MODULE *Sailfish*

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

We use a number of abstractions and simplifiying assumptions in order to expose the high-level principles of the algorithm clearly and in order to make model-checking of interesting configuations tractable :

1) Nodes read and write a global DAG. Each round, each node gets to see an arbitrary quorum of vertices from the previous round (and, after GST, this quorum must include all correct vertices).

2) We do not model timeouts. Instead, we assume that, every round r after GST, correct nodes always rbc-deliver all correct vertices of round $r-1$ before entering round r. $TODO$: is this an acceptable assumption? Should it be just $f+1$ correct vertices?

4) We model *Byzantine* nodes explicitly by assigning them an algorithm. This algorithm should allow the worst attacks possible, but, while the author thinks this is true, there is no formal guarantee that this is the case. A more realistic model would allow *Byzantine* nodes to send completely arbitrary messages at any time, but this would make model-checking too hard.

5) We do not explicitly model committing based on $2f+1$ first RBC messages.

This version of the algorithm does not use "*no_vote*" messages.

EXTENDS *DomainModel*

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
define {
$LeaderVertice(r) \triangleq \langle Leader(r), r \rangle$
$ValidVerticeQuorums(r) \triangleq$ Quorums of valid vertices of round r
$\{VQ \in \text{SUBSET } vs : \text{LET } NQ \triangleq \{Node(v) : v \in VQ\} \text{IN}$
$\land\ NQ \in Quorum$
$\land\ \forall v \in VQ :$
$\land\ Round(v) = r$
the leader vertice, if included, must be valid ($i.e.$ if it does not point to the previous leader vertice, then a quorum of votes must justify that):
$\land\ \lor\ \neg(r > 0 \land v = LeaderVertice(r) \land \langle v, LeaderVertice(r-1) \rangle \notin es)$
$\lor\ \exists\, VQ2 \in \text{SUBSET } VQ :$
$\land\ VQ2 \in Quorum$
$\land\ \forall v2 \in VQ2 : \langle v2, LeaderVertice(r-1) \rangle \notin es\}$
}
process ($correctNode \in N \setminus F$)
variables
$round = 0$, current round
$log = \langle\rangle$**;** delivered log

$$
\begin{array}{l}
\quad \{ \\
l0\!: \quad \textbf{while } (\ \textsc{true}\ ) \\
\quad\quad \textbf{with } (\ v = \langle self,\ round \rangle\ )\ \{ \\
\quad\quad\quad \text{complete a round: add the new } DAG \text{ vertice } v \text{, and maybe commit new leader vertice} \\
\quad\quad\quad vs := vs \cup \{v\}\,; \\
\quad\quad\quad \textbf{if } (\ round > 0\ ) \\
\quad\quad\quad \textbf{with } (\ VQ \in ValidVerticeQuorums(round - 1)\ )\ \{ \\
\quad\quad\quad\quad \text{from } GST \text{ onwards, each node receives all correct vertices of the previous round:} \\
\quad\quad\quad\quad \textbf{when } round \geq GST \Rightarrow (N \setminus F) \subseteq \{Node(v2) : v2 \in VQ\}\,; \\
\quad\quad\quad\quad \textbf{if } (\ Leader(round) = self\ )\ \{ \\
\quad\quad\quad\quad\quad \text{we must either include the previous leader vertice,} \\
\quad\quad\quad\quad\quad \text{or we must have seen a quorum of vertices not voting for the previous leader vertice} \\
\quad\quad\quad\quad\quad \textbf{when} \\
\quad\quad\quad\quad\quad\quad \lor\ LeaderVertice(round - 1) \in VQ \\
\quad\quad\quad\quad\quad\quad \lor\ \exists\, Q \in Quorum : \forall\, n \in Q \setminus \{self\} : \text{LET } vn \triangleq \langle n,\ round \rangle \text{IN} \\
\quad\quad\quad\quad\quad\quad\quad \land\ vn \in vs \\
\quad\quad\quad\quad\quad\quad\quad \land\ \langle vn,\ LeaderVertice(round - 1) \rangle \notin es\,; \\
\quad\quad\quad\quad \}\ ; \\
\quad\quad\quad\quad es := es \cup \{\langle v,\ pv \rangle : pv \in VQ\}\,; \quad \text{add } v\text{'s edges} \\
\quad\quad\quad\quad \text{possibly commit the leader vertice of round } r - 2\text{:} \\
\quad\quad\quad\quad \textbf{if } (\ round > 1\ ) \\
\quad\quad\quad\quad\quad \textbf{with } (\ votesForLeader = \{pv \in VQ : \langle pv,\ LeaderVertice(round - 2) \rangle \in es\}\ ) \\
\quad\quad\quad\quad\quad \textbf{if } (\ \{Node(pv) : pv \in votesForLeader\} \in Quorum\ ) \\
\quad\quad\quad\quad\quad\quad log := OrderDAG(es,\ [i \in 1 \mathrel{..} (round - 2) \mapsto LeaderVertice(i)]) \\
\quad\quad\quad \}\ ; \\
\quad\quad\quad round := round + 1 \\
\quad\quad \} \\
\quad \}
\end{array}
$$

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). Also note that creating a round-r vertice commutes to the left of actions of rounds greater than r and to the right of actions of rounds smaller than R, so we can, without loss of generality, schedule *Byzantine* nodes in the same "round-by-round" manner as other nodes.

$$
\begin{array}{l}
\quad \textbf{process } (\ byzantineNode \in F\ ) \\
\quad\quad \textbf{variables } round\_ = 0\,; \\
\quad \{ \\
l0\!: \quad \textbf{while } (\ \textsc{true}\ )\ \{ \\
\quad\quad\quad \text{maybe add a vertices to the } DAG\text{:} \\
\quad\quad\quad \textbf{either with } (\ v = \langle self,\ round\_ \rangle\ )\ \{ \\
\quad\quad\quad\quad vs := vs \cup \{v\}\,; \\
\quad\quad\quad\quad \textbf{if } (\ round\_ > 0\ ) \\
\quad\quad\quad\quad \textbf{with } (\ vq \in ValidVerticeQuorums(round\_ - 1)\ )\ \{ \\
\quad\quad\quad\quad\quad es := es \cup \{\langle v,\ pv \rangle : pv \in vq\} \\
\quad\quad\quad\quad \}
\end{array}
$$

} or skip ;

go to the next round:

$round\_ := round\_ + 1$

}

}

}

Next we define the safety and liveness properties

$Committed(v, view) \triangleq$ view intended to be a sub-DAG of the *DAG es*

$\quad \land\ v \in view$

$\quad \land\ Node(v) = Leader(Round(v))$

$\quad \land\ \exists\, Bl \in Blocking : Bl \subseteq \{Node(pv) : pv \in Parents(v, es) \cap view\}$

$\quad \land\ \lor\ Round(v) = 0$

$\quad\quad \lor\ LeaderVertice(Round(v) - 1) \in Children(v, es)$

$\quad\quad \lor\ \exists\, Q \in Quorum : \forall\, n \in Q : \text{LET } vn \triangleq \langle n, Round(v)\rangle \text{IN}$

$\quad\quad\quad \land\ vn \in view$

$\quad\quad\quad \land\ \langle vn, LeaderVertice(Round(v) - 1)\rangle \notin es$

$Safety \triangleq \forall\, n1, n2 \in N \setminus F :$

$\quad Compatible(log[n1], log[n2])$

TODO: update *Livenes* to use local logs

$Liveness \triangleq \forall\, r \in R :$

$\quad \land\ r \geq GST$

$\quad \land\ Leader(r) \notin F$

all correct $round - (r + 1)$ vertices are created:

$\quad \land\ \forall\, n \in N \setminus F : round[n] > r + 1$

$\quad \Rightarrow Committed(LeaderVertice(r), vs)$

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

$Quorum1 \triangleq \{Q \in \text{SUBSET } N : Cardinality(Q) \geq Cardinality(N) - Cardinality(F)\}$

$Blocking1 \triangleq \{Q \in \text{SUBSET } N : Cardinality(Q) > Cardinality(F)\}$

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

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

Basic typing invariant:

$TypeOK \triangleq$

$\quad \land\ \forall\, v \in vs : Node(v) \in N \land Round(v) \in Nat$

$\quad \land\ \forall\, e \in es :$

$\quad\quad \land\ e = \langle e[1], e[2]\rangle$

$\quad\quad \land\ \{e[1], e[2]\} \subseteq vs$

$\quad\quad \land\ Round(e[1]) > Round(e[2])$

$\quad \land\ \forall\, n \in N : Round\_(n) \in Nat$

Sequentialization constraints, which enforce a particular ordering of the actions. Because of how actions commute, the set of reachable states remains unchanged. Essentially, we schedule all nodes "round-by-round" and in lock-steps, with the leader last. This speeds up model-checking a lot.

Note that we must always schedule the leader last because, due to its use of other nodes's vertices, its action does not commute to the left of the actions of other nodes.

$SeqConstraints(n) \triangleq$

wait for all nodes to finish previous rounds:

$\land (Round\_(n) > 0 \Rightarrow \forall n2 \in N : Round\_(n2) \geq Round\_(n))$

wait for all nodes with lower index to leave the round (leader index is always last):

$\land \forall n2 \in N : NodeIndexLeaderLast(n2, Round\_(n)) < NodeIndexLeaderLast(n, Round\_(n)) \Rightarrow Round$

$SeqNext \triangleq (\exists self \in N \setminus F : SeqConstraints(self) \land correctNode(self))$
$\qquad \lor (\exists self \in F : SeqConstraints(self) \land byzantineNode(self))$

$SeqSpec \triangleq Init \land \Box[SeqNext]_{vars}$

Example assignment of leaders to rounds:

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

Constraint to stop the model checker:

$StateConstraint \triangleq$
$\quad$ LET $Max(S) \triangleq$ CHOOSE $x \in S : \forall y \in S : y \leq x$IN
$\qquad \forall n \in N : Round\_(n) \in 0 \mathrel{..} (Max(R) + 1)$

Some properties we expect to be violated (useful to get the model-checker to print interesting executions):

$Falsy1 \triangleq \neg($ we commit something in round 1
$\quad \exists n \in N \setminus F : log[n] \neq \langle\rangle \land Round(log[n][Len(log[n])]) \neq 0$
$)$

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