Distributed Systems
Non-Blocking Atomic Commit

Prof R. Guerraoui
Distributed Programming Laboratory
Non-Blocking Atomic Commit: An Agreement Problem
Transactions (Gray)

• A transaction is an atomic program describing a sequence of accesses to shared and distributed information

• A transaction can be terminated either by committing or aborting
Transactions

beginTransaction
  Pierre.credit(1.000.000)
  Paul.debit(1.000.000)
outcome := commitTransaction
if (outcome = abort) then ...
ACID properties

**Atomicity**: a transaction either performs entirely or none at all

**Consistency**: a transaction transforms a consistent state into another consistent state

**Isolation**: a transaction appears to be executed in isolation

**Durability**: the effects of a transaction that commits are permanent
The Consistency Contract

- Atomicity
- Isolation
- Durability

(system)

Consistency (local)

(programmer)

Consistency (global)
Distributed Transaction

A

abort-commit

B

C

abort-commit

abort-commit
Non-Blocking Atomic Commit

- As in consensus, every process has an initial value 0 (no) or 1 (yes) and must decide on a final value 0 (abort) or 1 (commit)
- The proposition means the ability to commit the transaction
- The decision reflects the contract with the user
- Unlike consensus, the processes here seek to decide 1 but every process has a veto right
Non-Blocking Atomic Commit

**NBAC1. Agreement:** No two processes decide differently

**NBAC2. Termination:** Every correct process eventually decides

**NBAC3. Commit-Validity:** 1 can only be decided if all processes propose 1

**NBAC4. Abort-Validity:** 0 can only be decided if some process crashes or votes 0
Non-Blocking Atomic Commit

p1
propose(0)
decide(0)

propose(1)
decide(0)

p2
propose(0)
decide(0)

propose(0)
decide(0)

p3
Non-Blocking Atomic Commit

\begin{itemize}
  \item p1: propose(1)
  \item p2: propose(1)
  \item p3: propose(1)
\end{itemize}

\begin{itemize}
  \item decide(0-1)
\end{itemize}

\textit{crash}
2-Phase Commit

propose(1)

propose(1)

propose(1)

propose(1)

decide(1)

decide(1)

decide(1)

decide(1)
2-Phase Commit

propose(1) -> decide(0)
propose(1) -> decide(0)
propose(1) -> decide(0)

p1 -> crash -> p2

p3
2-Phase Commit

propose(1)

p1

propose(1)  crash

propose(1)

p2

propose(1)

p3
Non-Blocking Atomic Commit

**Events**

- Request: `<Propose, v>`
- Indication: `<Decide, v'>`

**Properties:**
- `NBAC1, NBAC2, NBAC3, NBAC4`
Algorithm (nbac)

**Implements:** nonBlockingAtomicCommit (nbac).

**Uses:**
- BestEffortBroadcast (beb).
- PerfectFailureDetector (P).
- UniformConsensus (uniCons).

**upon event** < Init > do

- prop := 1;
- delivered := ∅; correct := Π;
Algorithm (nbac – cont’d)

upon event < crash, pi > do
  correct := correct \ {pi}
upon event < Propose, v > do
  trigger < bebBroadcast, v>;
upon event <bebDeliver, pi, v> do
  delivered := delivered U {pi};
  prop := prop * v;
Algorithm (nbac – cont’d)

\[\textbf{upon event} \ \text{correct} \ \backslash \ \text{delivered} = \text{empty} \ \textbf{do}\]

\[\textbf{if} \ \text{correct} \neq \Pi \]

\[\text{prop} := 0;\]

\[\textbf{trigger} < \text{uncPropose, prop}>;\]

\[\textbf{upon event} < \text{uncDecide, decision}> \ \textbf{do}\]

\[\textbf{trigger} < \text{Decide, decision}>;\]
nbac with ucons

propose(1)  

p1

propose(1)  

p2

propose(1)  

p3

UCons(1,1)  

decide(1)
nbac with ucons

propose(1)

p1

propose(1)
crash

propose(1)

p2

propose(1)

UCons(0,0)

decide(0)

propose(1)

p3

UCons(0,0)

decide(0)
nbac with ucons

propose(1)
propose(1)
propose(1)

p1

p2

p3

decide(0-1)

decide(0-1)

UCons(0,0-1)

UCons(1,0-1)

crash
Non-Blocking Atomic Commit

• Do we need perfect failure detector $P$?
  
  • 1. $<>P$ is not enough
  
  • 2. $P$ is needed if one process can crash
Non-Blocking Atomic Commit

- Do we need perfect failure detector P?
  
  1. \(<>P\) is not enough
  
  2. P is needed if one process can crash
1. Run 1

- **p1**
  - propose(0)
  - crash

- **p2**
  - propose(1)
  - decide(0)

- **p3**
  - propose(1)
  - decide(0)
1. Run 2

```
propose(1)

p1  crash

propose(1)  decide(0)

p2

propose(1)  decide(0)

p3
```
1. Run 3

propose(1)

p1

propose(1)

decide(0)

p2

propose(1)

decide(0)

p3

<>P becomes P
Non-Blocking Atomic Commit

- Do we need perfect failure detector P?
  - 1. $<>P$ is not enough
  - 2. $P$ is needed if one process can crash
2. P is needed with one crash

p1: NBAC(1,1) → NBAC(1,0) → suspect(p2)
p2: NBAC(1,1) → crash

p3: NBAC(1,1) → NBAC(1,0) → suspect(p2)
History

- Atomic Commit (Eswaran/Gray 76 – Gray 78)
- NBAC (Skeen 81)
  - Complexity of Sync NBAC (DS 83)
- Async NBAC (Had 90 – Gue 95)
  - Fast Async NBAC (KD95, GLS95, GL06)
- FD NBAC (DFGHTK 04)
  - Optimal NBAC (GW17)