Formalizing Raft by Modal Logic

Yuya Uezato

ABSTRACT

Informal Note

1 ROUGH IDEA

We borrow the notation from modal logic. At this time, unfortunately, it is not unclear that we also borrow the results from modal logic.

We want to write the following statement (on our transition system defined below):

$$(\text{Leader} \land \text{Commit}(A)) \to \Box(\text{Leader} \to \text{Commit}(A))$$

Modal operators are useful to write properties on transition systems.

2 TRANSITION SYSTEMS

Transition System for Raft Cluster.

Halt node For $(A, queue) \in C$,

$$A \overset{\text{halt}}{\mapsto} (\tilde{A}, queue).$$

Revole node For $(\tilde{A}, queue) \in C$,

$$\tilde{A} \overset{\text{revoke}}{\mapsto} (A, queue).$$

Add user data to leader For $(\hat{L}, queue) \in C$,

$$(A, q) \overset{\text{+user data } x}{\mapsto} (A, queue\, x).$$

Send log to follower For $(\hat{L}, queue) \in C$,

$$(A, q) \overset{\text{append log}}{\mapsto} (A, queue \wr q).$$

(Incorrect; it should be a partial update)

(Unconditioned) Election If there is no leader in C, we choose some one as the leader $(A, que) \to (\hat{A}, que)$.

Ack to User Can we answer users question: what is the value of a key K. I think we need to introduce some tool to divide uncommitted and commited data.

What happens if writing from the leader to followers fails majority. Since we do not assume simultaneous writing, there is no problem??

Transition System for Raft Manager.

$$\begin{tikzcd}
\mathcal{K} \arrow[r, "\text{command}_n"] \arrow[d, "\text{snd}_n"] & \mathcal{K}' \\
C \arrow[r, phantom, "\to \cdots \to"] & C' \arrow[u, "\text{ack}_n"]
\end{tikzcd}$$

Can we put a data before the previous one is acked?

2.1 Our Desirable Property

At most one leader.

$$\hat{A}, \hat{B} \in C \to A = B.$$

Once committed, never will lost.

$$(k, v) \in M \to \circ((k, v) \in M)$$

If we can prove the above on all worlds, it implies $(k, v) \in M \to \circ^n(k, v) \in M$.

$$(K, V) \in \text{Leader} \to \Box^*(\text{LeaderIs} \implies K \in \text{Leader})$$

2.2 Election for One Leader

$$L_1, L_2 \in C \to L_1 = L_2.$$

Counter Example:

$$\begin{array}{c} A : \epsilon \\ B : \epsilon \end{array} \to \begin{array}{c} \hat{A} : \epsilon \\ B : \epsilon \end{array} \to \begin{array}{c} \hat{A} : \epsilon \\ \hat{B} : \epsilon \end{array}$$

How to Remedy:

ここで既にleaderがいるならなれない、 というような全体知識にアクセスする方法はとれない。 とはいえ後発を受け入れようとすると leaderの取り合いになって計算が進まない。 Termを入れても結局ひたすらincrementされると爆発するしなあ。 久しぶりに復帰したノードが構成状況知るためには Termみたいなのが必要になるのかな。Termはやはり必要?? でもcommit長みれば分かりそうなもんだけど Leaderからの情報を受け付けるbufferと、 誰でも用のbufferの2-level bufferが必要で、 後者をoptimizeするとTermになるのかな（stackがcounterのenrichみたいな感じで）

2.3 OK: Existence of Multiple (Local) Leaders

自分のことをleaderだと思いこんでいるノードが複数あるのは問題がない。

2.4 Bad: Broadcasting data from Multiple Leaders

存在するところまでは別に問題ないが、データを配られてackされるとまずい。 例えば $\hat{n}_1 \xrightarrow{x} n_2$ でack来た後にxを外にackして、 一方で$\hat{n}_3$にqueryが来て、xは無いと答えられると不味い。 でもこれってLeaderが複数あることの問題になっていないか? raftlogではreadアクセス時のLeader search問題をどう解決しているか。 nodeがliveしている間はpushがcommitしたことをオンメモリでcacheしている。 ではnodeがdead->liveする時はどうか?

2.5 Ack Timing

fuga

$$(v \in \mathcal{K}, C) \to \Diamond(Leader \in C \to v \in C)$$

2.6 From $\Diamond$ to $\Box$

$$(v \in \mathcal{K}, C) \to \Box(Leader \in C \to v \in C)$$

We need to emphasize that, in order to state this, we need to introduce conditioned election and un/comitted divider.

Raft Take one of the maximal term
MultiPaxos Summing received data

3 CHANGE CONFIGURATION

3.1 Accommodate Immigrant

hoge

3.2 Remove Non-Leader

fuga

3.3 Remove Leader

piyo

4 ON IMPLEMENTATION: HOW WE TRANSFER USER DATA TO THE CURRENT LEADER

We assume a service manager knows all members. First of all, it proposes a new configuration to the raft service.

- The case Raft $\xrightarrow{\text{ack fail}}$ Service, Service re-proposes a new configuration. It was already committed, the cluster do nothing.

On the initial setting, until after committing and receiving its ack, the manger does not forward user data. Thatafter, until after committing and receiving its ack, the manager forward data to *old* or *new*, which??? Intuitively, we should forward user data to the raft cluster by following *old*. What happens if *new* is committed. First of all, until the manager receive the ack, we do not shotdown nodes to be removed. Therefore, each forward is forward to the leader if receiving node knows the actual leader.

We should consider what happens if a receiving node dow not know the actual lader (and known a fake leader). A fake leader cannot succeed to committing a data and thus the system becomes unavailable.

We somewhat need to know an actual working cluster (or randomly hit its). At this time, I (author) think it is impossible (if all or almost the notifications from the service drop due to some kinds of network troubles). One heuristics is that, on each election, the new leader send back itself information to the service manager.