Two Phase Commit Protocol

Murat Demirbas

[2019-08-26 Mon]

2 phase commit

A transaction is performed over resource managers (RMs)

The transaction manager (TM) finalizes the transaction

- For the transaction to be committed, each participating RM must be prepared to commit it
- Otherwise, the transaction must be aborted

Definitions

```
--fair algorithm 2PC{
variable rmState = [rm ∈ RM ↦ "working"],
         tmState = "init" ;

define {
   canCommit ≜ ∀ rm ∈ RM : rmState[rm] ∈ {"prepared"}
   canAbort  ≜ ∃ rm ∈ RM : rmState[rm] ∈ {"aborted"}
 }
```

TM modeling

```
   fair process ( TManager = 0 ) {
TM:  either
       { await canCommit ;
         tmState := "commit" ;
F1:      if ( TMMAYFAIL ) tmState := "unavailable" ;  }
     or
       { await canAbort ;
         tmState := "abort" ;
F2:      if ( TMMAYFAIL ) tmState := "unavailable" ;  }
   }
```

TM modeling

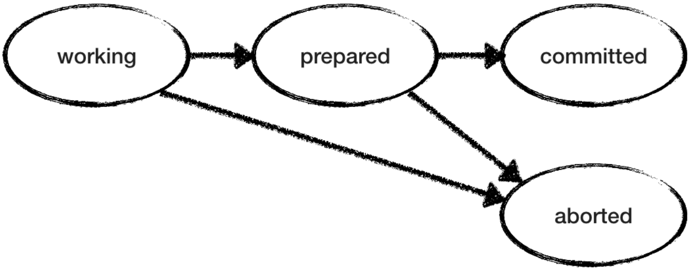
TM checks if it `canCommit` or `canAbort` and updates `tmState` accordingly.

TM can also fail making `tmState` "unavailable"

To keep things simple yet interesting, TM fails only after it makes a decision. These two updates are nonatomic:

- `tmState` is available for RMs to read for a duration
- labels at fail actions provide nonatomicity

RM modeling



RM modeling

```
    fair process ( RManager ∈ RM ) {
RS: while ( rmState[self] ∉ {"committed", "aborted"} ) {
          either {
             await rmState[self] = "working" ;
             with ( x ∈ {"prepared", "aborted"} ) rmState[self] := x ;  }
          or {
             await rmState[self] = "prepared" ∧ tmState = "commit" ;
RC:          rmState[self] := "committed" ;  }
         or  {
             await rmState[self] = "prepared" ∧ tmState = "abort" ;
RA:          rmState[self] := "aborted" ;  }
        }
    }
```

Invariants

$$Consistency \triangleq$$

A state predicate asserting that two *RMs* have not arrived at conflicting decisions.

$$\forall\, rm1,\, rm2 \in RM : \neg \land\, rmState[rm1] = \text{“aborted”}$$
$$\land\, rmState[rm2] = \text{“committed”}$$

$$Completed \triangleq \Diamond(\forall\, rm \in RM : rmState[rm] \in \{\text{“committed”}, \text{“aborted”}\})$$

Consistency checks that there are no 2 RMs such that one says "committed" and other says "aborted"

Model checking

If TM does not fail, the 2-phase commit algorithm is correct

When TM fails, termination can be violated

We add a Backup TM, to take over, and achieve termination

BTM modeling

```
     fair process ( BTManager = 10 ) {
BTM: either
          { await canCommit ∧ tmState = "unavailable" ;
BTC:        tmState := "commit" ;  }
        or
          { await canAbort ∧ tmState = "unavailable" ;
BTA:        tmState := "abort" ;  }
     }
```

Strengthening canCommit

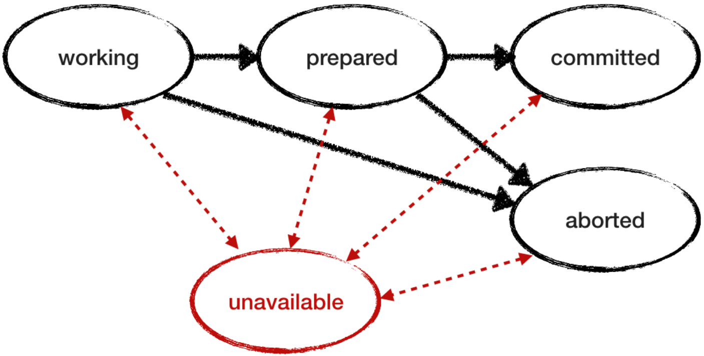
```
--fair algorithm 2PC{
variable rmState = [rm ∈ RM ↦ "working"],
         tmState = "init" ;

define {
   canCommit ≜    ∀ rm ∈ RM  : rmState[rm] ∈ {"prepared"}
               ∨  ∃ rmc ∈ RM : rmState[rmc] ∈ {"committed"}   for BTM
   canAbort ≜    ∃ rm ∈ RM : rmState[rm] ∈ {"aborted"}
 }
```

BTM modeling

BTM takes over when TM is `unavailable` and uses the same logic to make decisions

For simplicity we assume the BTM does not fail

What if RMs could also fail?



What if RMs could also fail?

```
  fair process ( RManager ∈ RM )
  variables pre = "" ; {
RS:  while ( rmState[self] ∉ {"committed", "aborted"} ) {
         either {
           await rmState[self] = "working" ;
           with ( x ∈ {"prepared", "aborted"} ) rmState[self] := x ;  }
         or {
           await rmState[self] = "prepared" ∧ tmState = "commit" ;
RC:        rmState[self] := "committed" ;  }
        or  {
           await rmState[self] = "prepared" ∧ tmState = "abort" ;
RA:        rmState[self] := "aborted" ;  }
        or  {
           await RMMAYFAIL ∧ pre ≠ rmState[self] ;
           pre := rmState[self] ;
           rmState[self] := "unavailable" ;
RR:        rmState[self] := pre ;
            }
       }
   }
```

Strengthening canAbort

$$
\begin{aligned}
&\textbf{define } \{ \\
&\quad \mathit{canCommit} \triangleq \quad \forall\, rmc \in RM : rmState[rmc] \in \{\text{“prepared”}\} \\
&\qquad\qquad\qquad\quad \lor\ \exists\, rm \in RM : rmState[rm] \in \{\text{“committed”}\} \quad \text{for } BTM \\
&\quad \mathit{canAbort} \triangleq \quad \exists\, rm \in RM : rmState[rm] \in \{\text{“aborted”}, \text{“unavailable”}\} \\
&\qquad\qquad\qquad \land\ \neg\exists\, rmc \in RM : rmState[rmc] = \text{“committed”} \quad \text{for } BTM \\
&\}
\end{aligned}
$$

Inconsistency problem!

<RS line 90, col 13 to line 111, col 32 of module 2PC2withBTM>	State (num = 6)
pc	(0 :> "F1" @@ 1 :> "RC" @@ 2 :> "RS" @@ 3 :> "RS" @@ 10 :> "BTM")
pre	<<"", "", "">>
rmState	<<"prepared", "prepared", "prepared">>
tmState	"commit"
<F1 line 139, col 7 to line 145, col 37 of module 2PC2withBTM>	State (num = 7)
pc	(0 :> "Done" @@ 1 :> "RC" @@ 2 :> "RS" @@ 3 :> "RS" @@ 10 :> "BTM")
pre	<<"", "", "">>
rmState	<<"prepared", "prepared", "prepared">>
tmState	"unavailable"
<RS line 90, col 13 to line 111, col 32 of module 2PC2withBTM>	State (num = 8)
pc	(0 :> "Done" @@ 1 :> "RC" @@ 2 :> "RR" @@ 3 :> "RS" @@ 10 :> "BTM")
pre	<<"", "prepared", "">>
rmState	<<"prepared", "unavailable", "prepared">>
tmState	"unavailable"
<BTM line 157, col 8 to line 162, col 47 of module 2PC2withBTM>	State (num = 9)
pc	(0 :> "Done" @@ 1 :> "RC" @@ 2 :> "RR" @@ 3 :> "RS" @@ 10 :> "BTA")
pre	<<"", "prepared", "">>
rmState	<<"prepared", "unavailable", "prepared">>
tmState	"unavailable"
<BTA line 169, col 8 to line 172, col 38 of module 2PC2withBTM>	State (num = 10)
pc	(0 :> "Done" @@ 1 :> "RC" @@ 2 :> "RR" @@ 3 :> "RS" @@ 10 :> "Done")
pre	<<"", "prepared", "">>
rmState	<<"prepared", "unavailable", "prepared">>
tmState	"abort"
<RC line 113, col 13 to line 116, col 43 of module 2PC2withBTM>	State (num = 11)
pc	(0 :> "Done" @@ 1 :> "RS" @@ 2 :> "RR" @@ 3 :> "RS" @@ 10 :> "Done")
pre	<<"", "prepared", "">>
rmState	<<"committed", "unavailable", "prepared">>
tmState	"abort"
<RS line 90, col 13 to line 111, col 32 of module 2PC2withBTM>	State (num = 12)
pc	(0 :> "Done" @@ 1 :> "RS" @@ 2 :> "RR" @@ 3 :> "RA" @@ 10 :> "Done")
pre	<<"", "prepared", "">>
rmState	<<"committed", "unavailable", "prepared">>
tmState	"abort"
<RA line 118, col 13 to line 121, col 43 of module 2PC2withBTM>	State (num = 13)
pc	(0 :> "Done" @@ 1 :> "RS" @@ 2 :> "RR" @@ 3 :> "RS" @@ 10 :> "Done")
pre	<<"", "prepared", "">>
rmState	<<"committed", "unavailable", "aborted">>

What went on?

- RM1 sees commit from TM
- TM becomes unavailable
- RM2 becomes unavailable
- BTM takes over for TM
- BTM decides on abort seeing <prepared, unavailable, prepared> from RMs. (It may also be that RM1 may also look unavailable due to unreachability)
- RM1 acts on commit from TM
- RM3 sees abort from BTM
- RM3 acts on abort from BTM
- Consistency is volated

Inconsistency problem!

If BTM decides, it may decide incorrectly, violating consistency

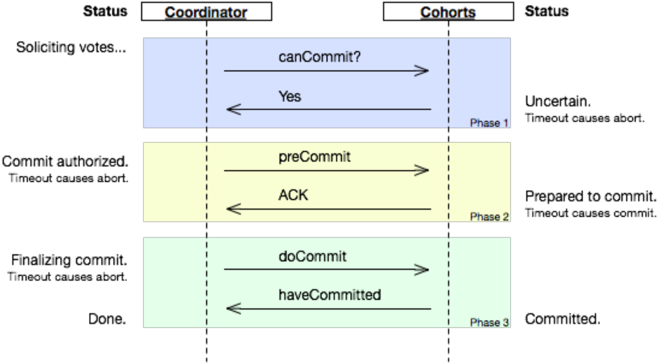
If BTM waits, it may be waiting forever on a crashed node, and violating termination (i.e., progress)

Timeouts may be incorrect, due to inopportune timing

FLP impossibility

In an asynchronous system, it is impossible to solve consensus (both safety and progress) in the presence of crash faults

What about 3PC?



3PC problems

BTM may act as if TM is down

RMs go with whatever TM or BTM says (2 leaders)

This asymmetry of information is the root of all evil in distributed systems

Lamport & Gray on transaction commit

"Several 3-phase protocols have been proposed, and a few have been implemented. They have usually attempted to "fix" the 2-Phase Commit protocol by choosing another TM if the first TM fails. However, we know of none that provides a complete algorithm proven to satisfy a clearly stated correctness condition. For example, the discussion of non-blocking commit in the classic text of Bernstein, Hadzilacos, and Goodman fails to explain what a process should do if it receives messages from two different processes, both claiming to be the current TM. Guaranteeing that this situation cannot arise is a problem that is as difficult as implementing a transaction commit protocol."

Paxos!

Paxos is always safe even in presence of inaccurate failure detectors, asynchronous execution, faults, and eventually makes progress when consensus gets in the realm of possibility

Paxos makes progress when the system is outside the realm of consensus impossibility

You can emulate TM with a Paxos cluster of 3 nodes and solve the inconsistent TM/BTM problem (Google Spanner approach)