**Question 1:**

**A)**

Make a Distributed round robin based scheduler backed by redis. The following should be considered.

- Image you have 5 servers s1,s2,s3,s4,s5) running behind a load balancer
- You are designing a round robin algorithm to distribute the load or traffic or https requests (h1,h2….hn ) to five of the servers si (i=1 to 5)
- Imagine you are having such 5 load balancers lb1, lb2, lb3
- All requests h1,h2…hn hit any of the load balancer
- Design round robin algorithm which is going to distribute any https request hk (k =1,2….) which hits any of the 3 load balancers and load balancer is going to distribute it to any of the servers si (i=1 to 5)
- Redis to be used for achieving distributed round robin.
- Now import the final code or module or library you have created lets say roundrobin.js, import it as a module from test.js file.
- test.js will be having setInterval running at 50ms interval. After 100 iterations stop the loop ( 5seconds). Simultaneously run parallely test.js 3 more instances. So in total 4 instances of test.js will be running parallely.
- Record the output. The output must contain the unix timestamp, counter value tracking iteration count, and the server to which the round robin algorithm as assigned (either s1 or s2 etc….)

**B)**

Extending **Question A),** write down the functions to dynamically add new servers or remove new servers that round robin algorithm is going to assign. Lets say add() function will add new servers s6,s7, s8. Now your algorithm must start assigning to new servers too without restarting the script.
remove() is going to remove the servers lets say from s1…s5 to now s1,s2,s3 only. Now you must distribute laid only between these 3 servers.
TO test the functions add(), remove(), in test.js, exactly after 100 iterations, call add, add 3 more servers s6,s7,s8 and stop after 100 iterations (in total 200 iterations)
TO test remove(), repeat the same thing like add(), only the fact that u remove servers s4,s5 after 100th iterations, and then run another 100 iterations and stop.
(NOTE: u should be calling add() or remove() only from one test.js instance and remaining 3 test.js instances should not add or remove. So when you launch test.js make it as an argument based script so running for example **node test.js add** will add the items, and run rest of 3 other instances using **node test.js** only

To test both add, remove, after 100 iterations perform add call, after 50 iterations (At 150th iteration) call remove, and at 200th iteration stop the loop.
(make a argument lik addremove and run **node test.js addremove**)

**C)**
Extend Script A, B with additional functionality, use the hash functions, to ensure that same iteration counter values will always hit the same server.
For Eg: if counter value is 1, it will hit server 1, 2 => s2, 3=> s3, 4=> s4, 5 => s5,
6 => s1, 7 => s2, …. 10 => s5
11 => s1,...
16 => s1 and so on……
You should be demonstrating consistent hashing concepts.
Run only one instance of test.js, and put counter and break loop after 20 iterations.

**D)**
Extending script A,B,C, add random round robin allocation rule, which is going to allocate randomly any server s1,s2,s3,s4,s5.
When you run the test.js (4 instance of test.js), u need to record what server is allocated.
After break at 100th loop, you need to out on the distribution of the requests to the servers s1,s2,s3,s4,s5. Figure out it is going to follow gaussian distribution or a random distribution.
Check if all 4 test.js files are going to follow the same distribution or not. X-axis would be the counter value and y axis the server value. You can make a graph to find out distribution or put on mathematical formulation to find distribution. (No need to programming draw a graph, ther calculations and graphs can be made on paper.)

**E)**
FInally error handling to be handled wisely. Start the script as mentioned in **A),** run upto 4 instances. This time set the setTImeout Interval to 1 second. Break at 100th loop.
Suddenly after 25 iterations, stop the redis server or crash the redis server. Look how the code is behaving. You should be handling the disconnects to redis and demonstrate error handling.
After 10 seconds, start redis server again, Your script should be able to reconnect to redis server automatically, and resume from where it is left.

In real life example, if redis is down, how would you distribute the requests now. U cannot drop the requests, Propose a strategy how you are going to handle this situation. (DO not mention about redis redundancy or using failover redis node or a cluster based solution.) Propose a solution how programmatically you can find an alternative for it.

**NOTE:**

- DO not use any of the third party libraries at all.
- Only the redis module or ioredis can be used.
- Programming language strictly nodejs/js should only be used.