 from tertiary server Command is "c,get,2" | Server returns value = 20 for object with key =2 | pass |
|  |  |  |  |
| read\_write\_029 | A client puts an object given key = 3,value = 30 to primary server:  Command is "a,put,3,30" | put operation is successful | pass |
| read\_write\_030 | A client gets an object given key = 3 from primary server Command is "a,get,3" | Server returns value = 30 for object with key = 3 | pass |
| read\_write\_031 | A client gets an object given key = 3 from secondary server  Command is "b,get,3" | Server returns value = 30 for object with key = 3 | pass |
| read\_write\_032 | A client gets an object given key = 3 from tertiary server Command is "c,get,3" | Server returns value = 30 for object with key =3 | pass |
|  |  |  |  |
| read\_write\_033 | A client puts an object given key = 4,value = 40 to primary server:  Command is "a,put,4,40" | put operation is successful | pass |
| read\_write\_034 | A client gets an object given key = 4 from primary server Command is "a,get,4" | Server returns value = 40 for object with key = 4 | pass |
| read\_write\_035 | A client gets an object given key = 4 from secondary server  Command is "b,get,4" | Server returns value = 40 for object with key = 4 | pass |
| read\_write\_036 | A client gets an object given key = 4 from tertiary server Command is "c,get,4" | Server returns value = 40 for object with key =4 | pass |
|  |  |  |  |
| read\_write\_037 | A client puts an object given key = 5,value = 50 to primary server:  Command is "a,put,5,50" | put operation is successful | pass |
| read\_write\_038 | A client gets an object given key = 5 from primary server Command is "a,get,5" | Server returns value = 50 for object with key = 5 | pass |
| read\_write\_039 | A client gets an object given key = 5 from secondary server  Command is "b,get,5" | Server returns value = 50 for object with key = 5 | pass |
| read\_write\_040 | A client gets an object given key = 5 from tertiary server Command is "c,get,5" | Server returns value = 50 for object with key =5 | pass |
|  |  |  |  |
| read\_write\_041 | A client puts an object given key = 6,value =60 to primary server:  Command is "a,put,6,60" | put operation is successful | pass |
| read\_write\_042 | A client gets an object given key = 6 from primary server Command is "a,get,6" | Server returns value = 60 for object with key = 6 | pass |
| read\_write\_043 | A client gets an object given key = 6 from secondary server  Command is "b,get,6" | Server returns value = 60 for object with key = 6 | pass |
| read\_write\_044 | A client gets an object given key = 6 from tertiary server Command is "c,get,6" | Server returns value = 60 for object with key = 6 | pass |
|  |  |  |  |
| read\_write\_045 | A client gets an non- existing object:  Implement "a,put,1,10"; Then implement "a,get,2"; Then implement "a,get,100". | The UI returns failed on getting the object | pass |
|  |  |  |  |
| read\_write\_046 | Implement "a,put,8,80" (Primary server: 1,second:2,teritery:3) Implement "a,put,15,81" Implement "a,put,22,82" Implement "a,put,29,83" Implement "a,put,36,84" Implement "a,put,43,85" Implement "a,put,50,86" Implement "a,put,57,87" Implement "a,put,64,88" Implement "a,put,71,89" Implement "a,put,78,90" Implement "a,put,78,100" Then implement "a,put,79,110"(Primary server: 2,....) | The put operation will be successful | pass |

## Result of Concurrent Write Operations:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Item** | **Test Case Description** | **Expected Result** | **Test Result** |
| concurrent\_write\_001 | a,wreq,0,20 b,wreq,0,20 | Write operation happens with the same order on all replicas | Pass. |
| concurrent\_write\_002 | a,wreq,0,30 b,wreq,0,30 c,wreq,0,30 | Write operation happens with the same order on all replicas | Pass. |
| concurrent\_write\_003 | a,wreq,6,30 b,wreq,6,30 c,wreq,6,30 | Write operation happens with the same order on all replicas | Pass. |
| concurrent\_write\_004 | 1) a,put,0,10 2)a,get,0 / b,get,0 /c get 0 return 10 3)a,put,6,60 4)a,get6 /b,get,6/ c,get,6 return 60 return 60 5) Isolate,0,1 6)c,wreq,0,3 return failed; c,get,0 return 10 7) a,wreq,0,30 return ok 8)b,wreq,0,30 return ok and write operations happen in the same order at "a “server and "b" server 9) c,get,0 return old value of 10; a,get,0 /b,get,0 return new value 10) a,wreq,6,30 return failed; a,get,6 return 60 11)b,wreq,6,30 return ok 12)c,wreq,6,30 return ok 13)a,get,6 return old value of 60 14)b,get,6/c,get,6 return new value | Write operation happens with the same order on two replicas of the major group | Pass. |
| concurrent\_write\_005 | 1)Isolate,1,2 2)a,wreq,0,30 return failed 2)b,wreq,0,30 3)c,wreq,0,30 | Write operation happens with the same order on two replicas of the major group | Pass. |

## Result of Network Partition Operations:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Item** | **Test Case Description** | **Expected Result** | **Test Result** |
| network\_partition\_001 | 1)a,put,0,6 2)c,get,0 return 6 3)b,get,0 return 6 4)c,get,0 return 6 5)Isolate,0,1 6)c,put,0,300 return failed 7)a,put,0,100 return successful 8)a,get,0 return 100 9)b,get,0 return 100 10)c,get,0 return 6 11)b,put,0,1000 12)b,get,0 return 1000 13)a,get,0 return 1000 14)c,get,0 return 6 | After implement network partition: 1)We are not able to update the value object by server2. We only can read old value of object0 from server2. 2) But we can update the value object by server 0 or server 1. We can read new value of object0 from either server0 or server1. | pass |
| network\_partition\_002 | 1) a,put,2,200 2) a,get,2 return 200 3) b,get,2 return 200 4) c,get,2 return 200 5) a,put,3,300 6) a,get,3 return 300 7) b,get,3 return 300 8) c,get,3 return 300 9) Isolate,0,1,2,3 10) c,put,2,2000 return failed 11) c,get,2,return 200 12) a,put,2,400 return successful 13) a,get,2 return 400 14) b,get,2 return 400 15) c,get,2 return 200 16) a,put,3,3000 return failed 17) b,put,3, 600 return ok 18) b,get,3 return 600 19) c, get,3 return 600 20) a,get,3 return 300 21) c,put,3,6000 return successful 22) c,get,3 return 6000 23) b,get,3 return 6000 24) a,get,3 return 300 | After implement network partition:  1) we are not able to update the value of object2 by server4, and we are not able to update the value of object3 by server3. We only can read old value of object2 by server4, and read old value of object3 by server3. 2) We can update of the value of object2 by server2, and server3. We can read new value of object2 by server2 and server3.We also can update of the value of object3 by server4 and server5.We can read new value of object3 by server4 and server5. | pass |

# Performance Statistics:

We intend to collect how long it takes to respond a write/read request issued by a client. Moreover, for the write request, we want to collect how many messages exchanged at servers to guarantee the put operation for an object will be implemented in the same order with all related servers.

## Observation on response time & exchanged message of write request without network partition

We implement “put” operation for multiple times and then calculate the average response time.

The response time for “put” operation is 373.1ms. And when three servers are available for a specific object, each put operation costs 8 messages exchanged among three servers to guarantee that the put operation is implemented with the same order among the three servers.

For example, server0 receives write request from a client, below figure shows how server0 exchanges messages with server1 and server2 to guarantee the write operation happens in the same order among the three servers:

SERVER\_PUT\_OBJECT

SERVER\_PUT\_OK

SERVER\_COMMIT\_OBJECT

SERVER\_COMMIT\_OBJECT

SERVER\_TO\_SERVER\_COMMIT\_OK

SERVER\_COMMIT\_OBJECT

SERVER\_PUT\_OK

SERVER\_PUT\_OBJECT

The details on the collected data sets of above observation is listed as below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Implement put operations without network partition** | | | | | |
| **Input\_Command** | **Operation\_Type** | **Operation \_Seq\_Num** | **ObjectId** | **ResponseTime(ms)** | **Exchanged messages  at servers** |
| a,wreq,0,10 | put | 1 | 0 |  | 8 |
| put | 2 | 0 | 353 | 8 |
| put | 3 | 0 | 391 | 8 |
| put | 4 | 0 | 392 | 8 |
| put | 5 | 0 | 390 | 8 |
| put | 6 | 0 | 388 | 8 |
| put | 7 | 0 | 388 | 8 |
| put | 8 | 0 | 397 | 8 |
| put | 9 | 0 | 389 | 8 |
| put | 10 | 0 | 383 | 8 |
| Average Response Time | | | | 347.1 |  |
| a,wreq,1,10 | put | 1 | 1 | 334 | 8 |
|  | put | 2 | 1 | 370 | 8 |
|  | put | 3 | 1 | 369 | 8 |
|  | put | 4 | 1 | 371 | 8 |
|  | put | 5 | 1 | 369 | 8 |
|  | put | 6 | 1 | 370 | 8 |
|  | put | 7 | 1 | 369 | 8 |
|  | put | 8 | 1 | 373 | 8 |
|  | put | 9 | 1 | 369 | 8 |
|  | put | 10 | 1 | 374 | 8 |
| Average Response Time | | | | 366.8 |  |
| a,wreq,2,10 | put | 1 | 2 | 371 | 8 |
|  | put | 2 | 2 | 372 | 8 |
|  | put | 3 | 2 | 372 | 8 |
|  | put | 4 | 2 | 374 | 8 |
|  | put | 5 | 2 | 376 | 8 |
|  | put | 6 | 2 | 371 | 8 |
|  | put | 7 | 2 | 372 | 8 |
|  | put | 8 | 2 | 371 | 8 |
|  | put | 9 | 2 | 372 | 8 |
|  | put | 10 | 2 | 372 | 8 |
| Average Response Time | | | | 335.2 |  |
| a,wreq,3,10 | put | 1 | 3 | 375 | 8 |
|  | put | 2 | 3 | 373 | 8 |
|  | put | 3 | 3 | 375 | 8 |
|  | put | 4 | 3 | 373 | 8 |
|  | put | 5 | 3 | 370 | 8 |
|  | put | 6 | 3 | 372 | 8 |
|  | put | 7 | 3 | 371 | 8 |
|  | put | 8 | 3 | 373 | 8 |
|  | put | 9 | 3 | 371 | 8 |
|  | put | 10 | 3 | 371 | 8 |
| Average Response Time | | | | 372.4 |  |
| a,wreq,4,10 | put | 1 | 4 | 426 | 8 |
|  | put | 2 | 4 | 377 | 8 |
|  | put | 3 | 4 | 376 | 8 |
|  | put | 4 | 4 | 371 | 8 |
|  | put | 5 | 4 | 377 | 8 |
|  | put | 6 | 4 | 374 | 8 |
|  | put | 7 | 4 | 378 | 8 |
|  | put | 8 | 4 | 375 | 8 |
|  | put | 9 | 4 | 372 | 8 |
|  | put | 10 | 4 | 373 | 8 |
| Average Response Time | | | | 379.9 |  |
| a,wreq,5,10 | put | 1 | 5 | 415 | 8 |
|  | put | 2 | 5 | 399 | 8 |
|  | put | 3 | 5 | 401 | 8 |
|  | put | 4 | 5 | 398 | 8 |
|  | put | 5 | 5 | 399 | 8 |
|  | put | 6 | 5 | 396 | 8 |
|  | put | 7 | 5 | 385 | 8 |
|  | put | 8 | 5 | 389 | 8 |
|  | put | 9 | 5 | 396 | 8 |
|  | put | 10 | 5 | 397 | 8 |
| Average Response Time | | | | 397.5 |  |
| a,wreq,6,10 | put | 1 | 6 | 413 | 8 |
|  | put | 2 | 6 | 413 | 8 |
|  | put | 3 | 6 | 411 | 8 |
|  | put | 4 | 6 | 417 | 8 |
|  | put | 5 | 6 | 410 | 8 |
|  | put | 6 | 6 | 424 | 8 |
|  | put | 7 | 6 | 413 | 8 |
|  | put | 8 | 6 | 413 | 8 |
|  | put | 9 | 6 | 397 | 8 |
|  | put | 10 | 6 | 418 | 8 |
| Average Response Time | | | | 412.9 |  |
| Average response time for all above operations | | | | 373.1 |  |

## Observation on response time & exchanged message of write request with network partition

We implement “put” operation for multiple times and then calculate the average response time.

The response time for “put” operation is 240.85ms which is smaller than the response time when three servers available, because if the majority group includes two servers, it costs 4 messages between the two servers to achieve consensus to implement the write operation.

For example, server0 and server1 form the majority group, server2 is in the minority group. Server0 receives write request from a client, below figure shows how server0 exchanges messages with server1 to guarantee the write operation happens in the same order for server0 and server1:

SERVER\_PUT\_OBJECT

SERVER\_PUT\_OK

SERVER\_COMMIT\_OBJECT

SERVER\_COMMIT\_OBJECT

The details on the collected data sets of above observation is listed as below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Implement put operations with network partition Isolate,0,1** | | | | | |
| **Input\_Command** | **Operation\_Type** | **Operation \_Seq\_Num** | **ObjectId** | **ResponseTime(ms)** | **ResponseTime(ms)** |
| a,wreq,0,10 | put | 1 | 0 | 220 | 4 |
| put | 2 | 0 | 208 | 4 |
| put | 3 | 0 | 209 | 4 |
| put | 4 | 0 | 209 | 4 |
| put | 5 | 0 | 206 | 4 |
| put | 6 | 0 | 207 | 4 |
| put | 7 | 0 | 206 | 4 |
| put | 8 | 0 | 207 | 4 |
| put | 9 | 0 | 206 | 4 |
| put | 10 | 0 | 213 | 4 |
| Average Reponse Time for server "a" | | | | 209.1 |  |
| b,wreq,0,10 | put | 1 | 0 | 245 | 4 |
| put | 2 | 0 | 285 | 4 |
| put | 3 | 0 | 265 | 4 |
| put | 4 | 0 | 299 | 4 |
| put | 5 | 0 | 343 | 4 |
| put | 6 | 0 | 246 | 4 |
| put | 7 | 0 | 356 | 4 |
| put | 8 | 0 | 238 | 4 |
| put | 9 | 0 | 223 | 4 |
| put | 10 | 0 | 226 | 4 |
| Average Response Time for server "b" | | | | 272.6 |  |
| Average Reponse Time for server "a","b" | | | | 240.85 |  |

## Observation on response time & exchanged message of read operation

We implement “get” operation for multiple times and then calculate the average response time. The response time for “put” operation is 46.2ms.

The details on the collected data sets of above observation is listed as below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input\_Command** | **Operation\_Type** | **Operation \_Seq\_Num** | **ObjectId** | **ResponseTime(ms)** |
| a,get,0 | get | 1 | 0 | 41 |
| get | 2 | 0 | 41 |
| get | 3 | 0 | 41 |
| get | 4 | 0 | 41 |
| get | 5 | 0 | 41 |
| get | 6 | 0 | 41 |
| get | 7 | 0 | 61 |
| get | 8 | 0 | 74 |
| get | 9 | 0 | 57 |
| get | 10 | 0 | 42 |
| Average response time for server "a" | | | | 48 |
| b,get,0 | get | 1 | 0 | 41 |
|  | get | 2 | 0 | 52 |
|  | get | 3 | 0 | 40 |
|  | get | 4 | 0 | 40 |
|  | get | 5 | 0 | 40 |
|  | get | 6 | 0 | 40 |
|  | get | 7 | 0 | 40 |
|  | get | 8 | 0 | 40 |
|  | get | 9 | 0 | 44 |
|  | get | 10 | 0 | 42 |
| Average response time for server "b" | | | | 41.9 |
| c,get,0 | get | 1 | 0 | 40 |
|  | get | 2 | 0 | 54 |
|  | get | 3 | 0 | 40 |
|  | get | 4 | 0 | 40 |
|  | get | 5 | 0 | 43 |
|  | get | 6 | 0 | 40 |
|  | get | 7 | 0 | 40 |
|  | get | 8 | 0 | 59 |
|  | get | 9 | 0 | 85 |
|  | get | 10 | 0 | 47 |
| Average response time for server "c" | | | | 48.8 |
| Average response time for server "a,b,c" | | | | 46.2 |