2. (15+15 marks) Based on the aforementioned ideas, design and implement the fault tolerant version of the Ivy architecture (the primary and back replica for the CM with consistency + related changes in the basic Ivy protocol).

In this implementation, I created a back replica of the main CM with strong consistency replication techniques. In this case, the main CM makes a replication call to the backup CM every time the metadata is updated. It then waits for a response before continuing to the next section of sending messages. This ensures strong consistency between the client and the replica.

The backup CM will also be making a health check call to the main CM to see if it should take over or be the backup. In this implementation, the main CM will always be the main CM if it comes back up, similar to how the Bully Algorithm works.

Rather than having individual queues for both CM, I used the concept of a reverse proxy, and the clients only had to send to that one common queue they shared. The proxy is defined as a global variable of channel.

For the purpose of this assignment, we only implement safety where the CM fails in between requests. There are more cases such as failing during handling requests, but there are too many scenarios to consider and cases to implement to keep everything perfectly safe.

3. (10 marks) Discuss whether your fault tolerant version of Ivy still preserves sequential consistency or not.

This version of ivy still preserves sequential consistency.

This proxy functions similarly to a message queue, ensuring that requests are processed sequentially. Each request is only consumed by the CM upon being handled, which prevents concurrent request consumption for a single page, thus reducing the risk of breaching sequential consistency.

Additionally, the proxy is equipped to manage multiple pages, utilizing a map structure in Go. This setup guarantees that requests for each page are handled in the order they are received, thereby preserving the overall sequential consistency of the system.

(15 marks) Without any faults, compare the performance of the basic version of Ivy protocol and the new fault tolerant version using requests from at least 10 clients.

For both experiments, they were run with 10 processors, 1 page, 100 requests for a total of 10 repetitions. The average time taken for each repetition is as follows:

Basic Version – 1.441 seconds
Fault Tolerant – 1.455 seconds

For the fault tolerant version, it took longer to complete as there is redundancy that has been introduced, being the backup CM. This backup CM in this implementation requires for replication of data every time the metadata is changed. This additional effort to pass messages and replicate data consumes more CPU usage and time, causing the fault tolerant one to be of slightly lower performance.

(15 marks) Evaluate the new design in the presence of a single fault – one CM fails only once. Specifically, you can simulate two scenarios a) when the primary CM fails at a random time, and b) when the primary CM restarts after the failure. Compare the performance of these two cases with the equivalent scenarios without any CM faults.

In this question, I am assuming that this question is asking to simulate the CM failing once, going through both scenarios simultaneously when the primary CM goes down and restarts later. Otherwise the latter "scenario" alone makes no sense. If you desire to just go through the first scenario of just permanent fault, comment out cms[0].comeAlive() in simulationNumber == 1

For both experiments, they were run with 10 processors, 1 page, 100 requests for a total of 10 repetitions. The average time taken for each repetition is as follows:

No Faults – 1.455 seconds
Single Fault – 1.532 seconds

For down time, the time that the main CM will be down is 150 milliseconds before coming back alive.

As I have introduced a timeout required for replication of data (for simulation purposes), this means that if the backup CM down, to replicate data, the main CM will wait for the timeout before determining that the backup is down.

Therefore, while the main CM is down, replication attempts will be delayed, causing the whole system to be slower during this period. Thus, the single fault scenario would have reduced performance.

(15 marks) Evaluate the new design in the presence of multiple faults for primary CM – primary CM fails and restarts multiple times. Compare the performance with respect to the equivalent scenarios without any CM faults.

For both experiments, they were run with 10 processors, 1 page, 100 requests for a total of 10 repetitions. The average time taken for each repetition is as follows:

No Faults – 1.455 seconds
Main Multiple Faults – 2.112 seconds

For down time, it will take 150 milliseconds till the first down time, then another 150 milliseconds before coming back alive. This cycle repeats itself until 100 requests have been processed.

Following up from the previous question, as there are multiple faults, this would stack on the waiting time during replication attempts, further increasing the time it takes to complete a request. In total, through the entire process, the performance decreases significantly as compared to one without fault.

The huge increase is due to 1 millisecond of waiting, which is much higher than a regular message passing time, making the increase a little more exaggerated.

(15 marks) Evaluate the new design in the presence of multiple faults for primary CM and backup CM – both primary CM and backup CM fail and restart multiple times. Compare the performance with respect to the equivalent scenarios without any CM faults.

For both experiments, they were run with 10 processors, 1 page, 100 requests for a total of 10 repetitions. The average time taken for each repetition is as follows:

No Faults – 1.455 seconds
Multiple random faults – 2.163 seconds

For faults, the system will randomize which CM goes down. It will then take 150 milliseconds till the first down time, then another 150 milliseconds before coming back alive. This cycle repeats itself until 100 requests have been processed.

The effect to the performance is like the previous question. Compared to without faults, the additional performance trade off comes from trying to replicate but failing to, resulting in more wait time and a weaker performance.