CSC469 Assignment 3

Alex K. Chang
Alda Liu

1000064681
1000798014

1.1 Introduction

For this assignment, we explored fault tolerance techniques outlined in [1] to handle server failure for a key-value service. More specifically for our implementation, our key-value service can handle a single key-value server failing, thus causing the metaserver to go into recovery mode; which then executes the recovery steps mentioned in [1].

1.2 Additional Modifications

Following the basic outline from [1], we implemented additional modifications to our key value service. Mainly, we added an additional thread to separate requests from client and server. In mserver.c, we created a new thread during the start of the mserver loop. The new thread is to exclusively handle the client requests while the old thread handles server tasks (requests and responses to servers), as well as updating the metadata for each server.

2 Conceptual Questions

2.1 If a server could run out of space for keys, it would not be able to add new keys to its primary set, nor would it have enough space for its secondary replica. In case of a crash of a server, if its secondary server ran out of space it might not have the complete replica of keys and information would be lost.

Real-world implementations tend to warn the user well in advance if the amount of space left crosses a threshold. In live systems, there should also be alarms which trigger once a certain metric crosses a threshold, in this case, memory. These alarms would allow the developer to react to the memory usage and act appropriately (buying more memory or paying for a bigger instance).

2.2 The complexity of the design would be simpler. In terms of performance, read would be fast since no bottleneck for client requests as all keys are replicated on all servers. When doing a write; update or delete, must send invalidate or updated key to all servers, can be slow.

2.3 If failures could happen during the recovery process, the severity of the failure would depend on which server fails. If the new server being brought up (Saa) fails again, then it wouldn't matter that much, since it's already failed and its key sets still live on the other servers, so we can just bring another server up. If Sb or Sc fails then, it would be a lot worse. We could potentially lose keys if we have more than one failed server at a time in our model, since the primary/secondary of the failing server lives on the other failed server.

2.4 The system must have more than 2n + 1 replicas machines which will allow the system to tolerate n simultaneous failures. This is because for every n simultaneous failure, you need to add n extra key sets. So for 2 simultaneous failures, each server would need to carry 3 key sets. This is because if primary set X and secondary set X were only the 2 machines which failed, then it would be impossible to recover. Thus we need 2n + 1 replica machines for n extra key sets for n simultaneous failures.

2.5 If changed to this type of model, say that a client requests "PUT" for a key into the server. "Get" requests for that same key, in this model, we can still route the client or other clients to the primary server to return back the value for the given key. However, if the any client chooses to write "PUT" a new value to that key, they will be sent back an error if the previous updates to that key have not been written to the secondary server yet.

2.6 Elastically scaling nodes is a great idea and is used widely in cloud computing today. Adding nodes in response to increasing

load on the system ensures that none of your servers becomes overworked and starts diminishing in throughput. It would be terrible if your system starts taking a long time to service requests due to one server under stress. Dynamically adding nodes to balance loads remedies this problem. Also, removing nodes during underloaded times can help save server costs. Instead of keeping around and paying for instances when they aren't needed, just remove them. During recovery, if we could elastically add nodes, we could just bring up a new server to take over while the recovery process for the failed server takes place behind-the-scenes, given the users an illusion of zero-downtime.

References:

[1] http://www.teach.cs.toronto.edu/~csc469h/fall/assignments/a3/a3.shtml