

EXTENDS *DomainModel*, *TLC*

CONSTANT

GST the first synchronous round (all later rounds are synchronous)

--algorithm *SailFish*{

variables

$vs = \{\}$, the vertices of the *DAG*

$es = \{\}$, the edges of the *DAG*

$no_vote = [n \in N \mapsto \{\}];$ *no_vote* messages sent by each node

define {

$LeaderVertex(r) \triangleq \langle Leader(r), r \rangle$

$VerticeQuorums(r) \triangleq$

$\{ VQ \in \text{SUBSET } vs : \\ \wedge \forall v \in VQ : Round(v) = r \\ \wedge \{ Node(v) : v \in VQ \} \in Quorum \}$

}

process (*correctNode* $\in N \setminus B$)

variables *round* = 0; current round

{

l0: while (TRUE)

either with ($v = \langle self, round \rangle$) {

add a new vertex to the *DAG* and go to the next round

$vs := vs \cup \{v\};$

if (*round* > 0)

with ($vq \in VerticeQuorums(round - 1)$) {

from *GST* onwards, each node receives all correct vertices of the previous round:

when $round \geq GST \Rightarrow (N \setminus B) \subseteq \{ Node(v2) : v2 \in vq \};$

$es := es \cup \{ \langle v, pv \rangle : pv \in vq \};$ add the edges

if ($LeaderVertex(round - 1) \notin vq$) send *no_vote* if previous leader vertex not included

$no_vote[self] := no_vote[self] \cup \{ LeaderVertex(round - 1) \}$

} ;

round := *round* + 1

}

or with ($r \in \{r \in R : r > round\}$) {

go to a higher round

when $round < GST;$ from *GST* onwards, correct nodes do not skip rounds

round := *r*

}

}

Next comes our model of *Byzantine* nodes. Because the real protocol disseminates *DAG* vertices using reliable broadcast, *Byzantine* nodes cannot equivocate and cannot deviate much from the protocol (lest their messages be ignored).

```

process ( byzantineNode ∈ B )
  variables round_ = 0 ;
  {
l0: while ( TRUE ) {
    maybe add a vertices to the DAG:
    either with ( v = ⟨self, round_⟩ ) {
      vs := vs ∪ {v} ;
      if ( round_ > 0 )
        with ( vq ∈ VerticeQuorums(round_ − 1) )
          es := es ∪ {⟨v, pv⟩ : pv ∈ vq}
        } or skip ;
      maybe send a no_vote messages:
      if ( round_ > 0 )
        either
          no_vote[self] := no_vote[self] ∪ {LeaderVertice(round_ − 1)}
        or skip ;
      go to the next round:
      round_ := round_ + 1
    }
  }
}

```

Next we define the safety and liveness properties

$$\begin{aligned}
\textit{Committed}(v) &\triangleq \\
&\wedge v \in vs \\
&\wedge \textit{Node}(v) = \textit{Leader}(\textit{Round}(v)) \\
&\wedge \{ \textit{Node}(pv) : pv \in \textit{Parents}(v, es) \} \in \textit{Quorum} \\
&\wedge \vee \textit{Round}(v) = 0 \\
&\quad \vee \textit{LeaderVertice}(\textit{Round}(v) - 1) \in \textit{Children}(v, es) \\
&\quad \vee \exists Q \in \textit{Quorum} : \forall n \in Q : \\
&\quad \quad \textit{LeaderVertice}(\textit{Round}(v) - 1) \in \textit{no_vote}[n]
\end{aligned}$$

$$\begin{aligned}
\textit{Safety} &\triangleq \forall v1, v2 \in vs : \\
&\wedge \textit{Committed}(v1) \\
&\wedge \textit{Committed}(v2) \\
&\wedge \textit{Round}(v1) \leq \textit{Round}(v2) \\
&\Rightarrow \textit{Reachable}(v2, v1, es)
\end{aligned}$$

$$\begin{aligned}
\textit{Liveness} &\triangleq \forall r \in R : \\
&\wedge r \geq \textit{GST} \\
&\wedge \textit{Leader}(r) \notin B \\
&\wedge \forall n \in N \setminus B : \textit{round}[n] > r + 1 \\
&\Rightarrow \textit{Committed}(\textit{LeaderVertice}(r))
\end{aligned}$$

Finally we make a few auxiliary definitions used for model-checking with *TLC*

The round of a node, whether *Byzantine* or not
 $Round_ (n) \triangleq \text{IF } n \in B \text{ THEN } round_ [n] \text{ ELSE } round[n]$

Basic typing invariant:
 $TypeOK \triangleq$
 $\wedge \forall v \in vs : Node(v) \in N \wedge Round(v) \in Nat$
 $\wedge \forall e \in es :$
 $\quad \wedge e = \langle e[1], e[2] \rangle$
 $\quad \wedge \{e[1], e[2]\} \subseteq vs$
 $\quad \wedge Round(e[1]) > Round(e[2])$
 $\wedge \forall n \in N :$
 $\quad \wedge Round_ (n) \in Nat$
 $\quad \wedge no_ vote[n] \subseteq \{ \langle Leader(r), r \rangle : r \in R \}$

Sequentialization constraints, which enforce a particular ordering of the actions. Because of how actions commute, the set of reachable states remains unchanged. This speeds up model-checking a lot.

$SeqConstraints(n) \triangleq$
 wait for all nodes to finish previous rounds:
 $\wedge (Round_ (n) > 0 \Rightarrow \forall n2 \in N : Round_ (n2) \geq Round_ (n))$
 wait for all nodes with lower index to leave the round:
 $\wedge \forall n2 \in N : NodeIndex(n2) < NodeIndex(n) \Rightarrow Round_ (n2) > Round_ (n)$

$SeqNext \triangleq (\exists self \in N \setminus B : SeqConstraints(self) \wedge correctNode(self))$
 $\quad \vee (\exists self \in B : SeqConstraints(self) \wedge byzantineNode(self))$
 $SeqSpec \triangleq Init \wedge \Box [SeqNext]_{vars}$

Example assignment of leaders to rounds:
 $ModLeader(r) \triangleq NodeSeq[(r \% Cardinality(N)) + 1]$

Constraint to stop the model checker:
 $StateConstraint \triangleq$
 $LET Max(S) \triangleq CHOOSE x \in S : \forall y \in S : y \leq x IN$
 $\forall n \in N : Round_ (n) \in 0 \dots (Max(R) + 1)$

Some properties we expect to be violated:

$Falsy1 \triangleq \neg($
 $\quad \wedge Committed(\langle Leader(1), 1 \rangle)$
 $)$

$Falsy2 \triangleq \neg($
 $\quad \wedge Committed(\langle Leader(0), 0 \rangle)$
 $\quad \wedge \neg Committed(\langle Leader(1), 1 \rangle)$
 $\quad \wedge \neg Committed(\langle Leader(2), 2 \rangle)$
 $\quad \wedge Committed(\langle Leader(3), 3 \rangle)$
 $)$

