Analyzing Paxos with Fault-Tolerant Multiparty Session Types

Bachelor thesis by Nicolas Daniel Torres
Date of submission: October 4, 2021

1. Review: Prof. Dr. Kirstin Peters
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department
<institute>
<working group>

Contents

1 Introduction

In distributed computing, it is often necessary for coordinating processes to reach consensus, i.e., agree on the value of some data that are needed during computation. These processes agree on the same values to ensure correct computation, which necessitates a correct consensus algorithm. Thus, proving the correctness of consensus algorithms is important. To achieve consensus these algorithms must satisfy the following properties: termination, validity, and agreement [1].

Due to the presence of faulty processes consensus algorithms are designed to be fault-tolerant.

Proving these properties can be complicated. Model checking tools lead to big state-spaces so static analysis is preferable. Multiparty Session Types are particularly interesting since session typing can ensure the absence of communication errors and deadlocks, and protocol conformance [3]. However, to properly model unreliable communication between processes a fault-tolerant extension to Multiparty Session Types is necessary.

Peters, Nestmann, and Wagner developed such an extension.

2 Technical Preliminaries

First, we define the sorts, some additional notation, and use them to define the global type. Afterwards we define some sets and functions to create the processes.

3 Model and Analysis

First, we specify some sorts with which we can then define the global type. Afterwards, we define the processes for the proposer and the acceptor. Finally, we will study an example run of the model.

3.0.1 Sorts

We assume the following sorts.

$\text{Maybe } a = \{\text{Just } a, \text{Nothing}\}$

$\text{Value} =$ Set of values.

$\text{Promise } a = \{\text{Promise } (\text{Maybe } (\text{Proposal } a)), \text{Nack } \mathbb{N}\}$

$\text{Proposal } a = \{\text{Proposal } \mathbb{N}\ a\}$

3.0.2 Global Type

Since each proposer initiates its own session the global type can be defined for one proposer. A quorum of acceptors A_Q is assumed.

The last phase of Paxos contains no inter-process communication, so it is not modeled in the global type.

$$
\begin{aligned}
&G_{p,A_Q} = (\mu X) \bigodot_{a \in A_Q} p \to_u a : l1a \langle \mathbb{N} \rangle . \bigodot_{a \in A_Q} a \to_u p : l1b \langle \text{Promise Value} \rangle . \\
&\quad p \to_w A_Q : \mathit{Accept}. \left(\bigodot_{a \in A_Q} p \to_u a : l2a \langle \text{Proposal Value} \rangle \right) . \mathit{end} \\
&\qquad \oplus \mathit{Restart}.X \\
&\qquad \oplus \mathit{Abort}.\mathit{end}
\end{aligned}
$$

We can distinguish the individual phases of the Paxos algorithm by the labels $l1a$, $l1b$, and $l2a$.

In the first two steps, $1a$ and $1b$, the proposer sends its proposal number to each acceptor in A_Q and listens for their responses. In step 2a the proposer decides whether to send an $Accept$ or $Restart$ message to restart the algorithm. This decision is broadcast to all acceptors in A_Q. Should the proposer crash the algorithm ends for this particular proposer and quorum of acceptors.

3.0.3 Functions and Sets

$\text{Bool} = \{true, false\}$

$\text{prNumber} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ returns a proposal number when given two natural numbers.

$\text{promValue} : [\text{Promise } a] \to a$ if none of the promises in the given list contain a value a new value is returned. A promise contains a value if it is of the form $\text{Promise (Just } v)$. v is the value.

$$\begin{aligned}
&\text{anyNack} : [\text{Promise } a] \to \text{Bool} \\
&\text{anyNack } [] = false \\
&\text{anyNack } ((\text{Nack } _ : _)) = true \\
&\text{anyNack } ((_ : xs)) = \text{anyNack } (xs)
\end{aligned}$$

$$\begin{aligned}
&\text{promCount} : [\text{Promise } a] \to \mathbb{N} \\
&\text{promCount } [] = 0 \\
&\text{promCount } ((\text{Promise } _ : xs)) = 1 + \text{promCount } (xs) \\
&\text{promCount } ((_ : xs)) = \text{promCount } (xs)
\end{aligned}$$

$$\begin{aligned}
&\text{gt} : a \to \text{Maybe } a \to \text{Bool} \\
&\text{gt } (_, \text{Nothing}) = true \\
&\text{gt } (a, \text{Just } b) = a > b
\end{aligned}$$

$$\begin{aligned}
&\text{ge} : a \to \text{Maybe } a \to \text{Bool} \\
&\text{ge } (_, \text{Nothing}) = true \\
&\text{ge } (a, \text{Just } b) = a \geq b
\end{aligned}$$

$$\begin{aligned}
&\text{nFromPr} : \text{Proposal } a \to \mathbb{N} \\
&\text{nFromPr } (\text{Proposal } n \ _) = n
\end{aligned}$$

$\text{genA}_\text{Q}\ (i, ac, pc)$ returns a set A_Q with $A_Q \subseteq A = \{1, \ldots, ac\}$ and $|A_Q| > \frac{|A|}{2}$.

3.0.4 Processes

System Initialization

$$\begin{aligned}&\mathrm{Sys}\left(ac, pc\right) = \overline{a}\left[2\right]\left(t\right).\,\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\left(ac+1, \mathrm{genA_Q}\left(ac+pc, ac, pc\right), ac+pc, []\right)\\ &\mid a\left[1\right]\left(t\right).\Pi_{ac<i<ac+pc}\ \mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\left(ac+1, \mathrm{genA_Q}\left(i, ac, pc\right), i, []\right)\\ &\mid \Pi_{1\leq j\leq ac}\ \mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\left(j, ac+1, ac, pc, n_j, pr_j\right)\end{aligned}$$

$$\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\left(i, A_Q, n, \overrightarrow{V}\right) = \overline{b_n}\left[i\right]\left(s\right).\,\mathrm{P}^{\mathrm{p}}$$

$$\mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\left(j, i, ac, pc, n, pr\right) = \Pi_{ac<k\leq ac+pc}\ b_k\left[j\right]\left(s\right).\,\mathrm{P}^{\mathrm{a}}$$

$\mathrm{Sys}\left(ac, pc\right)$, $\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\left(i, A_Q, n, \overrightarrow{V}\right)$, and $\mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\left(j, i, ac, pc, n, pr\right)$ describe the system initialization. ac and pc are the number of acceptors and proposers respectively.

An outer session is created through shared-point a. This outer session is not strictly necessary but was left in to allow for easier extension of the model. The acceptors are initialized using indices from 1 to ac and the proposers are initialized using indices from $ac+1$ to $ac+pc$.

$\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\left(i, A_Q, n, \overrightarrow{V}\right)$ is initialized with the proposer's role in its own session i, which is always $ac+1$, a quorum of acceptors A_Q, an index n, and a vector $\overrightarrow{V}$. Each proposer has the same role $i = ac+1$ but uses a different shared-point b_n according to its index n. $\overrightarrow{V}$ is used in the proposer to collect and evaluate the responses from the acceptors. It is always initialized with an empty list $[]$. Shared-point b_n is used to initiate a session. Afterwards, the process behaves like $\mathrm{P^p}$.

$\mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\left(j, i, ac, pc, n, pr\right)$ is initialized with the acceptor's index j, the proposer index i, which is always $ac+1$, ac, pc, initial knowledge for the highest promised proposal number n, if available, and initial knowledge for the most recently accepted proposal pr, if available. n is of type $\mathrm{Maybe}\ \mathbb{N}$ and pr is of type $\mathrm{Maybe}\ \left(\mathrm{Proposal}\ \mathrm{Value}\right)$ thus both can be $\mathrm{Nothing}$. Each of the proposers' session requests are accepted in a separate subprocess. These subprocesses run parallel to each other but still access the same values for n and pr. Afterwards, each subprocess behaves like $\mathrm{P^a}$.

Proposer

$$
\begin{array}{l}
\mathrm{P}^{\mathrm{p}} = (\mu X)\,\mathrm{update}\,(n, n+1)\,.\,\left(\bigodot_{j \in A_Q}\ s\,[i,j]!_u l1a\,\langle \mathrm{prNumber}\,(n,i)\rangle\right). \\
\quad \left(\bigodot_{j \in A_Q}\ s\,[j,i]?_u l1b\,\langle \bot\rangle\,(v_j)\right). \\
\quad \mathrm{if}\ \ \mathrm{anyNack}\left(\overrightarrow{V}\right)\ \ \mathrm{or}\ \ \mathrm{promCount}\left(\overrightarrow{V}\right) < \left\lceil \frac{|A_Q|}{2} \right\rceil \\
\quad\ \ \mathrm{then}\ \ s\,[i, A_Q]!_w Restart.X \\
\quad\ \ \mathrm{else} \\
\quad\ \ s\,[i, A_Q]!_w Accept.\bigodot_{j \in A_Q}\ s\,[i,j]!_u l2a\left\langle \mathrm{Proposal}\ \ \mathrm{prNumber}\,(n,i)\ \ \mathrm{promValue}\left(\overrightarrow{V}\right)\right\rangle. \\
\quad\ \ end
\end{array}
$$

Acceptor

$$
\begin{array}{l}
\mathrm{P}^{\mathrm{a}} = (\mu X)\,s\,[i,j]?_u l1a\,\langle \bot\rangle\,(n')\,. \\
\quad \mathrm{if}\ \ n' = \bot \\
\quad\ \ \mathrm{then}\ \ \mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} \\
\quad\ \ \mathrm{else} \\
\quad\ \ \mathrm{if}\ \ \mathrm{gt}\,(n', n) \\
\quad\ \ \mathrm{then}\ \ \mathrm{update}\,(n, n')\,.s\,[j,i]!_u l1b\,\langle \mathrm{Promise}\ \ pr\rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} \\
\quad\ \ \mathrm{else}\ \ s\,[j,i]!_u l1b\,\langle \mathrm{Nack}\ \ n\rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} \\
\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} = s\,[i,j]?_w Accept.s\,[i,j]?_u l2a\,\langle \bot\rangle\,(pr')\,. \\
\quad \mathrm{if}\ \ pr' = \bot \\
\quad\ \ \mathrm{then}\ \ X \\
\quad\ \ \mathrm{else} \\
\quad\ \ \mathrm{if}\ \ \mathrm{ge}\,(\mathrm{nFromPr}\,(pr')\,, n) \\
\quad\ \ \mathrm{then}\ \ \mathrm{update}\,(pr, pr')\,.\,\mathrm{update}\,(n, \mathrm{Just}\ \ \mathrm{nFromPr}\,(pr'))\,.X \\
\quad\ \ \mathrm{else}\ \ X \\
\quad \oplus Restart.X \\
\quad \oplus Abort.end
\end{array}
$$

3.1 Example

$ac = 3\ pc = 2$

$$
\begin{aligned}
&\mathrm{Sys}\,(3,2) = a\,[1]\,(t)\,.\Pi_{3<i<5}\ \mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\,(4, \mathrm{genA_Q}\,(i,3,2)\,, i, [])\\
&\mid \overline{a}\,[2]\,(t)\,.\,\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\,(4, \mathrm{genA_Q}\,(5,3,2)\,, 5, [])\\
&\mid \Pi_{1\leq j\leq 3}\ \mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\,(j,4,3,2,n_j,pr_j)\\
&\longmapsto (\mathrm{Init})\\
&(\nu t)\,\big(\,\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\,(4, A_{Q,1}, 4, [])\ \mid\ \mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\,(4, A_{Q,2}, 5, [])\\
&\mid\ \mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\,(1,4,3,2,n_1,pr_1)\ \mid\ \mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\,(2,4,3,2,n_2,pr_2)\ \mid\ \mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\,(3,4,3,2,n_3,pr_3)\,\big)
\end{aligned}
$$

4 Evaluation

RESULTS

5 Discussion

DISCUSSION