

Programming Assignment 3: Internet of Things - Fault tolerance, Replication, and Consistency

RESULTS DOCUMENTS

SUBMITTED BY: RAHUL RAJ AND OLENKA DEY

This document shows the generated output from our program for different activities in the smart home. This documents also includes all the scenario such as Replication, Dynamic Load Balancing, Consistency, cache implementation, cache replacement policy, fault tolerance implementation, failure recovery implementation as well as paxos implementation.

We have considered the same sensors and smart Devices design as in the last lab, we have considered the same three sensors (temperature sensor, motion sensor and Door Sensor) and smart devices (smart bulb device and smart outlet device). Our gateway is still 2 tier server client architecture, but now it is replicated, having 2 different replicas.

• REPLICATION AND LOAD BALANCING

Some of the devices are connected to Gateway-1 and some are connected to Gateway-2.

Devices connected to Gateway-1.

```
=====
All device registered to gateway - 1 are !!!
Node Name and Node Type are: {'doorSensor': 'Sensor', 'tempSensor': 'Sensor', 'smartOutlet': 'Device'}
Node Name and ID are: {'doorSensor': 4, 'tempSensor': 2, 'smartOutlet': 3}
=====
```

Devices connected to Gateway-2.

```
=====
All device registered to gateway - 2 are!!!
Node Name and Node Type are: {'motionSensor': 'Sensor', 'smartBulb': 'Device'}
Node Name and ID are: {'motionSensor': 6, 'smartBulb': 5}
=====
```

We have set a threshold of 3 devices/sensors to one of the Gateway. So, the devices/sensors starts with registering to Gateway-1 but as soon as the threshold of 3 is reached load balancing comes into picture and the other devices/sensors registers themselves to the Gateway-2.

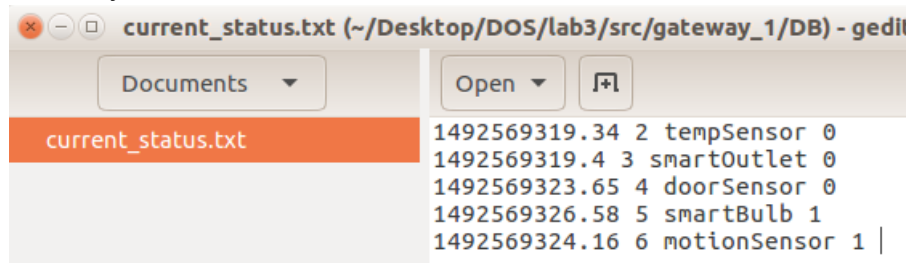
```
rahul@raj:~/Desktop/DOS/lab3/CODE/src$ ./run_me.sh 1 1
Starting and registering All the Devices and Sensors with the Gateway.
Waiting...
tempSensor connected to 8001
Registering tempSensor with gateway 1 with node ID 2
Adding the cache in Gateway-1.
Sending to the backend : node ID is 2 , Sensor/Device name is tempSensor , current state is 0
smartOutletDevice connected to 8001
Registering smartOutlet with gateway 1 with node ID 3
Cache full in GATEWAY-1 Evicting the cache using LRU and caching the current item
Sending to the backend : node ID is 3 , Sensor/Device name is smartOutlet , current state is 0
doorSensor connected to 8001
rahul@raj:~/Desktop/DOS/lab3/CODE/src$ Registering doorSensor with gateway 1 with node ID 4
Cache full in GATEWAY-1 Evicting the cache using LRU and caching the current item
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
smartBulbDevice connected to 9001
Registering smartBulb with gateway 2 with node ID 5
Adding the cache in Gateway-2.
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 0
motionSensor connected to 9001
Registering motionSensor with gateway 2 with node ID 6
Cache full in GATEWAY-2 Evicting the cache using LRU and caching the current item
Sending to the backend : node ID is 6 , Sensor/Device name is motionSensor , current state is 0
```

The figure on the left shows the replicated nature of the Gateway and how the devices/sensors are getting connected to 2 different gateways depending on the load. As soon as we start the program by running `run_me.sh`, it registers all the devices and sensors, tempSensor, smartOutlet and doorSensor are connected to Gateway-1 having port number 8001 and the smartBulb and motionSensor are connected to the Gateway-2 having port number 9001.

- **CONSISTENCY TECHNIQUE**

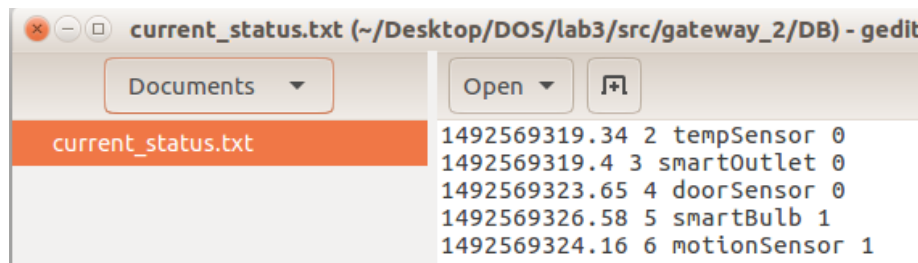
We have implemented strict Consistency in order to maintain the consistency in the database and the database states are synchronized. The consistency technique is implemented in both the replicas. Following is the snapshot of the current_state (current_status.txt) database showing that the data is consistent in both the database. This file can be found inside each replica under the DB (DataBase) directory. Similarly, consistency is maintained for history log as well. It is named as history.csv under the same directory path.

current_status.txt for Gateway-1



```
current_status.txt (~/Desktop/DOS/lab3/src/gateway_1/DB) - gedit
Documents
Open
current_status.txt
1492569319.34 2 tempSensor 0
1492569319.4 3 smartOutlet 0
1492569323.65 4 doorSensor 0
1492569326.58 5 smartBulb 1
1492569324.16 6 motionSensor 1 |
```

current_status for Gateway-2

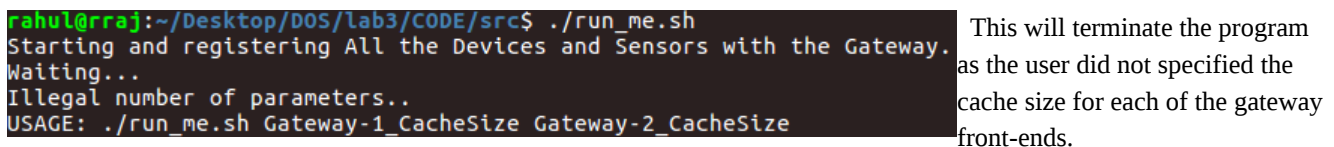


```
current_status.txt (~/Desktop/DOS/lab3/src/gateway_2/DB) - gedit
Documents
Open
current_status.txt
1492569319.34 2 tempSensor 0
1492569319.4 3 smartOutlet 0
1492569323.65 4 doorSensor 0
1492569326.58 5 smartBulb 1
1492569324.16 6 motionSensor 1
```

Both the files have same data and are consistent across different runs of the program. Same is true for history log.

- **CACHE IMPLEMENTATION AND REPLACEMENT POLICY**

We have implemented cache in both the Gateway Front-end. The cache size (N) is configurable for both the front-ends and can be specified by giving it as an argument to run_me.sh.



```
raahul@rraj:~/Desktop/DOS/lab3/CODE/src$ ./run_me.sh
Starting and registering All the Devices and Sensors with the Gateway.
Waiting...
Illegal number of parameters..
USAGE: ./run_me.sh Gateway-1_CacheSize Gateway-2_CacheSize
```

This will terminate the program as the user did not specified the cache size for each of the gateway front-ends.

The cache in the front end will help us to reduce the number of time the database is accessed as well as the number of times the connection is established with the sensors/devices in order to retrieve data.

The replacement policy we have implemented when cache is full is Least Recently Used (LRU). We are storing the access time stamp in the cache and by looking at the time stamp we can easily figure out which entry was least recent. Next, that entry can be thrown out from the cache and new entry can be cached.

We are using write-through policy, so whenever there is a write to the cache, the database is also written and there is no inconsistency between the cache and the database. One of the advantage of using write-through cache is that there will be no data loss if the replica crashes as we are writing to the database at the time of the cache write itself, in the case of write-back cache there was a chance of data loss or inconsistency because there is a possibility that the updated data might be in the cache but the data was not written to the disk and in the meanwhile the replica crashes so the data in the cache is also lost without updating the database and hence inconsistency or data loss will be there in this case.

Following example shows working of our cache and cache replacement policy.

```
rahul@raj:~/Desktop/DOS/lab3/CODE/src$ ./run_me.sh 2 3
Starting and registering All the Devices and Sensors with the Gateway.
Waiting...
tempSensor connected to 8001
Registering tempSensor with gateway 1 with node ID 2
Adding the cache in Gateway-1.
Sending to the backend : node ID is 2 , Sensor/Device name is tempSensor , current state is 0
smartOutletDevice connected to 8001
Registering smartOutlet with gateway 1 with node ID 3
Adding the cache in Gateway-1.
Sending to the backend : node ID is 3 , Sensor/Device name is smartOutlet , current state is 0
rahul@raj:~/Desktop/DOS/lab3/CODE/src$ doorSensor connected to 8001
Registering doorSensor with gateway 1 with node ID 4
Cache full in GATEWAY-1 Evicting the cache using LRU and caching the current item
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
smartBulbDevice connected to 9001
Registering smartBulb with gateway 2 with node ID 5
Adding the cache in Gateway-2.
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 0
motionSensor connected to 9001
Registering motionSensor with gateway 2 with node ID 6
Adding the cache in Gateway-2.
Sending to the backend : node ID is 6 , Sensor/Device name is motionSensor , current state is 0
```

In the above example the cache Size of Gateway-1 is 2 and that of Gateway-2 is 3. No. of devices/sensors registered to Gateway-1 is 3 and Gateway-2 is 2. So, when the 3rd Device/sensor tries to register to the Gateway-1, it sees that cache is full and LRU needs to be used to make space in the cache. Following are few more snapshots from the same run which shows how caches are updated and how the cached data is used and finally what is the cache miss and hit number.

```
This is an Gateway-2 event: QUERY BULB STATE
Calling smartBulb from Gateway Front End. Gateway current clock value is 3
Using the cached data.
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 0
-----
This is an User Event: BULB ON
Calling smartBulb from User Process. User current clock value is 1
set_state of bulb is being called.
Updating the cache in Gateway-2.
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 1
-----
This is an Gateway-2 event: QUERY BULB STATE
Calling smartBulb from Gateway Front End. Gateway current clock value is 7
Using the cached data.
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 1
```

QUERY BULB STATE which is a Gateway-2 event as smart Bulb device is registered with Gateway-2; it uses the cache data initially as it knows the state has not been changed after the device has registered. After the state of the bulb is changed we need to update the cache as shown in the red color in the middle. Finally, when we try to QUERY BULB STATE again it will use the cache data and return us the correct result.

```
This is an Gateway-1 event: QUERY DOOR STATE
Calling QdoorSensor from Gateway Front End. Gateway current clock value is 10
caching the new data..!!
Door sensor state returned is 0
Updating the cache in Gateway-1.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensorQ , current state is 0
```

Whenever there is a new data which we need to cache provided there is space in the cache ,it will do so and give a message “caching the new data..!!”. And by doing so it updated its own cache so whenever there will be a query in the future it can use this cached data. Following figure will show cache hit and miss number for this case.

```
Total time taken to complete the GATEWAY-2 Program is 11.0175161362
Total cache hits are 9 and miss are 0
Total time taken to complete the GATEWAY-1 program is 11.1935489178
Total cache hits are 11 and miss are 1
```

The figure shows cache hit and cache miss rate as well as total time to complete the program. Cache miss are less because we have not considered the cold miss in our case. We can easily consider that and the cache miss will increase in that case.

Lets run the same event sequence with no caching, this can be easily done by passing 0 as the parameter to run_me.sh

```
This is an Gateway-2 event: QUERY BULB STATE
Calling smartBulb from Gateway Front End. Gateway current clock value is 3
Will Proceed Without Caching as cacheSize is 0
Bulb device state returned is 0
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 0
-----
This is an User Event: BULB ON
Calling smartBulb from User Process. User current clock value is 1
set_state of bulb is being called.
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 1
-----
This is an Gateway-2 event: QUERY BULB STATE
Calling smartBulb from Gateway Front End. Gateway current clock value is 10
Will Proceed Without Caching as cacheSize is 0
Bulb device state returned is 1
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 1
```

As cache size is 0, none of the data will be cached and we will see “will proceed without caching as cacheSize is 0” prompt on the console. Following is the total run time as well as the cache hit and miss numbers.

```
Total time taken to complete the GATEWAY-2 Program is 11.1273779869
Total cache hits are 0 and miss are 0
Total time taken to complete the GATEWAY-1 program is 11.181483984
Total cache hits are 0 and miss are 0
```

As the number of events in the sequence is very less there is only a slight improvement in the run time of the program.

There will be much bigger improvements if we run the program for large number of events. The Above program was for the following event log.

```
START
GATEWAY:QUERY BULB STATE
USER:BULB ON
GATEWAY:QUERY BULB STATE
USER:DOOR OPEN PS
GATEWAY:QUERY DOOR STATE
USER:DOOR OPEN PS
GATEWAY:QUERY DOOR STATE
USER:MOTION ACTIVE
USER:MOTION INACTIVE
USER:DOOR OPEN PS
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY BULB STATE
STOP
```

The complete output of the code is as below.

1.

```
sh@sh:~$ ./run_me.sh 0 0
Starting and registering All the Devices and Sensors with the Gateway.
Waiting...
tempSensor connected to 8001
Registering tempSensor with gateway 1 with node ID 2
Will Proceed Without Caching as cacheSize is 0
Sending to the backend : node ID is 2 , Sensor/Device name is tempSensor , current state is 0
smartOutlet connected to 8001
Registering smartOutlet with gateway 1 with node ID 3
Sending to the backend : node ID is 3 , Sensor/Device name is smartOutlet , current state is 0
sh@sh:~$ ./run_me.sh 0001/sr5 doorSensor connected to 8001
Registering doorSensor with gateway 1 with node ID 4
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
smartBulb connected to 8001
Registering smartBulb with gateway 2 with node ID 5
Will Proceed Without Caching as cacheSize is 0
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 0
motionSensor connected to 8001
Registering motionSensor with gateway 2 with node ID 6
Sending to the backend : node ID is 6 , Sensor/Device name is motionSensor , current state is 0

=====
All device registered to gateway - 2 are!!!
Node Name and Node Type are: {'motionSensor': 'Sensor', 'smartBulb': 'Device'}
Node Name and ID are: {'motionSensor': 6, 'smartBulb': 5}
=====
All device registered to gateway - 1 are!!!
Node Name and Node Type are: {'doorSensor': 'Sensor', 'tempSensor': 'Sensor', 'smartOutlet': 'Device'}
Node Name and ID are: {'doorSensor': 4, 'tempSensor': 2, 'smartOutlet': 3}
=====
Started parsing the Event log, GATEWAY.
-----
Started parsing the Event log, GATEWAY.
```

2.

```
This is an Gateway-2 event: QUERY BULB STATE
Calling smartBulb from Gateway Front End. Gateway current clock value is 3
Will Proceed Without Caching as cacheSize is 0
Bulb device state returned is 0
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 0
-----
This is an User Event: BULB ON
Calling smartBulb from User Process. User current clock value is 1
set_state of bulb is being called.
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 1
-----
This is an Gateway-2 event: QUERY BULB STATE
Calling smartBulb from Gateway Front End. Gateway current clock value is 10
Will Proceed Without Caching as cacheSize is 0
Bulb device state returned is 1
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 1
-----
This is an User Event: DOOR OPEN PS
Calling doorSensor from User Process. User current clock value is 1
set_state of door is being called from user process.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 1
Closing Door Automatically.
Calling doorSensor from User Process. User current clock value is 2
set_state of door is being called from user process.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
-----
This is an Gateway-1 event: QUERY DOOR STATE
Calling doorSensor from Gateway Front End. Gateway current clock value is 10
Will Proceed Without Caching as cacheSize is 0
Door sensor state returned is 0
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
-----
This is an User Event: DOOR OPEN PS
Calling doorSensor from User Process. User current clock value is 3
set_state of door is being called from user process.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 1
Closing Door Automatically.
Calling doorSensor from User Process. User current clock value is 4
set_state of door is being called from user process.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
-----
```

3.

```

This is an User Event: MOTION ACTIVE
Calling motionSensor from User Process. User current clock value is 2
set_state of motionSensor is being called from user process.
Sending to the backend : node ID is 6 , Sensor/Device name is motionSensor , current state is 1
Turning on Lights.
Calling smartBulb from User Process. User current clock value is 5
set_state of bulb is being called.
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 1
USER ENTERED
-----
This is an User Event: MOTION INACTIVE
Calling motionSensor from User Process. User current clock value is 3
set_state of motionSensor is being called from user process.
Sending to the backend : node ID is 6 , Sensor/Device name is motionSensor , current state is 0
-----
This is an User Event: DOOR OPEN PS
Calling doorSensor from User Process. User current clock value is 6
set_state of door is being called from user process.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 1
Closing Door Automatically.
Calling doorSensor from User Process. User current clock value is 7
set_state of door is being called from user process.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
USER EXIT
-----
This is an Gateway-1 event: QUERY DOOR STATE
Calling QdoorSensor from Gateway Front End. Gateway current clock value is 30
Will Proceed Without Caching as cacheSize is 0
Door sensor state returned is 0
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensorQ , current state is 0
-----
This is an Gateway-2 event: QUERY BULB STATE
Calling smartBulb from Gateway Front End. Gateway current clock value is 18
Will Proceed Without Caching as cacheSize is 0
Bulb device state returned is 1
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 1
-----
Reached end of log file. Finished Parsing.
-----
Reached end of log file. Finished Parsing.

```

Lets try running the program with larger number of event. It will help us clearly demonstrate the effect of caching as well as when LRU policy comes into action. Following is the event sequence for which we will test the system.

START

GATEWAY:QUERY OUTLET STATE Running with 0 cache size.

```

GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY BULB STATE
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY TEMPERATURE
GATEWAY:BULB ON
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:OUTLET ON
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY OUTLET STATE
GATEWAY:DOOR CLOSE PS
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY OUTLET STATE
STOP

```

```

Total time taken to complete the GATEWAY-2 Program is 11.150799036
Total cache hits are 0 and miss are 0
Total time taken to complete the GATEWAY-1 program is 11.644589901
Total cache hits are 0 and miss are 0

```

Running with cache size of 1 in both the Gateways.

```

GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY BULB STATE
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY TEMPERATURE
GATEWAY:BULB ON
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:OUTLET ON
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY OUTLET STATE
GATEWAY:DOOR CLOSE PS
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY OUTLET STATE
STOP

```

```

Total time taken to complete the GATEWAY-2 Program is 10.734459877
Total cache hits are 10 and miss are 1
Total time taken to complete the GATEWAY-1 program is 11.3740828037
Total cache hits are 9 and miss are 11

```

Running with cache size of 6 in both the Gateways.

```

GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY MOTION STATE
GATEWAY:QUERY OUTLET STATE
GATEWAY:DOOR CLOSE PS
GATEWAY:QUERY DOOR STATE
GATEWAY:QUERY OUTLET STATE
STOP

```

```

Total time taken to complete the GATEWAY-2 Program is 10.695797205
Total cache hits are 11 and miss are 0
Total time taken to complete the GATEWAY-1 program is 11.0526080132
Total cache hits are 15 and miss are 0

```

In the case of Cache Size =1 for both the gateways there was greater number of misses so we can inspect this case and look into working of LRU.

```

This is an Gateway-1 event: QUERY OUTLET STATE
Calling smartOutlet from Gateway Front End. Gateway current clock value is 4
caching the new data.!!!
Outlet device state returned is 0
Updating the cache in Gateway-1.
Sending to the backend : node ID is 3 , Sensor/Device name is smartOutlet , current state is 0
-----
This is an Gateway-1 event: QUERY DOOR STATE
Calling QdoorSensor from Gateway Front End. Gateway current clock value is 8
Cache full in GATEWAY-1 Evicting the cache using LRU and caching the current item
Door sensor state returned is 0
Cache full in GATEWAY-1 Evicting the cache using LRU and caching the current item
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensorQ , current state is 0

```


Gateway-1 has both door as well as smart outlet connected to it. But the cache size is only 1 so it can cache 1 record at a time. When we query outlet state the cache is updated with that but again when we try to query door state the cache is full and Gateway-1 has to run LRU to evict the older entry and cache the new entry, which is what exactly is done by the program.

• FAULT TOLERANCE AND FAILURE RECOVERY

We have implemented fault detection using the probing technique in which both the replicas of the Gateway probe each other by sending heartbeat message to each other. Whenever one of the replica doesn't acknowledge to the heartbeat message the other replica declare it as dead and initiate the fault recovery mechanism. In the case of Failure recovery all the devices connected to the gateway which crashed has to reconfigure itself and register themselves with the healthy gateway.

Lets look into the same example as before and we will crash the Gateway-1, 3 seconds into the program. Following screen shot is from the run_me.sh file which will initiate the crashing of Gateway-1.

```
##Starting the First Replica of Gateway
python gateway_1/Frontend_1.py $1 &
python gateway_1/Backend_1.py &
##Starting the Second Replica of Gateway
python gateway_2/Frontend_2.py $2 &
python gateway_2/Backend_2.py &
##Starting all the Sensors and Devices
python tempSensor/tempSensor.py &
/bin/sleep 0.1
python smartOutletDevice/smartOutlet.py &
/bin/sleep 0.1
python doorSensor/doorSensor.py &
/bin/sleep 0.1
python smartBulbDevice/smartBulb.py &
/bin/sleep 0.1
python motionSensor/motionSensor.py &
python user/user.py &
/bin/sleep 3
pkill -f Frontend_1.py
```

The last line of this file will kill the Gateway-1 front-end.

Following few snapshot will show that the gateway-1 has crashed and gateway-2 takes over the responsibility of the other gateway also.

```
This is an Gateway-2 event: QUERY BULB STATE
Calling smartBulb from Gateway Front End. Gateway current clock value is 10
Using the cached data.
Sending to the backend : node ID is 5 , Sensor/Device name is smartBulb , current state is 1
Devices connected to Failed replica are {'doorSensor': 4, 'tempSensor': 2, 'smartOutlet': 3}
GATEWAY-1 FAILED. INITIATE FAILURE RECOVERY.
```

At some point of time GATEWAY-1 failed as shown above. It also shows the devices which are connected to the failed replica. Now whenever the door sensor is queried it should be done from Gateway-2 and not Gateway-1.

```
This is an User Event: DOOR OPEN PS
Calling doorSensor from User Process. User current clock value is 2
set_state of door is being called from user process.
Closing Door Automatically.
Calling doorSensor from User Process. User current clock value is 3
set_state of door is being called from user process.
-----
This is an Gateway-2 event: QUERY DOOR STATE
Calling QdoorSensor from Gateway Front End. Gateway current clock value is 12
Cache full in GATEWAY-2 Evicting the cache using LRU and caching the current item
Door sensor state returned is 0
Updating the cache in Gateway-2.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
Reconfiguring the Device.
Updating the cache in Gateway-2.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
-----
This is an User Event: DOOR OPEN PS
Calling doorSensor from User Process. User current clock value is 4
set_state of door is being called from user process.
Updating the cache in Gateway-2.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 1
Closing Door Automatically.
Calling doorSensor from User Process. User current clock value is 5
set_state of door is being called from user process.
Updating the cache in Gateway-2.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
-----
This is an Gateway-2 event: QUERY DOOR STATE
Calling QdoorSensor from Gateway Front End. Gateway current clock value is 24
Using the cached data.
Sending to the backend : node ID is 4 , Sensor/Device name is doorSensor , current state is 0
-----
This is an User Event: MOTION ACTIVE
Calling motionSensor from User Process. User current clock value is 6
set_state of motionSensor is being called from user process.
Cache full in GATEWAY-2 Evicting the cache using LRU and caching the current item
Sending to the backend : node ID is 6 , Sensor/Device name is motionSensor , current state is 1
```

QUERY DOOR STATE first reconfigured the device and then it updated the Gateway-2 as shown in the snapshot. Any further query to door sensor will be done via Gateway-2. Similarly, for all the other devices/sensors connected to failed replica.

Total time taken to complete the GATEWAY-2 increased as well as cache miss also increased for the same configuration because now it has to performs the tasks of Gateway-1 also.

```
Total time taken to complete the GATEWAY-2 Program is 16.5312199593
Total cache hits are 13 and miss are 6
```

• PAXOS IMPLEMENTATION RESULTS

We considered six gateway replicas where three of them are serving the temperature sensor, smart bulb device and motion sensor separately and other three are just acts as backup gateways (we can add more sensors and devices and connect them to those replicas, but for simplicity of the output we have considered on three sensor/device nodes). Just to demonstrate paxos, we kept the sensor/device functionality simple. They just perform state change events like request bulb on, change motion sensor to inactive, set temperature etc or ask their respective gateway for their older states. We can considered gateway-6 to be the leader.

Bellow figure shows the file structure for paxos implementation:

```
olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAXO$ ls
gateway1.py gateway3.py gateway5.py Motionsensor.py smartBulb.py
gateway2.py gateway4.py Gateway6.py run_me.sh Tempensor.py
olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAXO$
```

Run the paxos demonstration by running the command “./run_me.sh” from the terminal. The output generated that implements paxos is as shown below:

```
olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAXO$ ./run_me.sh
Waiting...
Waiting to connect
Write request recieved 1:54 - Tempsensor
Starting consensus agreement
Starting round 1
-----
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 1
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 2
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 3
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 4
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 5
All gateways agreed on consensus (node ID: state) 1:54 - Tempsensor
olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAXO$ Gateway 1: Reading data

Returned read Temp: 1:54 - Tempsensor
-----
Waiting to connect

Write request recieved 2:ACTIVE - Motion_sensor
Starting consensus agreement
Starting round 2
-----
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 1
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 2
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 3
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 4
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 5
All gateways agreed on consensus (node ID: state) 2:ACTIVE - Motion_sensor
```

Here you can see that initially the leader is waiting for any paxos request. Once the temperature sensor requests its serving gateway to set the temperature, a request is made by the serving gateway to the leader that a request has come, start the

consensus agreement. Leader then initiates and connects to all other for acceptance and once all nodes makes a promise to accept the consensus, it sends the event to all replicas to log it in their database. When temperature sensor queries from serving gateway (reading event) then you can clearly see that the service is served by the gateway itself and it doesn't start the paxos.

Therefore, paxos is only initiated when there is any request for some kind of write in one of the gateway's database. Similarly, motion sensor and smart bulb also makes requests and paxos gets initiated only during writes. Complete output shown below proves the statement described. The "Accepted write request= 3" represent sequence number.

```
olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAXOS$ ./run_me.sh
Waiting...
Waiting to connect
Write request recieved 1:54 - Tempsensor
Starting consensus agreement
Starting round 1
.....
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 1
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 2
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 3
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 4
Accepted write request: 1 Consensus on node:state : 1:54 - Tempsensor my id: 5
All gateways agreed on consensus (node ID: state) 1:54 - Tempsensor
olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAXOS$ Gateway 1: Reading data

Returned read Temp: 1:54 - Tempsensor
.....
Waiting to connect

Write request recieved 2:ACTIVE - Motion_sensor
Starting consensus agreement
Starting round 2
.....
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 1
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 2
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 3
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 4
Accepted write request: 2 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 5
All gateways agreed on consensus (node ID: state) 2:ACTIVE - Motion_sensor
Gateway 2: Reading data

Returned read Temp: 2:ACTIVE - Motion_sensor
.....
Waiting to connect

Write request recieved 3:ON - SamrtBulb
Starting consensus agreement
Starting round 3
.....
Accepted write request: 3 Consensus on node:state : 3:ON - SamrtBulb my id: 1
Accepted write request: 3 Consensus on node:state : 3:ON - SamrtBulb my id: 2
Accepted write request: 3 Consensus on node:state : 3:ON - SamrtBulb my id: 3
Accepted write request: 3 Consensus on node:state : 3:ON - SamrtBulb my id: 4
Accepted write request: 3 Consensus on node:state : 3:ON - SamrtBulb my id: 5
All gateways agreed on consensus (node ID: state) 3:ON - SamrtBulb
Gateway 3: Reading data

Returned read state: 3:ON - SamrtBulb
.....
```

```
Gateway 3: Reading data

Returned read state: 3:ON - SamrtBulb
.....
Write request recieved 1:42 - Tempsensor
Starting consensus agreement
Starting round 4
.....
Accepted write request: 4 Consensus on node:state : 1:42 - Tempsensor my id: 1
Accepted write request: 4 Consensus on node:state : 1:42 - Tempsensor my id: 2
Accepted write request: 4 Consensus on node:state : 1:42 - Tempsensor my id: 3
Accepted write request: 4 Consensus on node:state : 1:42 - Tempsensor my id: 4
Accepted write request: 4 Consensus on node:state : 1:42 - Tempsensor my id: 5
All gateways agreed on consensus (node ID: state) 1:42 - Tempsensor
Write request recieved 2:ACTIVE - Motion_sensor
Starting consensus agreement
Starting round 5
.....
Accepted write request: 5 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 1
Accepted write request: 5 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 2
Accepted write request: 5 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 3
Accepted write request: 5 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 4
Accepted write request: 5 Consensus on node:state : 2:ACTIVE - Motion_sensor my id: 5
All gateways agreed on consensus (node ID: state) 2:ACTIVE - Motion_sensor
Write request recieved 3:OFF - SamrtBulb
Starting consensus agreement
Starting round 6
.....
Accepted write request: 6 Consensus on node:state : 3:OFF - SamrtBulb my id: 1
Accepted write request: 6 Consensus on node:state : 3:OFF - SamrtBulb my id: 2
Accepted write request: 6 Consensus on node:state : 3:OFF - SamrtBulb my id: 3
Accepted write request: 6 Consensus on node:state : 3:OFF - SamrtBulb my id: 4
Accepted write request: 6 Consensus on node:state : 3:OFF - SamrtBulb my id: 5
All gateways agreed on consensus (node ID: state) 3:OFF - SamrtBulb

olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAXOS$
```

There can be situation when the gateways are not ready to accept the request from the current leader. The index N, specified by the leader may have been occupied in one of the replica then the leader keeps trying in multiple rounds and finally gets the consensus when majority of them are ready.


```

Waiting to connect

Write request recieved 1:45 - Tempsensor
Starting consensus agreement
Starting round 1
-----
Accepted write request: 1 Consensus on node:state : 1:45 - Tempsensor my id: 1
Round failed !! Starting another round
Accepted write request: 1 Consensus on node:state : 1:45 - Tempsensor my id: 3
Accepted write request: 1 Consensus on node:state : 1:45 - Tempsensor my id: 4
Accepted write request: 1 Consensus on node:state : 1:45 - Tempsensor my id: 5
Starting round 2
Accepted write request: 2 Consensus on node:state : 1:45 - Tempsensor my id: 1
Accepted write request: 2 Consensus on node:state : 1:45 - Tempsensor my id: 2
Accepted write request: 2 Consensus on node:state : 1:45 - Tempsensor my id: 3
Accepted write request: 2 Consensus on node:state : 1:45 - Tempsensor my id: 4
Accepted write request: 2 Consensus on node:state : 1:45 - Tempsensor my id: 5
All gateways agreed on consensus (node ID: state) 1:45 - Tempsensor

```

Clearly, we can see from above output that gateway 2 did not accept the consensus, as the index N was pre-occupied so the leader runs one more round to that everyone accepts the consensus and successfully log the temperature into their database.

The event logs generated by all the gateway replicas are consistent. Hence consistency is achieved using paxos.
Event logs generated by the gateways are as follows:

```

olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAX0$ cat event_log1.txt
Seq No. 1 Device:State -> 1:54 - Tempsensor
Seq No. 2 Device:State -> 2:ACTIVE - Motion_sensor
Seq No. 3 Device:State -> 3:ON - SamrtBulb
Seq No. 4 Device:State -> 1:42 - Tempsensor
Seq No. 5 Device:State -> 2:ACTIVE - Motion_sensor
Seq No. 6 Device:State -> 3:OFF - SamrtBulb

olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAX0$ cat event_log2.txt
Seq No. 1 Device:State -> 1:54 - Tempsensor
Seq No. 2 Device:State -> 2:ACTIVE - Motion_sensor
Seq No. 3 Device:State -> 3:ON - SamrtBulb
Seq No. 4 Device:State -> 1:42 - Tempsensor
Seq No. 5 Device:State -> 2:ACTIVE - Motion_sensor
Seq No. 6 Device:State -> 3:OFF - SamrtBulb
olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAX0$

olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAX0$ cat event_log5.txt
Seq No. 1 Device:State -> 1:54 - Tempsensor
Seq No. 2 Device:State -> 2:ACTIVE - Motion_sensor
Seq No. 3 Device:State -> 3:ON - SamrtBulb
Seq No. 4 Device:State -> 1:42 - Tempsensor
Seq No. 5 Device:State -> 2:ACTIVE - Motion_sensor
Seq No. 6 Device:State -> 3:OFF - SamrtBulb
olenka@olenka-Inspiron-7559:~/Desktop/DOS/PAX0$

```