Distributed Systems

ALI KAMANDI, PH.D.

SCHOOL OF ENGINEERING SCIENCE

COLLEGE OF ENGINEERING

UNIVERSITY OF TEHRAN

KAMANDI@UT.AC.IR

Distributed Database Commit

چند سایت در اجرای یک تراکنش نقش دارند. وقتی هر یک از سایتها وظیفه خود را به پایان رساند، باید تصمیم گیری کند که تراکنش commit شود یا abort؟

در صورتی که هر سایت توانسته باشد کارهای مربوط به خود را به صورت محلی انجام دهد، ترجیح می دهد تصمیم commit بگیرد و در غیر اینصورت abort

وقتی کامیت انجام می شود که همه سایت ها روی آن توافق داشته باشند.

ترجیحا توافق باید روی کامیت انجام شود.

مدل مساله

فرض می کنیم خطای ارسال پیام (لینک ارتباطی) نداریم و صرفا خطای سایت یا نود داریم. (Process Failure)

مجموعه تصمیم گیری شامل {0,1} است: ۰ به معنای abort و ۱ به معنای کامیت خواهد بود.

شبکه را گراف کامل در نظر می گیریم.

شرایط

Agreement: No two processes decide on different values.

Validity:

1. If any process starts with 0, then 0 is the only possible decision value.

2. If all processes start with 1 and there are no failures, then 1 is the only possible decision value.

Termination:

The **weak termination** condition says that if there are no failures then all processes eventually decide.

The **strong termination** condition (also known as the non-blocking condition) says that all non-faulty processes eventually decide.

MODULE *TCommit*

CONSTANT *RM* — The set of participating resource managers

VARIABLE *rmState* — *rmState*[*rm*] is the state of resource manager *rm*.

$$TCTypeOK \triangleq$$
$$\quad rmState \in [RM \rightarrow \{\text{“working”}, \text{“prepared”}, \text{“committed”}, \text{“aborted”}\}]$$

$$TCInit \triangleq \quad rmState = [r \in RM \mapsto \text{“working”}]$$

$$canCommit \triangleq \forall r \in RM : rmState[r] \in \{\text{"prepared"}, \text{"committed"}\}$$

$$notCommitted \triangleq \forall r \in RM : rmState[r] \neq \text{"committed"}$$

$$Prepare(r) \triangleq \begin{array}{l} \land\ rmState[r] = \text{"working"} \\ \land\ rmState' = [rmState \ \text{EXCEPT} \ ![r] = \text{"prepared"}] \end{array}$$

$$Decide(r) \triangleq \begin{array}{l} \lor \begin{array}{l} \land\ rmState[r] = \text{"prepared"} \\ \land\ canCommit \\ \land\ rmState' = [rmState \ \text{EXCEPT} \ ![r] = \text{"committed"}] \end{array} \\ \lor \begin{array}{l} \land\ rmState[r] \in \{\text{"working"}, \text{"prepared"}\} \\ \land\ notCommitted \\ \land\ rmState' = [rmState \ \text{EXCEPT} \ ![r] = \text{"aborted"}] \end{array} \end{array}$$

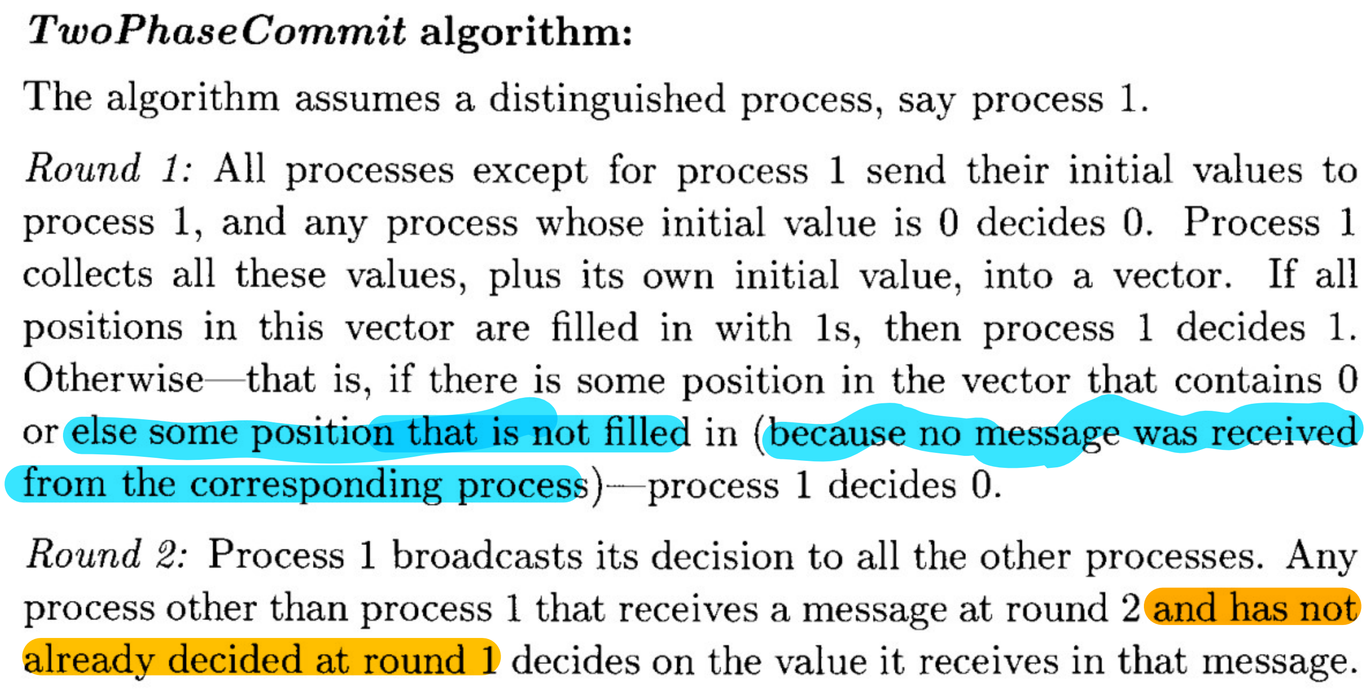
$$TCNext \triangleq \exists r \in RM : Prepare(r) \lor Decide(r)$$

$$TCConsistent \triangleq \forall r1, r2 \in RM : \neg \land rmState[r1] = \text{"aborted"} \land rmState[r2] = \text{"committed"}$$

$$TCSpec \triangleq TCInit \land \Box[TCNext]_{rmState}$$

$$\text{THEOREM } TCSpec \Rightarrow \Box(TCTypeOK \land TCConsistent)$$

Two-Phase Commit



***TwoPhaseCommit* algorithm:**

The algorithm assumes a distinguished process, say process 1.

Round 1: All processes except for process 1 send their initial values to process 1, and any process whose initial value is 0 decides 0. Process 1 collects all these values, plus its own initial value, into a vector. If all positions in this vector are filled in with 1s, then process 1 decides 1. Otherwise—that is, if there is some position in the vector that contains 0 or else some position that is not filled in (because no message was received from the corresponding process)—process 1 decides 0.

Round 2: Process 1 broadcasts its decision to all the other processes. Any process other than process 1 that receives a message at round 2 and has not already decided at round 1 decides on the value it receives in that message.

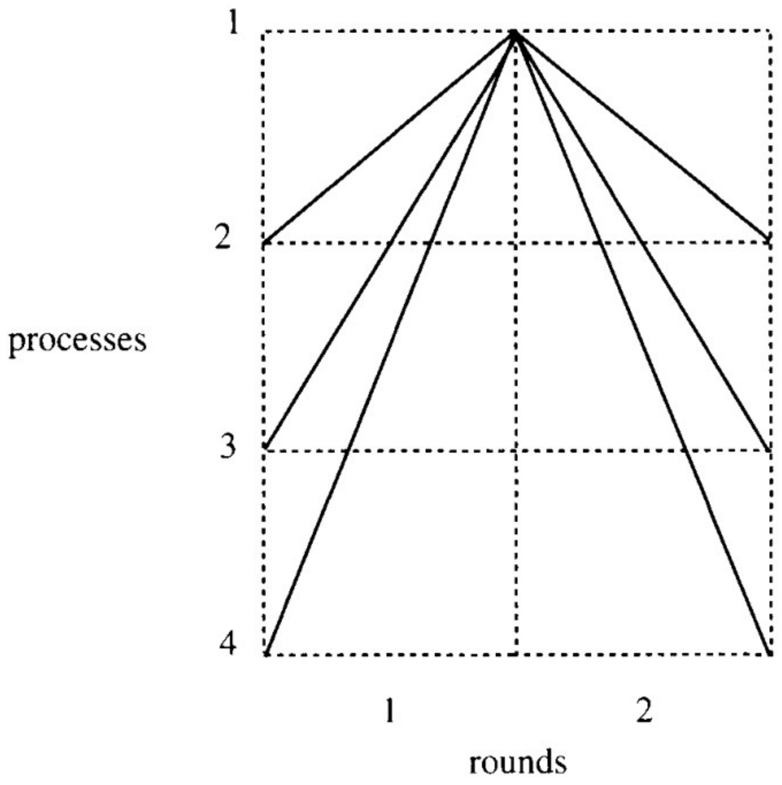


Figure 7.4: Communication pattern in *TwoPhaseCommit*.

- ❑Agreement
- ❑Validity
- ❑Weak Termination
- ❑Strong Termination

if process 1 fails before beginning its broadcast in round 2, then no nonfaulty process whose initial value is 1 ever decides.

Strong Termination

اگر پروسس ۱ قبل از اعلام نتیجه از دسترس خارج شود؟

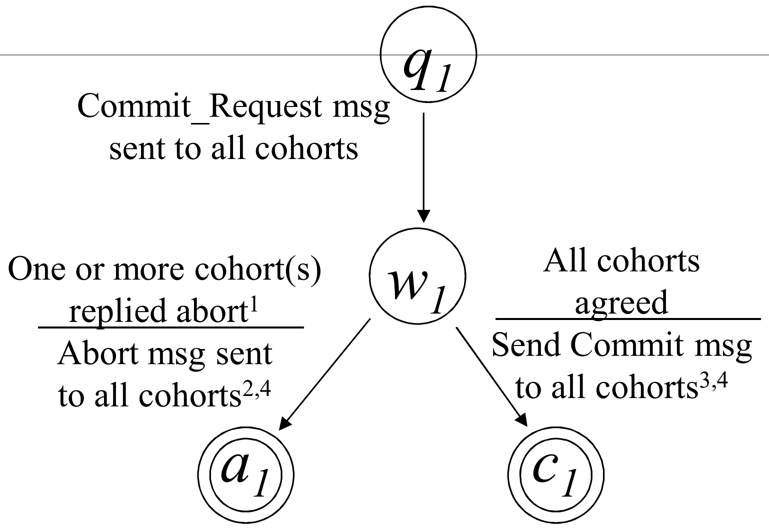
راه حل: یکی از سایت هایی که مثلا نظر ۰ داشته به بقیه اطلاع دهد.

اگر همه روی ۱ توافق داشته باشند و ۱ قبل از اینکه هر گونه پیامی ارسال کند از دسترس خارج شود؟

- بقیه سایت ها از نظر ۱ مطلع نبوده و نمی توانند تصمیم گیری کنند.

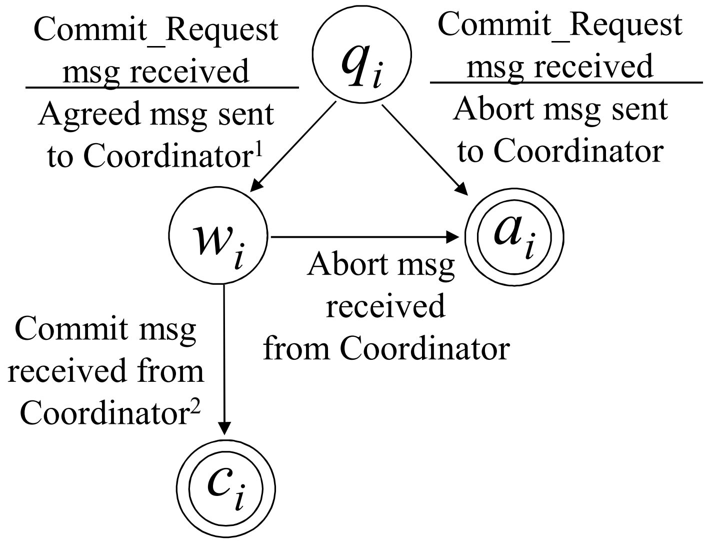
2-phase Commit Protocol

Coordinator


q_1
Commit_Request msg sent to all cohorts
w_1
One or more cohort(s) replied abort[1]
Abort msg sent to all cohorts[2,4]
All cohorts agreed
Send Commit msg to all cohorts[3,4]
a_1
c_1

1. Assume ABORT if there is a timeout
2. First, writes ABORT record to stable storage.
3. First, writes COMMIT record to stable storage.
4. Write COMPLETE record when all msgs confirmed.

Cohort i (i=2,3, …, n)


q_i
Commit_Request msg received
Agreed msg sent to Coordinator[1]
Commit_Request msg received
Abort msg sent to Coordinator
w_i
a_i
Abort msg received from Coordinator
Commit msg received from Coordinator[2]
c_i

1. First, write UNDO/REDO logs on stable storage.
2. Writes COMPLETE record; releases locks

Site Failures

Who Fails	At what point	Actions on recovery
Coordinator	before writing Commit	Send Abort messages
Coordinator	after writing Commit but before writing Complete	Send Commit messages
Coordinator	after writing Complete	None.
Cohort	before writing Undo/Redo	None. Abort will occur.
Cohort	after writing Undo/Redo	Wait for message from Coordinator.

پیچیدگی زمانی

۲ مرحله

سوال: چرا در حالی که الگوریتم تفاهم در شرایط خطای پروسس به f + 1 مرحله نیاز داشت، پروتکل 2PC صرفا به دو مرحله نیاز دارد؟

پاسخ: شرط پایان متفاوت است (weak termination)

پیچیدگی پیامی

در بدترین حالت $2(n-1)$ پیام در صورتی که خطا رخ ندهد.

MODULE *TwoPhase*

CONSTANT RM The set of resource managers

VARIABLES

$rmState$, $rmState[r]$ is the state of resource manager r.

$tmState$, The state of the transaction manager.

$tmPrepared$, The set of RMs from which the TM has received "Prepared" messages.

$msgs$

$$Messages \triangleq [type : \{\text{"Prepared"}\}, rm : RM] \cup [type : \{\text{"Commit"}, \text{"Abort"}\}]$$

$$\begin{aligned} TPTypeOK \triangleq\ & \land rmState \in [RM \rightarrow \{\text{"working"}, \text{"prepared"}, \text{"committed"}, \text{"aborted"}, \text{"Failed"}\}] \\ & \land tmState \in \{\text{"init"}, \text{"done"}\} \\ & \land tmPrepared \subseteq RM \\ & \land msgs \subseteq Messages \end{aligned}$$

$$\begin{aligned} TPInit \triangleq\ & \land rmState = [r \in RM \mapsto \text{"working"}] \\ & \land tmState = \text{"init"} \\ & \land tmPrepared = \{\} \\ & \land msgs = \{\} \end{aligned}$$

$TMRcvPrepared(r) \triangleq$
$\land\ tmState = \text{“init”}$
$\land\ [type \mapsto \text{“Prepared”},\ rm \mapsto r] \in msgs$
$\land\ tmPrepared' = tmPrepared \cup \{r\}$
$\land\ \text{UNCHANGED}\ \langle rmState,\ tmState,\ msgs \rangle$

$TMCommit \triangleq$
$\land\ tmState = \text{“init”}$
$\land\ tmPrepared = RM$
$\land\ tmState' = \text{“done”}$
$\land\ msgs' = msgs \cup \{[type \mapsto \text{“Commit”}]\}$
$\land\ \text{UNCHANGED}\ \langle rmState,\ tmPrepared \rangle$

$TMAbort \triangleq$
$\land\ tmState = \text{“init”}$
$\land\ tmState' = \text{“done”}$
$\land\ msgs' = msgs \cup \{[type \mapsto \text{“Abort”}]\}$
$\land\ \text{UNCHANGED}\ \langle rmState,\ tmPrepared \rangle$

$TMFailed \triangleq$
$\land\ tmState' = \text{“Failed”}$

$$\begin{array}{l}
RMPrepare(r) \triangleq \\
\quad \land\ rmState[r] = \text{"working"} \\
\quad \land\ rmState' = [rmState \ \text{EXCEPT}\ ![r] = \text{"prepared"}] \\
\quad \land\ msgs' = msgs \cup \{[type \mapsto \text{"Prepared"},\ rm \mapsto r]\} \\
\quad \land\ \text{UNCHANGED}\ \langle tmState,\ tmPrepared \rangle
\end{array}$$

$$\begin{array}{l}
RMChooseToAbort(r) \triangleq \\
\quad \land\ rmState[r] = \text{"working"} \\
\quad \land\ rmState' = [rmState \ \text{EXCEPT}\ ![r] = \text{"aborted"}] \\
\quad \land\ \text{UNCHANGED}\ \langle tmState,\ tmPrepared,\ msgs \rangle
\end{array}$$

$$\begin{array}{l}
RMRcvCommitMsg(r) \triangleq \\
\quad \land\ [type \mapsto \text{"Commit"}] \in msgs \\
\quad \land\ rmState' = [rmState \ \text{EXCEPT}\ ![r] = \text{"committed"}] \\
\quad \land\ \text{UNCHANGED}\ \langle tmState,\ tmPrepared,\ msgs \rangle
\end{array}$$

$$\begin{array}{l}
RMRcvAbortMsg(r) \triangleq \\
\quad \land\ [type \mapsto \text{"Abort"}] \in msgs \\
\quad \land\ rmState' = [rmState \ \text{EXCEPT}\ ![r] = \text{"aborted"}] \\
\quad \land\ \text{UNCHANGED}\ \langle tmState,\ tmPrepared,\ msgs \rangle
\end{array}$$

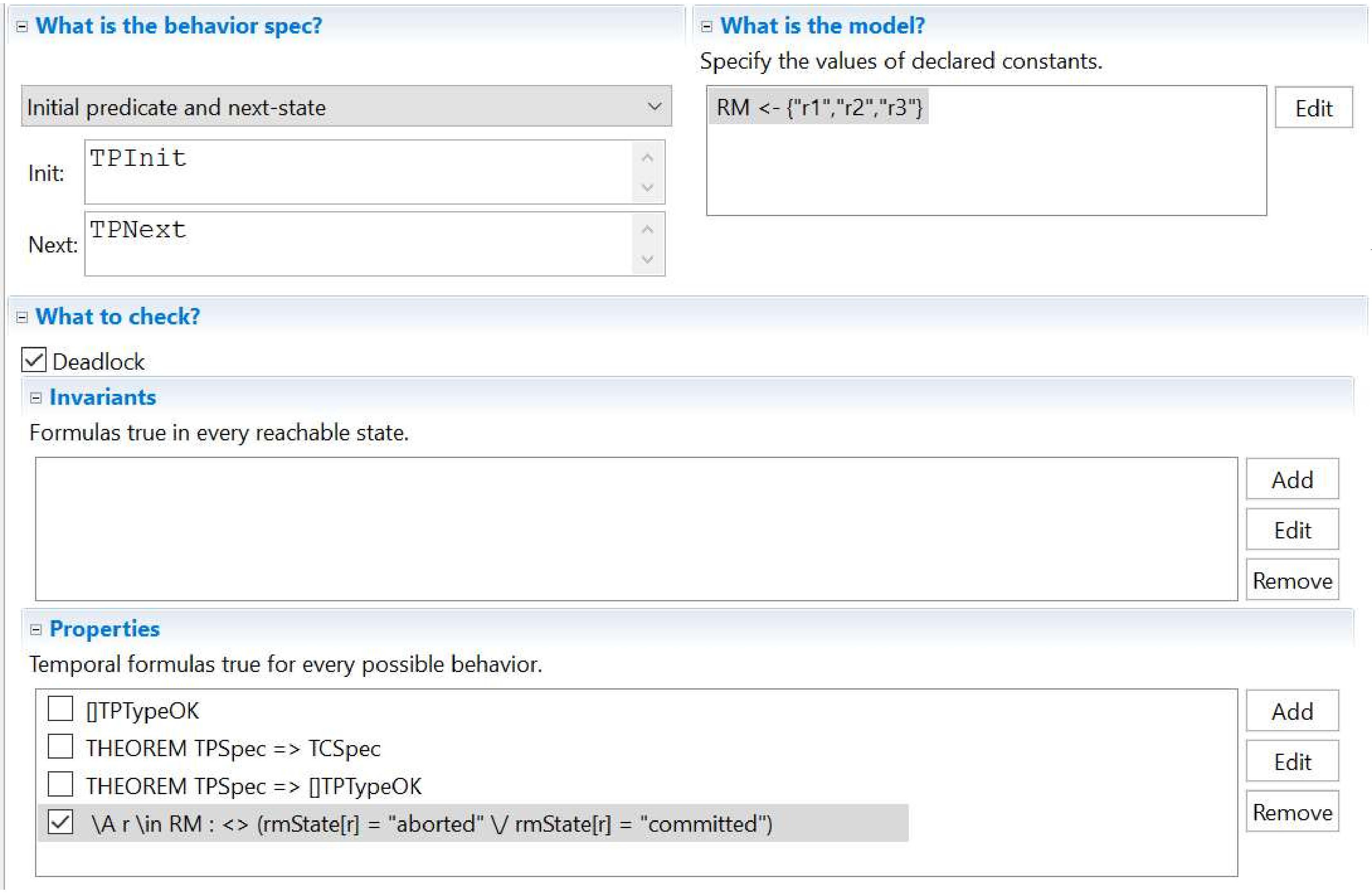
$$
\begin{array}{l}
\mathit{TPNext} \triangleq \\
\quad \lor\ \mathit{TMCommit} \lor \mathit{TMAbort} \\
\quad \lor\ \exists\, r \in \mathit{RM} : \\
\qquad\quad \mathit{TMRcvPrepared}(r) \lor \mathit{RMPrepare}(r) \lor \mathit{RMChooseToAbort}(r) \\
\qquad\qquad \lor\ \mathit{RMRcvCommitMsg}(r) \lor \mathit{RMRcvAbortMsg}(r)
\end{array}
$$

$$\mathit{TPSpec} \triangleq \mathit{TPInit} \land \Box[\mathit{TPNext}]_{\langle \mathit{rmState},\, \mathit{tmState},\, \mathit{tmPrepared},\, \mathit{msgs} \rangle}$$

THEOREM $\mathit{TPSpec} \Rightarrow \Box\, \mathit{TPTypeOK}$

INSTANCE $\mathit{TCommit}$

THEOREM $\mathit{TPSpec} \Rightarrow \mathit{TCSpec}$



What is the behavior spec?

Initial predicate and next-state

Init: `TPInit`

Next: `TPNext`

What is the model?

Specify the values of declared constants.

RM <- {"r1","r2","r3"}

What to check?

[✓] Deadlock

Invariants

Formulas true in every reachable state.

Properties

Temporal formulas true for every possible behavior.

- [ ] []TPTypeOK
- [ ] THEOREM TPSpec => TCSpec
- [ ] THEOREM TPSpec => []TPTypeOK
- [✓] \A r \in RM : <> (rmState[r] = "aborted" \/ rmState[r] = "committed")

***ThreePhaseCommit* algorithm, first three rounds:**

Round 1: All processes except for 1 send their initial values to process 1, and any process whose initial value is 0 decides 0. Process 1 collects all these values, plus its own initial value, into a vector. If all positions in this vector are filled in with 1s, then process 1 becomes *ready* but does not yet decide. Otherwise—that is, if there is some position that contains 0 or else some position that is not filled in (because no message was received from the corresponding process)—process 1 decides 0.

Round 2: If process 1 has decided 0, then it broadcasts *decide*(0). If not, then process 1 broadcasts *ready*. Any process that receives *decide*(0) decides 0. Any process that receives *ready* becomes *ready*. Process 1 decides 1 if it has not already decided.

Round 3: If process 1 has decided 1, it broadcasts *decide*(1). Any process that receives *decide*(1) decides 1.

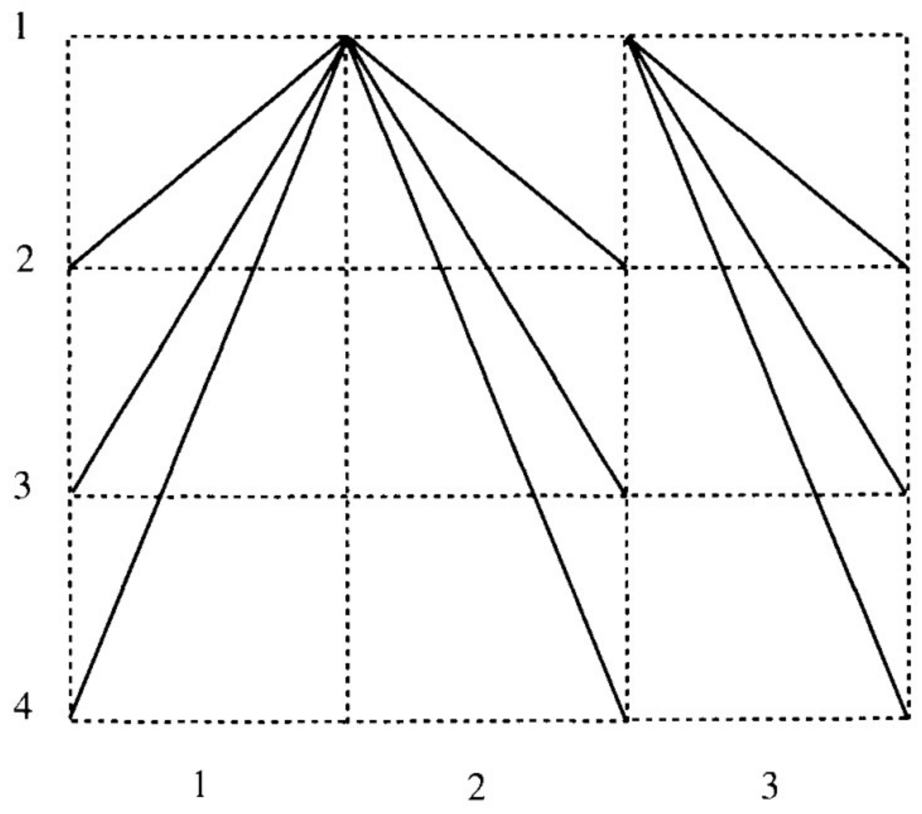
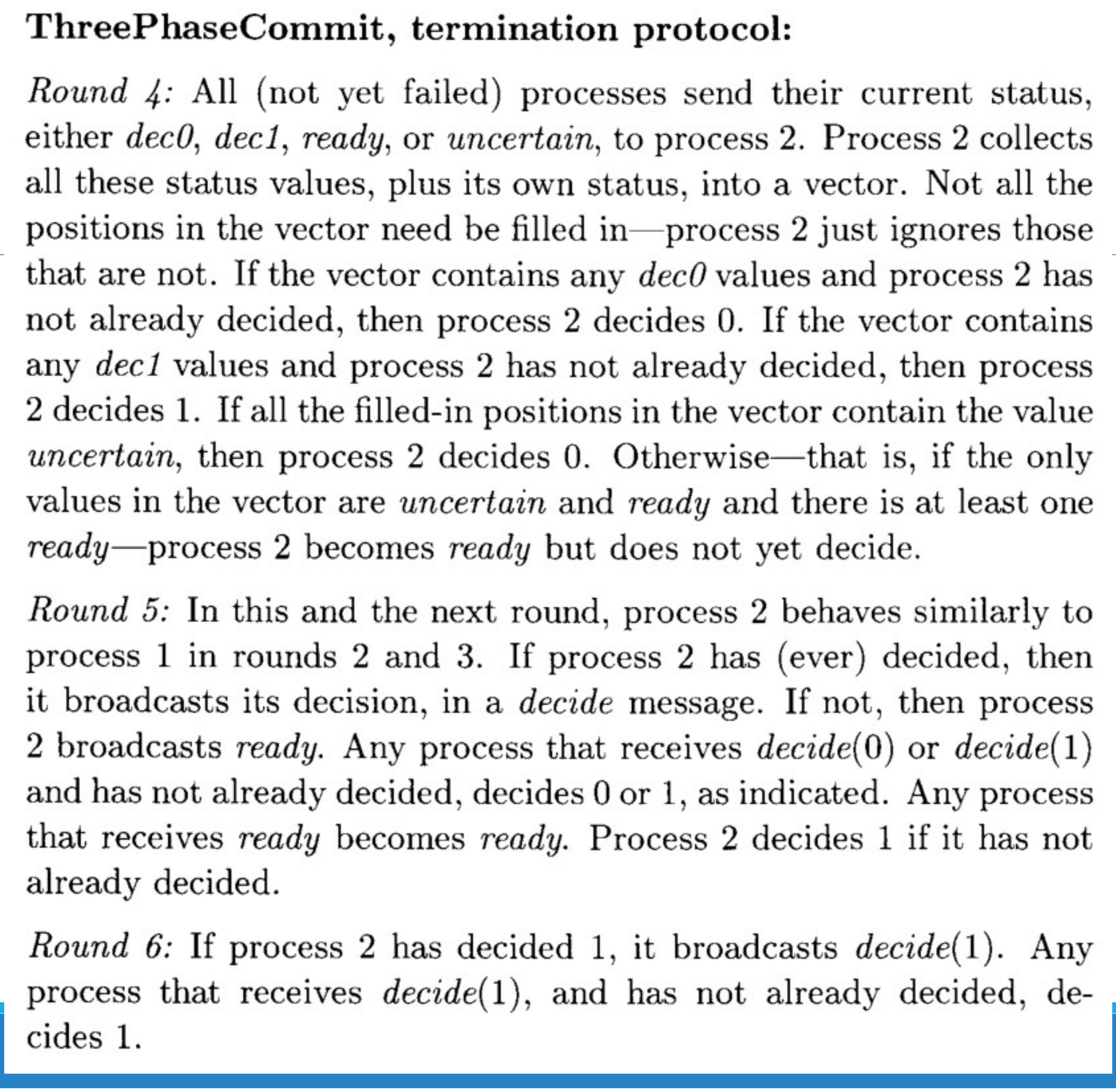


Figure 7.5: Communication pattern in *ThreePhaseCommit*.



ThreePhaseCommit, termination protocol:

Round 4: All (not yet failed) processes send their current status, either *dec0*, *dec1*, *ready*, or *uncertain*, to process 2. Process 2 collects all these status values, plus its own status, into a vector. Not all the positions in the vector need be filled in—process 2 just ignores those that are not. If the vector contains any *dec0* values and process 2 has not already decided, then process 2 decides 0. If the vector contains any *dec1* values and process 2 has not already decided, then process 2 decides 1. If all the filled-in positions in the vector contain the value *uncertain*, then process 2 decides 0. Otherwise—that is, if the only values in the vector are *uncertain* and *ready* and there is at least one *ready*—process 2 becomes *ready* but does not yet decide.

Round 5: In this and the next round, process 2 behaves similarly to process 1 in rounds 2 and 3. If process 2 has (ever) decided, then it broadcasts its decision, in a *decide* message. If not, then process 2 broadcasts *ready*. Any process that receives *decide*(0) or *decide*(1) and has not already decided, decides 0 or 1, as indicated. Any process that receives *ready* becomes *ready*. Process 2 decides 1 if it has not already decided.

Round 6: If process 2 has decided 1, it broadcasts *decide*(1). Any process that receives *decide*(1), and has not already decided, decides 1.

- ❑Agreement
- ❑Validity
- ❑Weak Termination
- ❑Strong Termination

در صورتی که پروسس ۱ دچار خطا نشود برقرار است.

تحلیل پیچیدگی

پیچیدگی زمانی:

3n مرحله در صورتی که n پروسس دچار خطا شود.

الگوریتم stopping agreement با حدود n مرحله به جواب می رسد.

سوال: چرا کامیت سه مرحله ای؟

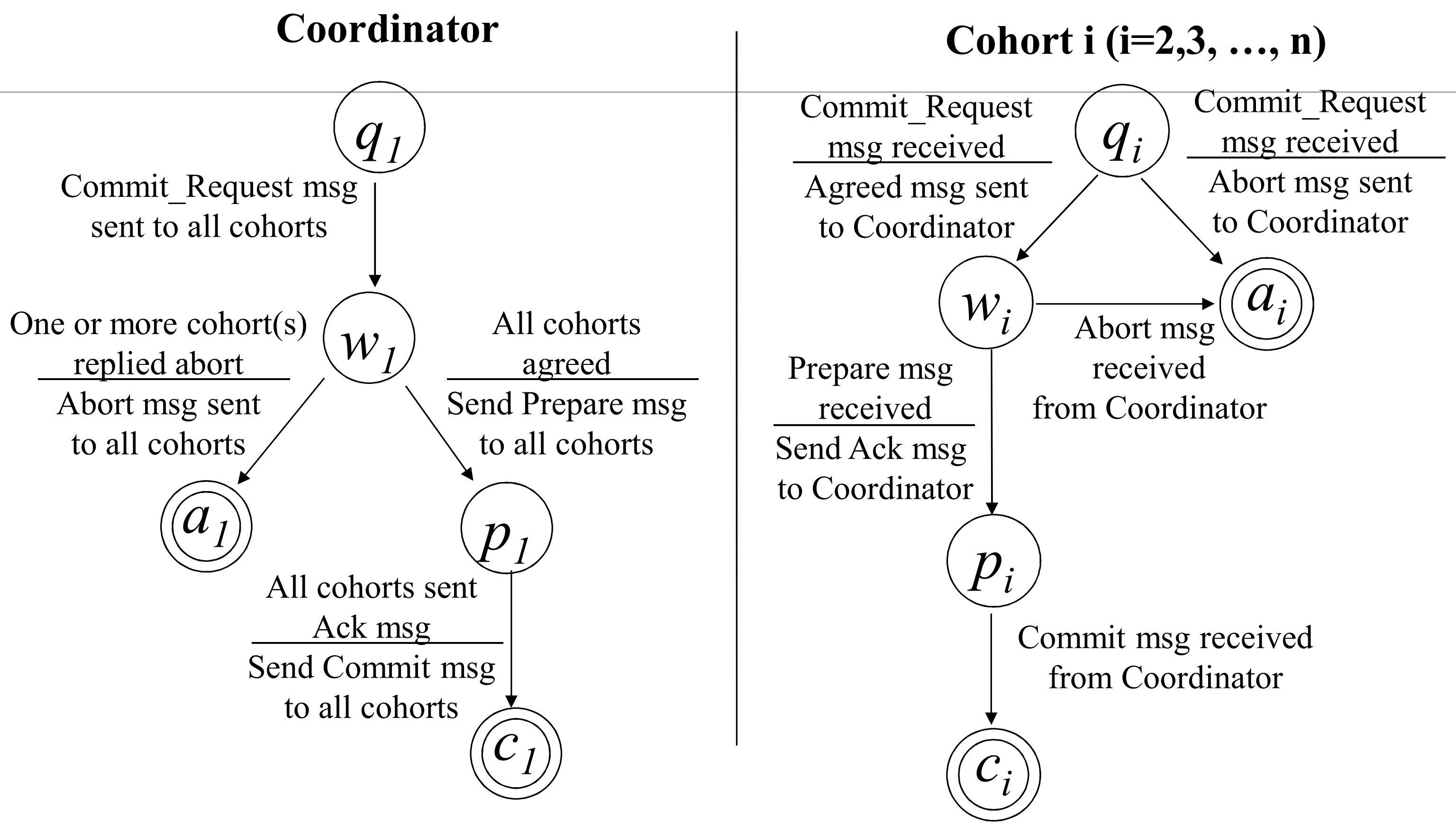
پاسخ: در شرایط نرمال، الگوریتم در سه مرحله کار می کند که بسیار سریع تر خواهد بود.

پیچیدگی پیامی:

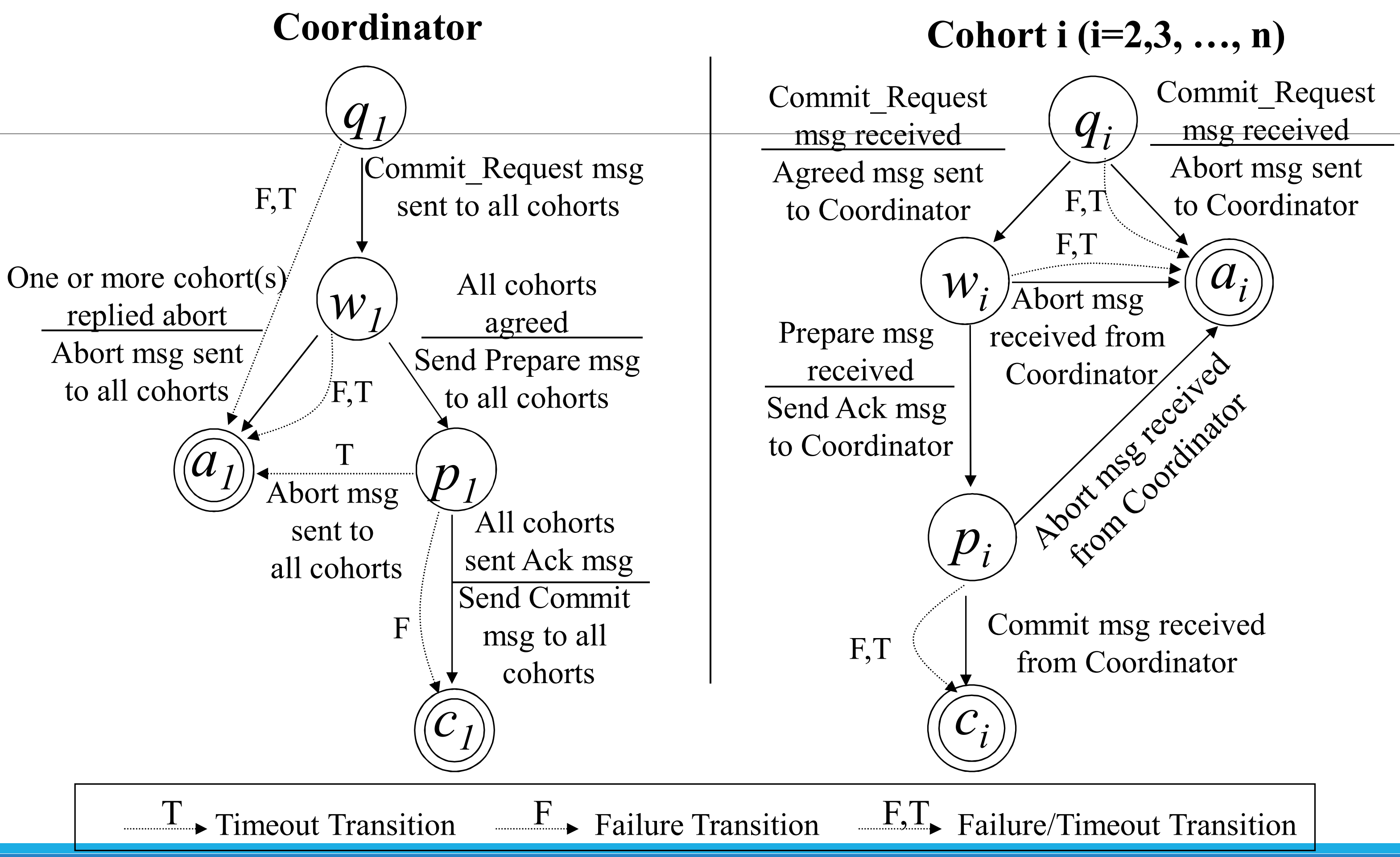
O(n)

قضیه: هر الگوریتم که مساله کامیت را حل کند، حداقل به 2 n – 2 پیام در صورت عدم بروز خطا و شرایطی که همه با ۱ آغاز کنند، نیاز دارد.

3-phase Commit Protocol


Coordinator
Cohort i (i=2,3, …, n)
q_1
Commit_Request msg sent to all cohorts
w_1
One or more cohort(s) replied abort / Abort msg sent to all cohorts
All cohorts agreed / Send Prepare msg to all cohorts
a_1
p_1
All cohorts sent Ack msg / Send Commit msg to all cohorts
c_1
q_i
Commit_Request msg received / Agreed msg sent to Coordinator
Commit_Request msg received / Abort msg sent to Coordinator
w_i
a_i
Abort msg received from Coordinator
Prepare msg received / Send Ack msg to Coordinator
p_i
Commit msg received from Coordinator
c_i

Timeout and Failure Transitions


Coordinator
Cohort i (i=2,3, …, n)
q_1
Commit_Request msg sent to all cohorts
F,T
w_1
One or more cohort(s) replied abort
Abort msg sent to all cohorts
All cohorts agreed
Send Prepare msg to all cohorts
F,T
a_1
T
Abort msg sent to all cohorts
p_1
All cohorts sent Ack msg
Send Commit msg to all cohorts
F
c_1
q_i
Commit_Request msg received
Agreed msg sent to Coordinator
Commit_Request msg received
Abort msg sent to Coordinator
F,T
F,T
w_i
a_i
Abort msg received from Coordinator
Prepare msg received
Send Ack msg to Coordinator
p_i
Abort msg received from Coordinator
F,T
Commit msg received from Coordinator
c_i
T Timeout Transition
F Failure Transition
F,T Failure/Timeout Transition

منبع

فصل ۷ کتاب Lynch