

Specification of the *Sailfish* consensus algorithm at a high level of abstraction.

Compared to the *Sailfish1* specification, we additionally model committing with just  $f+1$  parents, as is possible in the *Sailfish* paper.

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** {

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

$VertexQuorums(r) \triangleq$

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

  }

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

**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** (  $vg \in VertexQuorums(round - 1)$  ) {

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

**when**  $round \geq GST \Rightarrow (N \setminus F) \subseteq \{ Node(v2) : v2 \in vg \};$

**if** (  $Leader(round) = self$  )

          we must either include the previous leader vertice,

          or a quorum of *no\_vote* messages:

**when**  $LeaderVertex(round - 1) \in vg$

$\vee \exists Q \in Quorum : \forall n \in Q : LeaderVertex(round - 1) \in no\_vote[n];$

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

**if** (  $LeaderVertex(round - 1) \notin vg$  ) send *no\_vote* if previous leader vertice 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  $r \leq GST$ ; 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 (  $byzantineNode \in F$  )
  variables  $round_- = 0$ ;
  {
l0: while ( TRUE ) {
    maybe add a vertices to the DAG:
    either with (  $v = \langle self, round_- \rangle$  ) {
       $vs := vs \cup \{v\}$ ;
      if (  $round_- > 0$  )
        with (  $vq \in VertexQuorums(round_- - 1)$  ) {
          if (  $Leader(round_-) = self$  )
            we must either include the previous leader vertice,
            or a quorum of no_vote messages:
            when  $LeaderVertice(round_- - 1) \in vq$ 
               $\vee \exists Q \in Quorum : \forall n \in Q : LeaderVertice(round_- - 1) \in no\_vote[n]$ ;
             $es := es \cup \{\langle v, pv \rangle : pv \in vq\}$ 
          }
        } or skip;
      maybe send a no_vote messages:
      if (  $round_- > 0$  )
        either
           $no\_vote[self] := no\_vote[self] \cup \{LeaderVertice(round_- - 1)\}$ 
        or skip;
      go to the next round:
       $round_- := round_- + 1$ 
    }
  }
}

```

Next we define the safety and liveness properties

$$\begin{aligned}
Committed(v) &\triangleq \\
&\wedge v \in vs \\
&\wedge Node(v) = Leader(Round(v)) \\
&\wedge \exists Bl \in Blocking : Bl \subseteq \{Node(pv) : pv \in Parents(v, es)\} \\
&\wedge \vee Round(v) = 0 \\
&\quad \vee LeaderVertice(Round(v) - 1) \in Children(v, es) \\
&\quad \vee \exists Q \in Quorum : \forall n \in Q : \\
&\quad \quad LeaderVertice(Round(v) - 1) \in no\_vote[n]
\end{aligned}$$

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

$$\begin{aligned}
\text{Liveness} &\triangleq \forall r \in R : \\
&\quad \wedge r \geq GST \\
&\quad \wedge \text{Leader}(r) \notin F \\
&\quad \text{all correct } round - (r + 1) \text{ vertices are created:} \\
&\quad \wedge \forall n \in N \setminus F : round[n] > r + 1 \\
&\quad \Rightarrow \text{Committed}(\text{LeaderVertice}(r))
\end{aligned}$$

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

$$\begin{aligned}
\text{Quorum1} &\triangleq \{Q \in \text{SUBSET } N : \text{Cardinality}(Q) \geq \text{Cardinality}(N) - \text{Cardinality}(F)\} \\
\text{Blocking1} &\triangleq \{Q \in \text{SUBSET } N : \text{Cardinality}(Q) > \text{Cardinality}(F)\}
\end{aligned}$$

The round of a node, whether *Byzantine* or not

$$\text{Round}_-(n) \triangleq \text{IF } n \in F \text{ THEN } round\_ [n] \text{ ELSE } round[n]$$

Basic typing invariant:

$$\begin{aligned}
\text{TypeOK} &\triangleq \\
&\quad \wedge \forall v \in vs : \text{Node}(v) \in N \wedge \text{Round}(v) \in Nat \\
&\quad \wedge \forall e \in es : \\
&\quad \quad \wedge e = \langle e[1], e[2] \rangle \\
&\quad \quad \wedge \{e[1], e[2]\} \subseteq vs \\
&\quad \quad \wedge \text{Round}(e[1]) > \text{Round}(e[2]) \\
&\quad \wedge \forall n \in N : \\
&\quad \quad \wedge \text{Round}_-(n) \in Nat \\
&\quad \quad \wedge no\_vote[n] \subseteq \{\langle \text{Leader}(r), r \rangle : r \in R\}
\end{aligned}$$

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.

$$\begin{aligned}
\text{SeqConstraints}(n) &\triangleq \\
&\quad \text{wait for all nodes to finish previous rounds:} \\
&\quad \wedge (\text{Round}_-(n) > 0 \Rightarrow \forall n2 \in N : \text{Round}_-(n2) \geq \text{Round}_-(n)) \\
&\quad \text{wait for all nodes with lower index to leave the round:} \\
&\quad \wedge \forall n2 \in N : \text{NodeIndex}(n2) < \text{NodeIndex}(n) \Rightarrow \text{Round}_-(n2) > \text{Round}_-(n)
\end{aligned}$$

$$\begin{aligned}
\text{SeqNext} &\triangleq (\exists self \in N \setminus F : \text{SeqConstraints}(self) \wedge \text{correctNode}(self)) \\
&\quad \vee (\exists self \in F : \text{SeqConstraints}(self) \wedge \text{byzantineNode}(self)) \\
\text{SeqSpec} &\triangleq \text{Init} \wedge \Box[\text{SeqNext}]_{vars}
\end{aligned}$$

Example assignment of leaders to rounds:

$$\text{ModLeader}(r) \triangleq \text{NodeSeq}[(r \% \text{Cardinality}(N)) + 1]$$

Constraint to stop the model checker:

$$\begin{aligned}
StateConstraint &\triangleq \\
&\text{LET } Max(S) \triangleq \text{CHOOSE } x \in S : \forall y \in S : y \leq x \text{ IN} \\
&\forall n \in N : Round\_ (n) \in 0 \dots (Max(R) + 1)
\end{aligned}$$

Some properties we expect to be violated:

$$\begin{aligned}
Falsy1 &\triangleq \neg( \\
&\quad \wedge Committed(\langle Leader(1), 1 \rangle) \\
& )
\end{aligned}$$

$$\begin{aligned}
Falsy2 &\triangleq \neg( \\
&\quad \wedge Committed(\langle Leader(0), 0 \rangle) \\
&\quad \wedge \neg Committed(\langle Leader(1), 1 \rangle) \\
&\quad \wedge \neg Committed(\langle Leader(2), 2 \rangle) \\
&\quad \wedge Committed(\langle Leader(3), 3 \rangle) \\
& )
\end{aligned}$$

\_\_\_\_\_