Viewstamped Replication

This report describes a simple algorithm for implementing viewstamped replication. VR provides state machine replication in an asynchronous network like the Internet and handles nodes failures. The high level architecture of the implementation is shown in figure 1.
VR code contains logic for maintaining a consensus among majority of replicas for each operation. VR proxy sits between the real application client and handles its request. It share the configuration state such as current view number, current primary with replicas. Figure 2 list the state maintained by replica.

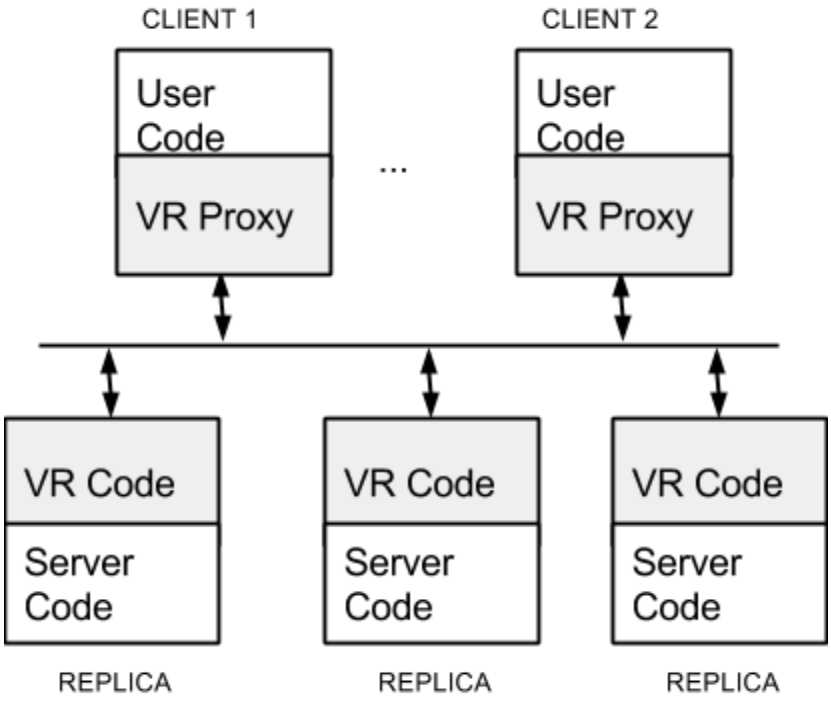

Fig 1: VR Architecture: the figure show configuration when f = 1

The VR protocol can be broadly divided into three modes of operations:

1. **Normal Operation**
   In this mode, the primary replica accepts operation from the VR proxy and queues them for replicating on to atleast f+1 replicas(including itself). Once the operation is logged on majority of replica, it is commited on the primary and is executed against the application service and the response is updated to client. The thrift definitions of the RPCs implemented for this mode is shown in Fig3
2. **View Change**
   If the primary goes down, the replicas will have timeout alerts and will start the view

change algorithm. Each replica will send StartViewChange to all replicas and when a replicas get f such requests, it sends a DoViewChange message to the new primary. The new primary waits for f+1 DoViewChange message and picks the most updated logs from these call. It then sends StartView to all the replica and resume operation in normal mode again. Since log state is transferred in multiple message, this phase has high throughput. Checkpointing the logs will be used to decrease the amount of bytes exchanged.

3. **StateRecovery**
   When a replica is joined back after a failure, it catches up with the latest log through Recovery.

```
ReplicaState {
   List<String> qouroms ;
   int replicaNumber;
   int viewNumber;
   ReplicaStatus status;
   int opNumber;
   List<Log> logs;
   int commitNumber;;
   int checkPoint;
   HashMap<String, ClientRequest > clientTable;
   ArrayDeque<ClientRequest> requestQueue;  //Used only in primary Replica
}
```

Fig 2: VR state at each Replica

```
truct StartViewChangeParameter{
    1: int newViewNumber;
    2: int replicaNumber;
}
struct StartViewChangeResponse{

}
StartViewChangeResponse rpcStartViewChange(1:
StartViewChangeParameter
startViewChangeParameter),
```

```
struct StartViewParameter{
    1: int viewNumber;
    2: map<int,Log> logs;
    3: int checkPoint;
    4: int opNumber;
    5: int commitNumber;
    6: int replicaNumber;
    7: int retryCount =0;
}
struct StartViewResponse{
}
StartViewResponse rpcStartView(1:
StartViewParameter startViewParameter),
```

```
struct DoViewChangeParameter{
    1: int newViewNumber;
    2: map<int,Log> logs;
    3: int oldViewNumber;
    4: int checkPoint;
    5: int opNumber;
    6: int commitNumber;
    7: int replicaNumber;
    8: int retryCount =0;
}
struct DoViewChangeResponse{
}

DoViewChangeResponse rpcDoViewChange(1:
DoViewChangeParameter doViewChangeParameter),
```

Fig 4: View Change RPCs