Recovery for $f = 1$

Parents of $p$
Messages previously sent to $p$ by its parents
SSN order

RSN order
Determinants of messages delivered by $p$
Children of $p$
Recovery for \( f = 1 \)

Parents of \( p \):

- Messages previously sent to \( p \) by its parents
  - SSN order

\( p \) SSN order

RSN order

Determinants of messages delivered by \( p \)

Children of \( p \):

- 2 3 4
- 5 6 2 8

Who is my next parent?

What is the next message from each parent?
Recovery for $f = 1$

Parents of $p$:
- Messages previously sent to $p$ by its parents
- SSN order

Determinants of messages delivered by parents
- RSN order
- Determinants of messages delivered by $p$'s children

What is the next message from each parent? Who is my next parent?

Family-Based Logging

Each $p$ maintains $D_p \equiv \{#m : p \in \text{Depend}(m)\}$ in volatile memory.

On sending a message $m'$:
- Adds $m'$ to volatile send log
- Piggybacks on messages to $q$
  - All determinants $#m \in D_p$ s.t. $|\text{Log}(m)|_p \leq f \land (q \notin \text{Log}(m)_p)$

On receiving a message $m'$:
- Adds to $D_p$ any new determinant piggybacked on $m'$
- Adds $#m'$ to $D_p$
- Updates its estimate of $|\text{Log}(m)|_p$ for all determinants $#m \in D_p$

Estimating $\text{Log}(m)$ and $|\text{Log}(m)|$:

Each process $p$ maintains estimates of $\text{Log}(m)_p$ and $|\text{Log}(m)|_p$.

$p$ piggybacks $#m$ on $m'$ to $q$ if $|\text{Log}(m)|_p \leq f \land (q \notin \text{Log}(m)_p)$

Estimating $\text{Log}(m)$ and $|\text{Log}(m)|$:

Each process $p$ maintains estimates of $\text{Log}(m)_p$ and $|\text{Log}(m)|_p$.

$p$ piggybacks $#m$ on $m'$ to $q$ if $|\text{Log}(m)|_p \leq f \land (q \notin \text{Log}(m)_p)$

• How can $p$ estimate $\text{Log}(m)_p$ and $|\text{Log}(m)|_p$?
**Estimating Log(m) and |Log(m)|**

Each process $p$ maintains estimates of $\text{Log}(m)_p$ and $|\text{Log}(m)|_p$.

$p$ piggybacks $\#m$ on $m'$ to $q$ if

$|\text{Log}(m)|_p \leq f \land (q \notin \text{Log}(m)_p)$

- How can $p$ estimate $\text{Log}(m)_p$ and $|\text{Log}(m)|_p$?
- How accurate should these estimates be?
  - Inaccurate estimates cause useless piggybacking.
  - Keeping estimates accurate requires extra piggybacking.

---

**The Idea**

Because $\forall m : (\neg \text{stable}(m) \Rightarrow (\text{Depend}(m) \subseteq \text{Log}(m)))$

we can approximate $\text{Log}(m)$ from below with:

and then use vector clocks to track $\text{Depend}(m)$!

$\text{Log}(m) = \begin{cases} 
\text{Depend}(m) & \text{if } |\text{Depend}(m)| \leq f \\
\text{Any set } S : |S| > f & \text{otherwise}
\end{cases}$

---

**Dependency Vectors**

Dependency Vector (DV): vector clock that tracks causal dependencies between message delivery events.

$\text{deliver}_p(m) \rightarrow \text{deliver}_q(m') \equiv 
DV_p(\text{deliver}_p(m))[p] \leq DV_q(\text{deliver}_q(m'))[p]$
Use WDVs to determine if \( p \in \text{Log}(m) \):

\[
p \in \text{Depend}(m) \land |\text{Depend}(m)| \leq f \Rightarrow \\
WDV_p[m.\text{dest}] \geq m.\text{rsn}
\]

\[
WDV_p[m.\text{dest}] \geq m.\text{rsn} \Rightarrow \\
p \in \text{Depend}(m)
\]

Each \( p \) keeps a Dependency Matrix (DM_p)

Given \( #m = \langle u, s, i, H, \|s\rangle \),

\[
\text{DM}_p = \begin{bmatrix}
p & q & i & \text{dest} \\
\text{source} & s & & \\
\text{rsn} & & & \\
\text{sn} & & & \\
\end{bmatrix}
\]

\[
\text{Log}(m)_p = \{p, q, s\} 
\]
Egida
- Transparent
  - seamless integration with applications
- Extensible
  - easily handles new sources of non-determinism
- Flexible
  - allows to select best protocol for application
- Smart
  - don’t want to implement 300 protocols
- Powerful
  - a “microscope” to understand rollback recovery

Rollback Recovery Protocols: A Success Story?
- Over 300 papers in the area
- Relatively few implementations
- Why?
  - Performance issues not understood
  - Hard to integrate recovery protocol with application
  - One size doesn’t fit all

The Unifying Theme
- All rollback recovery protocols enforce the no-orphans consistency condition
- The challenge is handling non-determinism
  - a process may execute non-deterministic events
  - a process may interact with other processes or with the environment and generate dependencies on these events
- Characterize a protocol according to how it handles non-determinism
  - identify relevant events
  - specify which actions to take when event occurs
Relevant Events

- Non-deterministic events
  - message delivery, file read, clock read, lock acquire
- Failure-detection events
  - time-outs, message delivery
- Internal dependency-generating events
  - message send, file write, lock acquire
- External dependency-generating events
  - output to printer or screen, file write
- Checkpointing events
  - time-outs, explicit instruction, message delivery

Event handlers invoked on relevant events

Library of modules
- implement core functionalities
- checkpointing, creating determinants, logging, piggybacking, detecting orphans, restarting a faulty process
- provide basic services
- (stable storage, failure detection, etc)
- single interface–multiple implementations

Specification language to select desired modules and corresponding implementations

Synthesize protocol automatically from specification

The Architecture

Integration with MPICH

MPICH
- 2 layers architecture
- upper layer exports MPI to application
- lower layer performs data transfer using application-specific libraries (e.g. P4)

Application
- Modifications to MPICH:
  - Replace calls to P4 with call to Egida’s API
- Modifications to P4:
  - Handle socket-level errors
  - Allow reconnection of recovering process
- Modification to applications:
  - NONE

Application
- Modifications to MPICH:
  - Replace calls to P4 with call to Egida’s API
- Modifications to P4:
  - Handle socket-level errors
  - Allow reconnection of recovering process
- Modification to applications:
  - NONE