CMPSCI 677 Operating Systems **Spring 2015**

Lab 3: May 8

Lecturer: Prashant Shenoy *Developers: Patrick Pegus II, Amee Trivedi*

System Overview

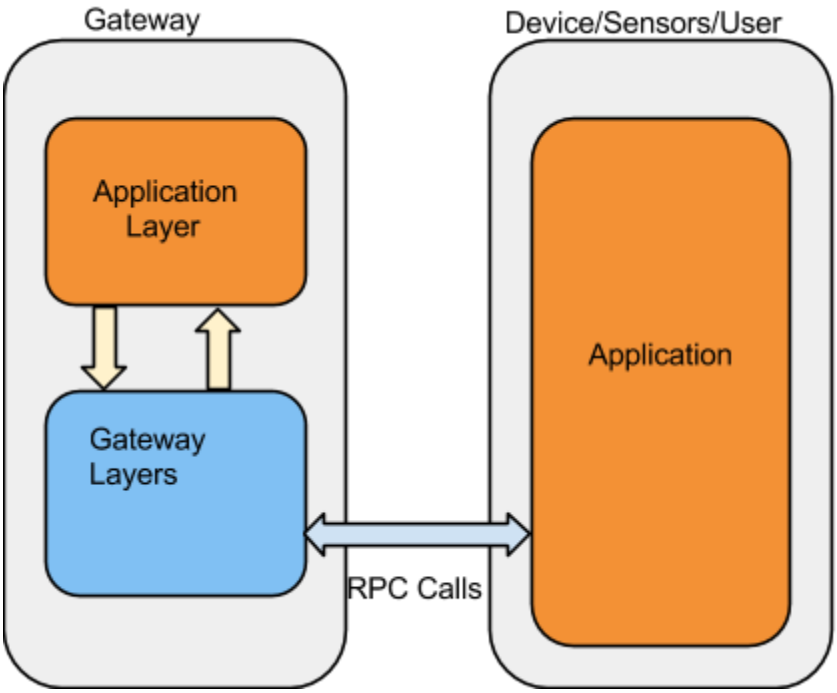
To build IoT fault tolerant system, supporting :

- 2 Gateway replicas
- 3 Sensors comprising of a motion sensor, temperature sensor and door sensor
- 2 Devices : Smart Bulb and Smart Outlet

The system should do load balancing across the gateway replicas and be fault tolerant against gateway replica failures. The system should also implement cache and database consistency.

System Design

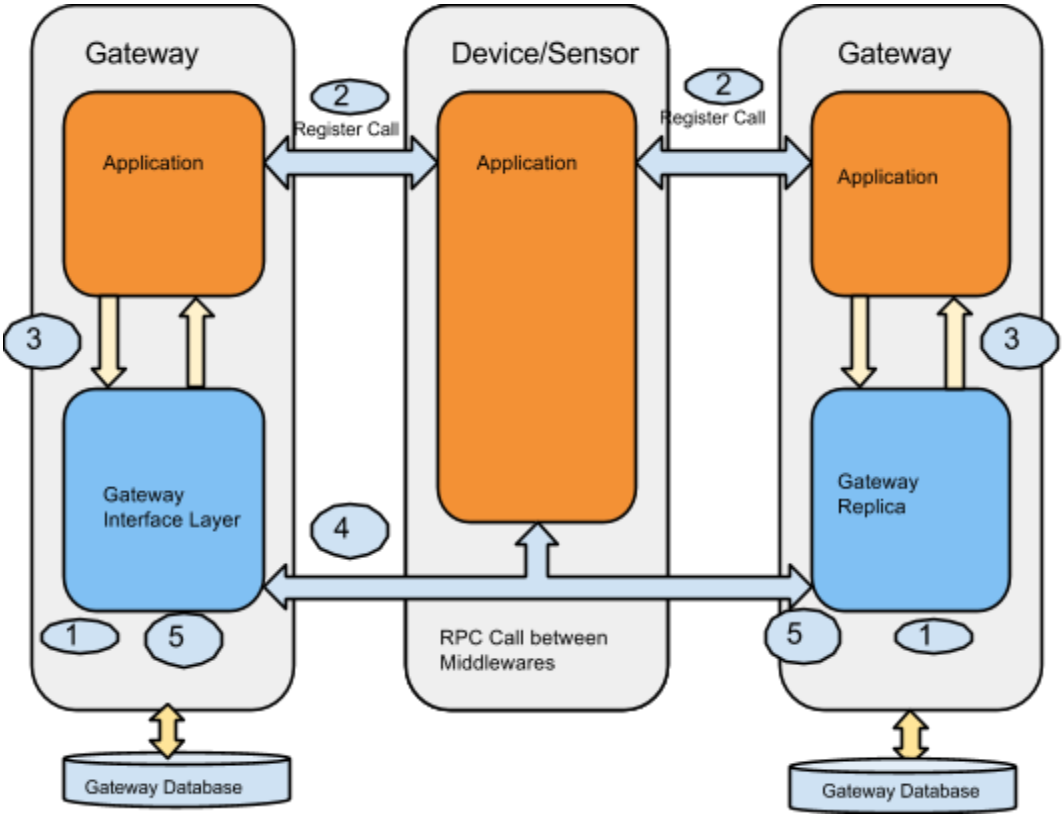
Programming language used is Go Lang. Go Gophers! Go!



CMPSCI 677 Operating Systems **Spring 2015**

Lab 3: May 8

Lecturer: Prashant Shenoy *Developers: Patrick Pegus II, Amee Trivedi*



Design Description

All the features supported by sensors, devices and gateways are carry forwarded from the previous lab. So, features such as motion detection, temperature increase, etc are not describe here, only new features of Lab 3 are stated.

System Start and Device Registration Design

1. The databases come up first
2. The gateways come up next and elect a leader between them
3. The devices and sensors multicast the registration request to both the gateways
4. The leader gateway sends a reply to the registration request. The reply comprises of :
    a. device/sensor id
    b. Address of gateway to send further event updates to(this is for load balancing between replicas)
    c. Port of the the gateway selected in pt. b above
5. Next, the devices start communicating to their assigned gateway replicas

Fault tolerance

1. The gateways keep sending "IAMALIVE" messages to each other to ensure that they are up
2. Incase, a gateway replica crashes/dies (say replica 1) then the other gateway (say replica 2) does not receive "IAMALIVE" token and then replica 2 will send a message to all devices stating that gateway replica 2 is the only gateway in the system so all devices should send updates/event notifications to replica 2
3. When gateway replica 1 comes up then it will check with replica 2 the last event handled by matching the timestamps
4. All events from gateway replica 2 will be replicated in the database of replica 1 and the databases will be made consistent
5. Again load balancing will be done and a message will be sent to some registered devices (selected based on the load balancing algorithm) to update the address and port of gateway replica they should send notifications to

Caching

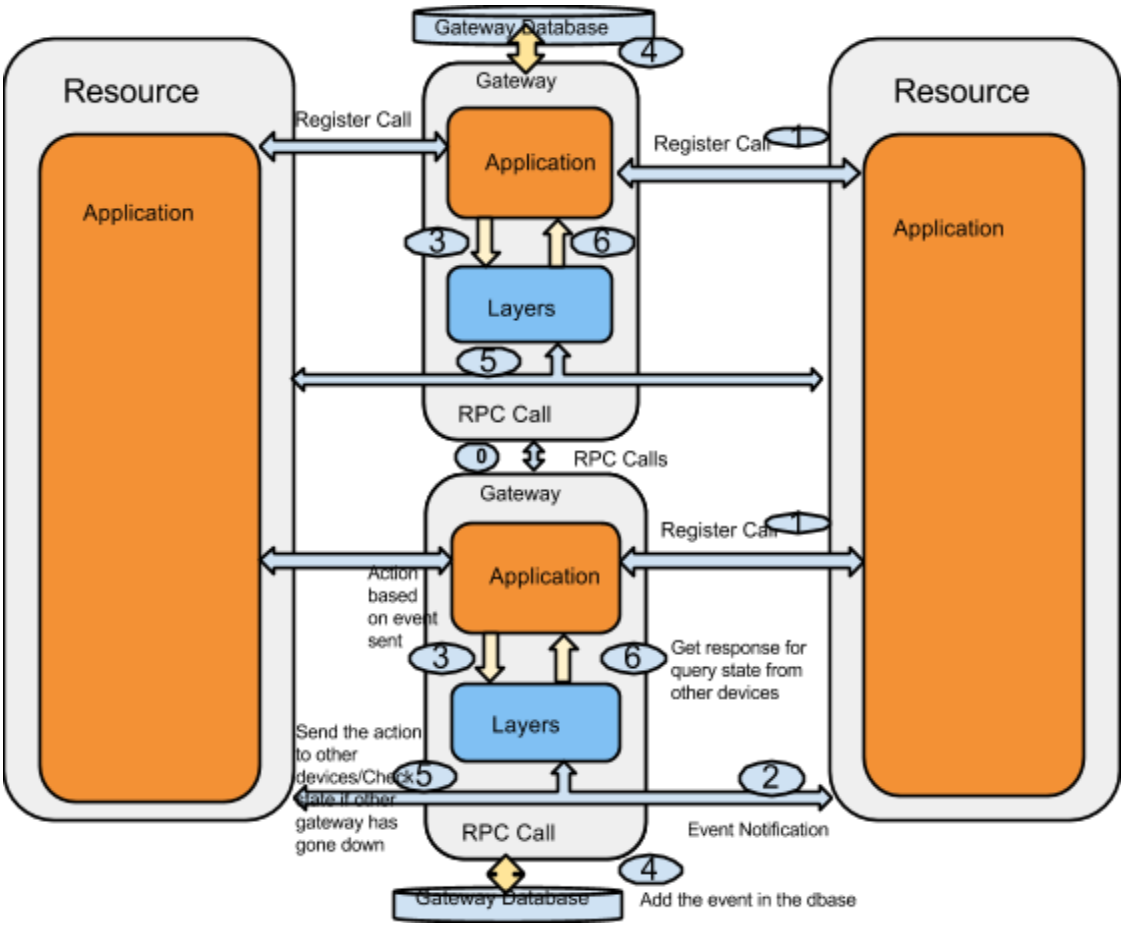
1. The cache is a write through cache
2. Caching algorithm used is Least Recently Used.
3. The cache implements 2 main functionalities:
    a. Add a new record of StateInfo. In this function, it check if the record already exists, if it does then it updates the record and the referenced timestamp value
    b. Fetch an existing record from the cache. Given the deviceid fetch the stateinfo value from the cache, if we get a nil it means the value does not exist and we need to fetch it from the database.

CMPSCI 677 Operating Systems **Spring 2015**

Lab 3: May 8

Lecturer: Prashant Shenoy *Developers: Patrick Pegus II, Amee Trivedi*

Figure with 2 gateways and 2 devices



System Assumptions

1. All communication channels are up and running always. If the TCP Dial service fails, it is assumed that the device/sensor is down
2. The Gateway is the central entity of system. All intelligence is in the gateway
3. Database always comes up first, followed by Gateway and then, the rest devices.

4. The Temperature increments and decrements are always measured by unit change. i.e. Increment is +1 degree and Decrement is -1 degree
5. All device clocks are in sync
6. The temperature sensor polling is done by the gateway every 60 seconds

System Design Decisions

Leader Selection between Gateway Replicas

The leader selection algorithm is "Bully Algorithm". The replicas select a leader who registers the devices and does load sharing.

Consistency Guarantee

If a gateway goes down and then comes up, to ensure consistency the replica gateway that comes up sinks data from the other gateway that was up in the network

Gateway Layered Approach [The power of layers and interfaces]

The gateway is implemented using a layered approach. So, it is easy to add layers and separate the functionality. All incoming requests do not get forwarded to the above layer from a below layer. This helps forward certain requests to the leader while the rest gateway replicas need not get those notifications.

Timestamp

All events are timestamped to decide the order of execution. Unix time is used to time-stamp events.

RPC calls

The calls between middlewares of resources are RPC based calls.

System Dependencies

None

Code Organization

All the structs, type and interface declarations are in api.go
File database.go defines the structs and interfaces needed

The rest resources are in the <resource-name> dir with main.go file, which is responsible to start the resource instance and the <resource-name>.go file that has all the interface implementations and it also starts its middleware.

Known Issue

None

Measurements

Cache Inconsistency:
The architecture implement requires the replica that has failed over and restarted to sync the database from the new leader. There is a window of maximum 2 secs when the “IAMALIVE” messages are send and the prior failed replica that has come up has been identified by the new leader. Once the database are in sync within 2 secs of the failed replica the load balancing and other features are resumed.

Fault tolerance:
Take-over measurements ranged from 100msec - 1500 msec, reason is IAMALIVE broadcast is every 2 secs.
If the devices go down then the recovery time for the other replica to take over is equal to the time elapsed since the last IAMALIVE was received. In our system, IAMALIVE is sent every 2 secs. So, there is a chance that the leader goes down and in that window when new leader is not selected a device tries to register and is unable to register. But, if the IAMALIVE message broadcast time is reduced from 2secs to 150ms-300ms then the window can be reduced further.

Test Logs:
All test logs can found in the \tests* files

Paxos Implementation for Lab 3 IoT Design
The 2 main things to keep in mind for Paxos are satisfaction of the attributes Safety and Liveness.
We can implement Paxos for our architecture by *implementing a log replication model* as explained below.

There are 3 phases in the implementation:

1. Leader Election
2. Log replication
3. Updation of log and notification

Assumptions:

1. No messages are lost
2. No messages are corrupted in transit
3. Clocks are synchronized
4. All logs when consistent and synchronized result in the same set of events in the same order
5. A node can stand in election only if its log is in a consistent state and reflects the latest transactions.

Phase 1: Leader Election

A leader election algorithm such as bully algorithm selects the current gateway leader in the system
All registration requests are directed to the leader.
During the registration request the leader assigns device id and the replica to talk to do load balancing.
All incoming from the sensors are also directed to the leader
A heartbeat is sent to all other replicas every 300msec to check if the Leader is alive. If the leader is not alive then the server to find that leader is dead starts a new election.

Phase 2: Log replication

The correctness of the entire Paxos depends on the consistency of database or event log. Each of the k gateway replicas has a database associated with it where it keeps logging the events from devices/sensors and other replicas.

When an event comes to the leader gateway,

1. It adds the event in the database but marks it as uncommitted
2. Broadcast the event message to all the replica gateways (termed as followers in Paxos)
3. Leader waits for majority of replicas to reply stating they have appended the entry in their logs and the outcome of event is x.
4. Now, the leader commits when it sees that it has gathered a majority of x votes out of K replicas else it tries again until it gets a majority
5. Eventually, a commit will be made by the leader and then the server sends a notification message to the followers (other gateway replicas) that the entry is committed

Phase 3 : Safety

At the end, the leader sends the final agreed upon reply to the client.

At this point, if in step 3 , any followers in minority did not append the entry or negated the request from leader to append to the logs then, the followers have to append now when the leader gateway executes step 5.

All the state change notifications to the devices can be done directly by the replica without having to route it through the leader. The replica assigned to the device as its load balancing replica sends the subsequent message, which might be issued as an outcome of an event reported by the sensor.

Eg. Motion sensor senses state change from no motion to Motion, triggers the gateway to send a Turn bulb On, state change request to the smartbulb

Motion sensor state change is send to leader
The 3 phases described above are followed and the replicas commit the event and agree on state change event.
Now, the replica assigned for load balancing sends a statechange event to the smartbulb and broadcasts the event to other replicas too.

The reason it is safe to not do paxos for event triggered from replica due to a sensor event is that incase the load balancing replica goes down, then the leader can Query the state (RPC Querystate) of the device and find out if the triggered event was committed or not. If not, it can reissue the event.

This design would work for network partition too, with the rule that a node with the consistent updated log can only stand in election. Consistent updated log is identified by the timestamp and event entry listed in the logs.