HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

CÁC HỆ THỐNG PHÂN TÁN VÀ ỨNG DỤNG

Chương 6: Tính chịu lỗi

Nội dung

1. Mở đầu
2. Khả năng phục hồi của các tiến trình
3. Trao đổi thông tin client-server tin cậy
4. Trao đổi thông tin nhóm tin cậy
5. Commit phân tán
6. Phục hồi

1. Mở đầu

1.1. Các khái niệm cơ bản

1.2. Các mô hình lỗi

1.3. Che giấu lỗi bởi sự dư thừa

1.1. Khái niệm cơ bản

- Khả năng chịu lỗi liên quan đến khái niệm *hệ thống đáng tin cậy:*
  - Tính sẵn sang (Availability)
  - Tính tin cậy (Reliability)
  - Tính an toàn (Safety)
  - Khả năng bảo trì được (Maintainability)
- Các khái niệm cơ bản:
  - *Fail/Fault*
  - *Fault Tolerance*
  - *Transient Faults*
  - *Intermittent Faults*
  - *Permanent Faults*

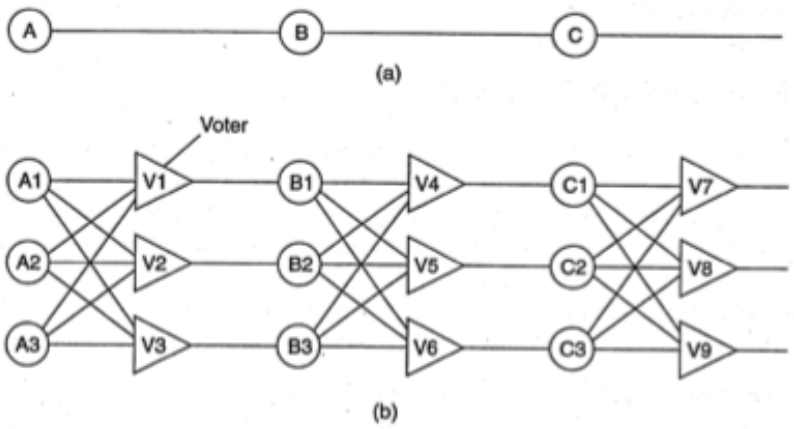
1.2. Các mô hình lỗi

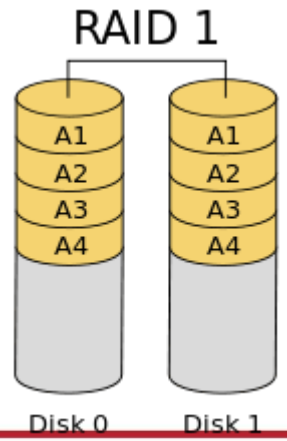
- Các kiểu lỗi khác nhau

Kiểu lỗi	Mô tả
Crash failure	A server halts, but is working correctly until it halts
Omission failure	Aserver fails to respond to incoming requests
Receive omission	A server falls to receive incoming messages
Send omission	A server falls to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure	A server's response is incorrect
Value failure	The value of the response is wrong
State transition failure	The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times
Fail-stop failure	A server stops producing output and its halting can be detected by other systems
Fail-silent failure	Another process may incorrectly conclude that a server has halted
Fail-safe	A server produces random output which is recognized by other processes as plain junk

1.3. Che giấu lỗi với sự dư thừa

- 3 kiểu
  - ***Dư thừa thông tin***
  - ***Dư thừa thời gian***
  - ***Dư thừa thiết bị vật lý***
- VD1: Triple Modular Redundancy (***TMR***)
- VD2: RAID 1





2. Khả năng phục hồi của các tiến trình

2.1. Vấn đề thiết kế

2.2. Che giấu lỗi và sao lưu

2.3. Thống nhất trong hệ thống lỗi

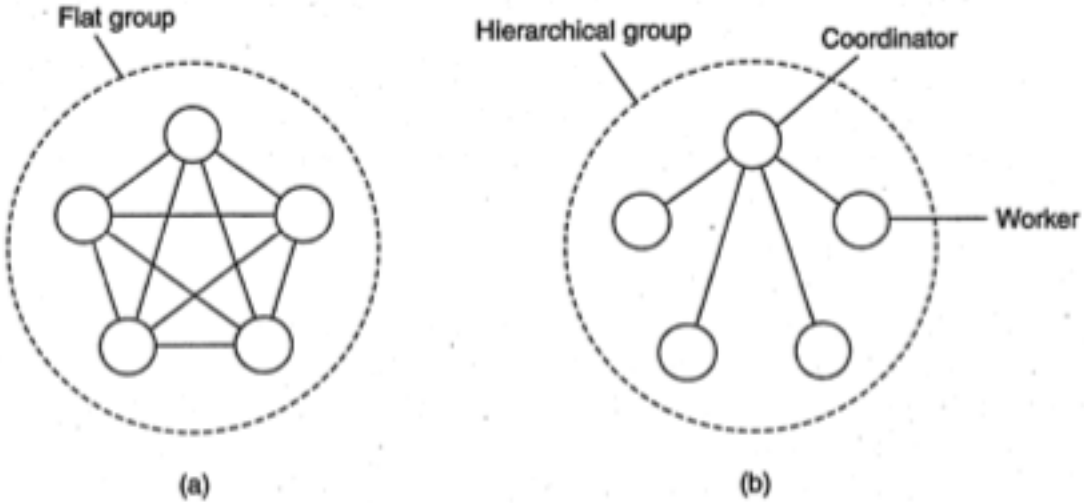
2.4. Phát hiện lỗi

2.1. Các vấn đề thiết kế (1/3)

- ***Nhóm tiến trình***
  - Hướng tiếp cận: Tổ chức nhiều tiến trình giống nhau vào cùng 1 nhóm
  - Đặc điểm chính: các thông điệp gửi đến nhóm sẽ được gửi cho tất cả các tiến trình trong nhóm
  - Tính cơ động: tạo, hủy nhóm, thêm, bớt tiến trình

2.1. Các vấn đề thiết kế (2/3)

- ***Nhóm phẳng*** và ***Nhóm phân cấp***

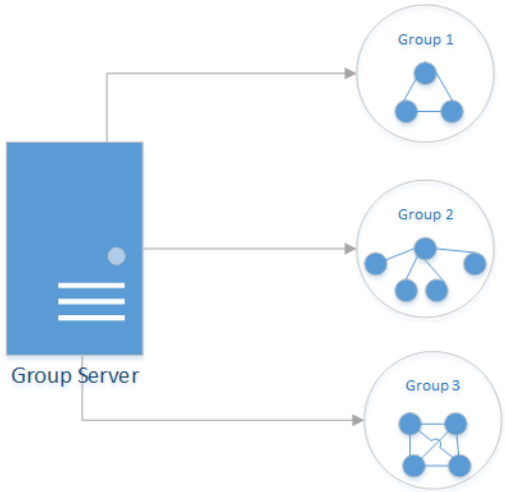


  - So sánh

	Ưu điểm	Nhược điểm
Nhóm phẳng	Vai trò ngang hàng Không có điểm đơn chịu lỗi Nhóm vẫn hoạt động nếu có 1 vài tiến trình lỗi	Khó ra quyết định
Nhóm phân cấp	Dễ dàng ra quyết định	Điểm đơn chịu lỗi

2.1. Các vấn đề thiết kế (3/3)

- ***Server tập trung***
  - Hướng tiếp cận
    - Các yêu cầu gửi đến server
    - Quản lý csdl
    - Quản lý các thành viên
  - Nhược điểm
    - *Điểm đơn chịu lỗi*



- ***Kiểu phân tán***

Hướng tiếp cận: ngang hàng, các thành viên chủ động liên lạc với nhau

Nhược điểm:

- *Lỗi dừng* (fail stop): → không phù hợp
- Rời hoặc gia nhập nhóm phải đồng bộ các thông điệp đã được gửi

- ***Vấn đề:***

Nhiều máy trong nhóm hỏng cùng lúc? →giao thức xây dựng lại nhóm

2.2. Che giấu lỗi và Sao lưu

- ***Giao thức dựa trên primary (Primary-based protocols)***
- Tổ chức nhóm các tiến trình theo kiểu phân cấp
- Nếu coordinator (primary) hỏng thì sử dụng các thuật toán bầu chọn để bầu lại

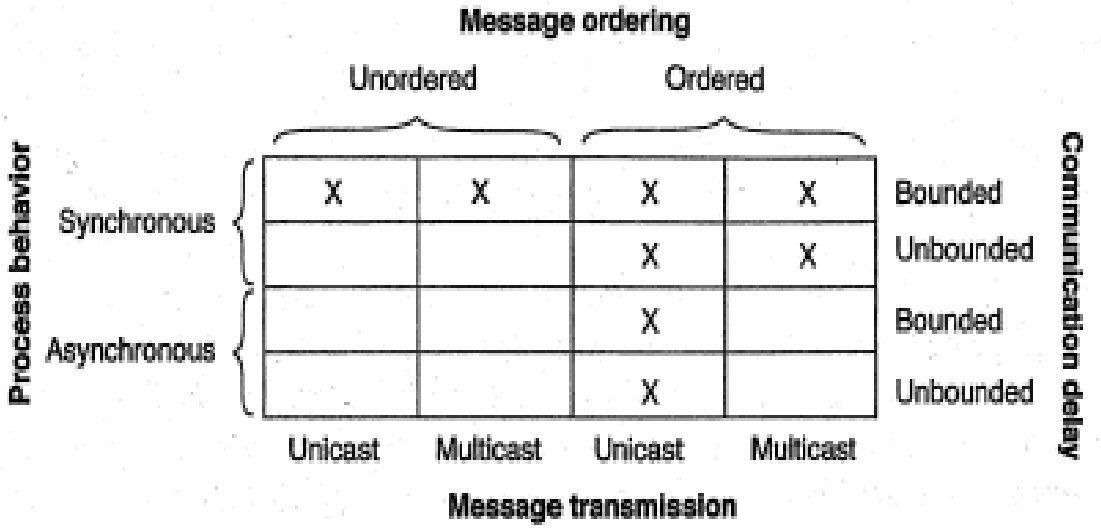
- ***Giao thức ghi trên các bản sao (Replicated-write protocols)***
- Sử dụng sao lưu tích cực (*active replication*) hoặc sao lưu dựa trên định số tối thiểu (*quorum-based protocols*)
- Tổ chức các tiến trình giống nhau vào 1 nhóm phẳng
- Gọi là *'k fault tolerant'* nếu hệ thống có thể hoạt động tốt với *k* nút hỏng

2.3. Đồng thuận trong các hệ thống có lỗi (1/3)

- ***Các trường hợp khác nhau***

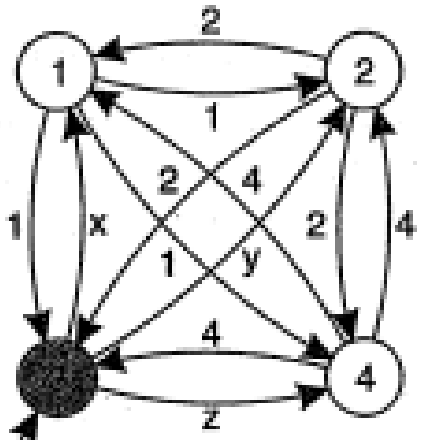
1. Đồng bộ vs Không đồng bộ
2. Giao tiếp có ngưỡng thời gian hay không
3. Việc nhận thông điệp theo thứ tự hay không
4. Truyền thông điệp theo kiểu unicasting hay multicasting

- ***Các trường hợp mà đồng thuận phân tán có thể thực hiện:***



2.3. Đồng thuận trong các hệ thống có lỗi (2/3)

- ***Giải thuật của Lamport: Byzantine agreement***
- ***VD:*** $N = 4$ and $k = 1$



(a)

1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

2.3. Đồng thuận trong các hệ thống có lỗi (3/3)

- ***Lamport et al. (1982)*** đã chứng minh được rằng sự đồng thuật của hệ thống có thể đạt được nếu có 2k+1 tiến trình hoạt động đúng với tổng (3k+1) tiến trình, trong đó có k tiến trình lỗi.

(hơn 2/3 tiến trình chạy tốt)

- ***Fisher et al. (1985)*** *chứng minh rằng* các thông điệp không được gửi đến trong khoảng thời gian giới hạn biết trước thì sẽ không có khả năng thực hiện đồng thuận được

2.4. Phát hiện lỗi

- 2 cơ chế: ***Active process và Passive Process***
- Dựa vào *timeouts:* (cần xác định rõ xem có phải lỗi mạng không)
- Cách kiểm tra:
  - gossiping
  - probe
  - liên tục trao đổi thông tin với các nút hàng xóm

3. Trao đổi thông tin client-server tin cậy

3.1. Trao đổi thông tin điểm-điểm

3.2. Lỗi trong RPC

3.1. Trao đổi thông tin điểm-điểm

- Sử dụng giao thức giao vận đáng tin cậy (TCP)
  - TCP che giấu *lỗi bỏ xót* bằng cách sử dụng những thông điệp báo nhận → lỗi đã được che giấu với người dùng

  - Tuy nhiên, với *lỗi sụp đổ* (crash failures) không thể che giấu vì kết nối TCP bị gián đoạn

    -> người dùng được thông báo thông qua các exception

    -> HPT thiết lập kết nối mới

3.2. Các lỗi xảy đến với RPC (1/5)

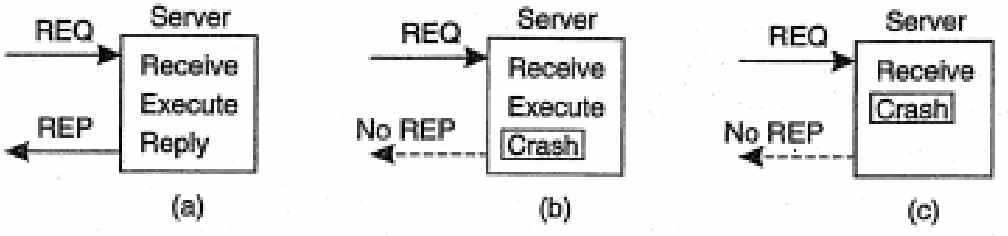
- **RPC (Remote Procedure Calls)** đảm bảo tính trong suốt trong cơ chế thực hiện
- **Các lỗi có thể xảy đến**:
  - Client không xác định được vị trí Server
  - Thông điệp yêu cầu gửi từ client đến server bị mất
  - Client bị treo/hỏng sau khi gửi 1 yêu cầu
  - Server bị treo/hỏng sau khi nhận được request
  - Thông điệp trả lời từ server về đến client bị mất

3.2. Các lỗi xảy đến với RPC (2/5)

- Lỗi Client không tìm được Server, vd: client không thể tìm được server, hoặc tất cả các server đã hỏng
  -> Giải pháp: sinh ra **Exception**
  Nhược điểm:
  - không phải tất cả các ngôn ngữ đều hỗ trợ Exception
  - sử dụng Exception → giảm tính trong suốt
- Thông điệp gửi đi bị mất: phát hiện bằng cách thiết lập timer
  - vượt quá timeouts mà không nhận được reply/ack → gửi lại thông điệp
  - Quá nhiều thông điệp bị mất → client từ bỏ và kết luận rằng server hỏng, từ đó quay lại lỗi "không thể xác định vị trí máy chủ"
  - Khi yêu cầu không bị mất: để cho các server phát hiện và xử lý với việc truyền lại.

3.2. Các lỗi xảy đến với RPC (3/5)

- Server hỏng



(a) Normal Case (b) Crash after execution (c) Crash before execution

Khó phân biệt giữa (b) và (c)

- (b) hệ thống gửi báo lỗi lại cho client
- (c) cần truyền lại yêu cầu

3 nguyên lý cho servers:

- At least once
- At most once
- Exactly once

4 nguyên lý cho client

- Không bao giờ gửi lại yêu cầu
- Luôn luôn gửi lại yêu cầu
- Chỉ gửi lại khi không nhận được ACK
- Chỉ gửi lại khi nhận được ACK

3.2. Các lỗi xảy đến với RPC (4/5)

- Server hỏng (tiếp)

8 sự kết hợp nguyên lý hoạt động của Client và Server

- 3 sự kiện: M (gửi thông điệp ACK), P (thực thi in văn bản), C (hỏng)

1. *M -> P -> C*
2. *M -> C (-> P)*
3. *P -> M -> C*
4. *P -> C –(> M)*
5. *C (-> P -> M)*
6. *C (-> M -> P)*

Client	Server					
	Strategy M → P			Strategy P → M		
Reissue strategy	MPC	MC(P)	C(MP)	PMC	PC(M)	C(PM)
Always	DUP	OK	OK	DUP	DUP	OK
Never	OK	ZERO	ZERO	OK	OK	ZERO
Only when ACKed	DUP	OK	ZERO	DUP	OK	ZERO
Only when not ACKed	OK	ZERO	OK	OK	DUP	OK

OK = Text is printed once
DUP = Text is printed twice
ZERO = Text is not printed at all

Kết luận:

- Khả năng máy chủ bị treo thay đổi bản chất của RPC và phân biệt các hệ thống đơn vi xử lý với HPT
- Trong trường hợp trước đây, máy chủ bị hỏng cũng dẫn tới máy khách hỏng

3.2. Các lỗi xảy đến với RPC (5/5)

- Mất thông điệp trả lời

Sau một khoảng timeouts thì phải gửi lại yêu cầu.

Vấn đề: client không biết là mất yêu cầu hay mất thông điệp trả lời của server.

Nếu gửi lại → Vấn đề idempotent

- Client hỏng
- **→ vấn đề orphans**

Khó khăn:

- Tốn tài nguyên CPU
- Khóa file và làm treo những tài nguyên khác
- Client khởi động lại, gửi lại yêu cầu, và nhận được trả lời từ yêu cầu trước → nhầm lẫn

- **Giải pháp:**
  - Diệt orphan (Orphan extermination)
  - Đầu thai (Reincarnation)
  - Gentle Reincarnation
  - Expiration

4. Trao đổi thông tin theo nhóm tin cậy

4.1. Dịch vụ Reliable Multicasting

4.2. Khả năng mở rộng của Reliable Multicasting

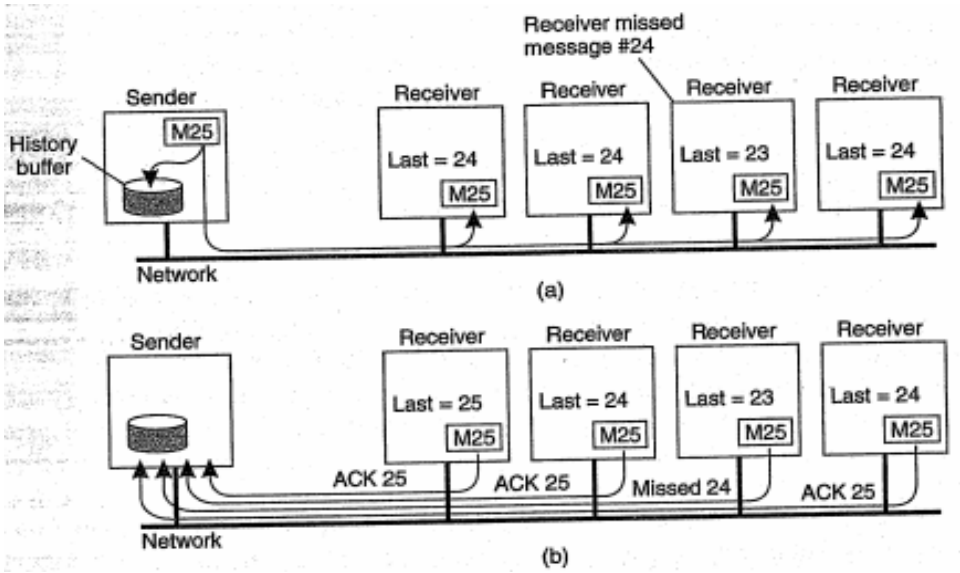
4.3. Atomic Multicast

4.1. Reliable Multicasting

- **Multicasting:** 1 thông điệp gửi đến 1 nhóm các tiến trình, thì cần phải được gửi đến mọi tiến trình trong nhóm.
- **Nếu có các tiến trình lỗi**: multicasting là đáng tin cậy nếu tất cả các tiến trình không hỏng đều phải nhận được thông điệp

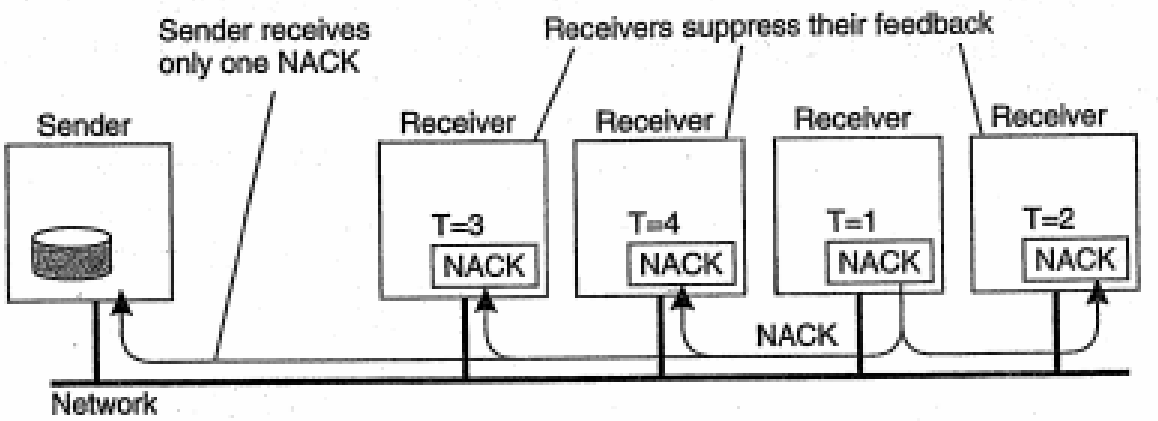
(a) Truyền thông điệp

(b) Phản hồi



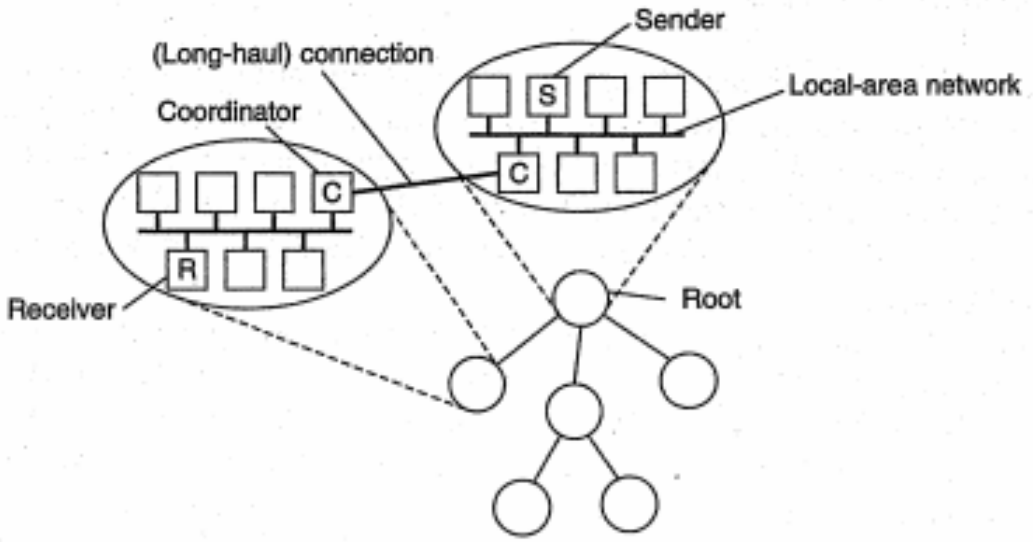
4.2. Khả năng mở rộng với Reliable Multicasting (1/2)

- **Vấn đề với reliable multicast scheme** là không hỗ trợ nếu có quá nhiều tiến trình nhận.
  - Nút gửi sẽ bị quá tải với quá nhiều ACK nhận được
  - Với mạng WAN, các nút nhận nằm rải rác ở khắp nơi
- **Cơ chế điều khiển phản hồi không phân cấp (Nonhierarchical feedback control)**
  - Ý tưởng: giảm số thông điệp phản hồi
  - Giải pháp: mỗi Receiver nào mà nhận được thông điệp lỗi sẽ gửi multicast NACK cho các nút khác nữa → giúp xóa hết các NACK của các node khác. Như vậy Sender chỉ nhận được 1 NACK thôi
  - **Vấn đề:**
    - Đồng bộ việc gửi NACK
    - Gửi multicasting NACK sẽ làm ảnh hưởng các node nhận tốt



4.2. Khả năng mở rộng với Reliable Multicasting (2/2)

- **Cơ chế điều khiển phản hồi không phân cấp (Hierarchical feedback control)**
  - Chia thành các nhóm
  - Mỗi groups sẽ có Coordinator.
  - Cooridinator chuyên đảm nhận việc chuyển tiếp thông điệp cho các thành viên trong nhóm và sau đó kiểm soát các thông điệp phản hồi.

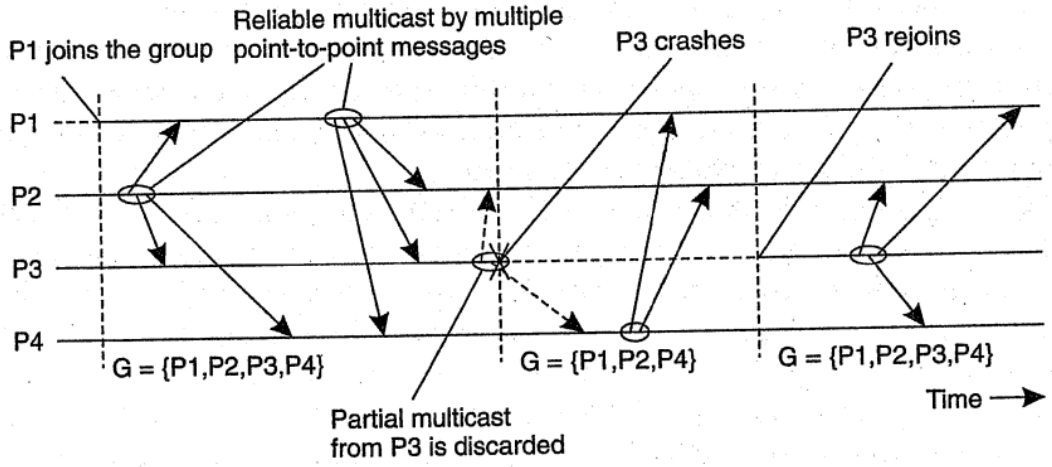


4.3. Atomic Multicast (1)

- **Vấn đề Atomic multicast:**
  - Một HPT với nhiều replicas
  - Các cập nhật được thực hiện dựa trên multicasting
  - Tuy nhiên ở giữa chừng có 1 replica bị treo/hỏng → mất thời gian để reboot/sửa chữa/phục hồi. Sau khi phục hồi thì nó đã bị lỡ các cập nhật trong thời gian nó không hoạt động.
  - →Không thống nhất được dữ liệu với các replicas khác.
  - →Cần có **Atomic Multicast**: tức là hoặc là phổ biến được đến hết các nút trong nhóm, hoặc là không cho bất kỳ node nào.

4.3. Atomic Multicast (2)

- Mỗi tiến trình có danh sách các thành viên trong nhóm: Group view
- Nếu có nút rời hoặc tham gia nhóm → View change → gửi thông điệp VC (view change)
- Vậy vấn đề: VC hay thông điệp m đến trước?
- => Cần đảm bảo tất cả nhận được m trước khi nhận được vc, hoặc không node nào nhận được m.
- Mô hình Reliable Multicast được coi là Virtual Synchrony nếu nó đảm bảo tất cả các thông điệp multicast hoàn thành trước khi có view change.



4.3. Atomic Multicast (3)

- **Thứ tự các thông điệp**
  - Unordered multicasts

Process P1	Process P2	Process P3
sends m1	receives m1	receives m2
sends m2	receives m2	receives m1

Sample of three communicating processes in the same group -> the ordering of events per process is shown along the vertical axis

  - FIFO-ordered multicasts

Process P1	Process P2	Process P3	Process P3
sends m1	receives m1	receives m3	receives m3
sends m2	receives m3	receives m1	receives m4
	receives m2	receives m2	
	receives m4	receives m4	

Sample of four processes in the same group with two different senders, and a possible delivery order of messages under FIFO-ordered multicasting

  - Causally-ordered multicasts
  - Totally-ordered multicasts

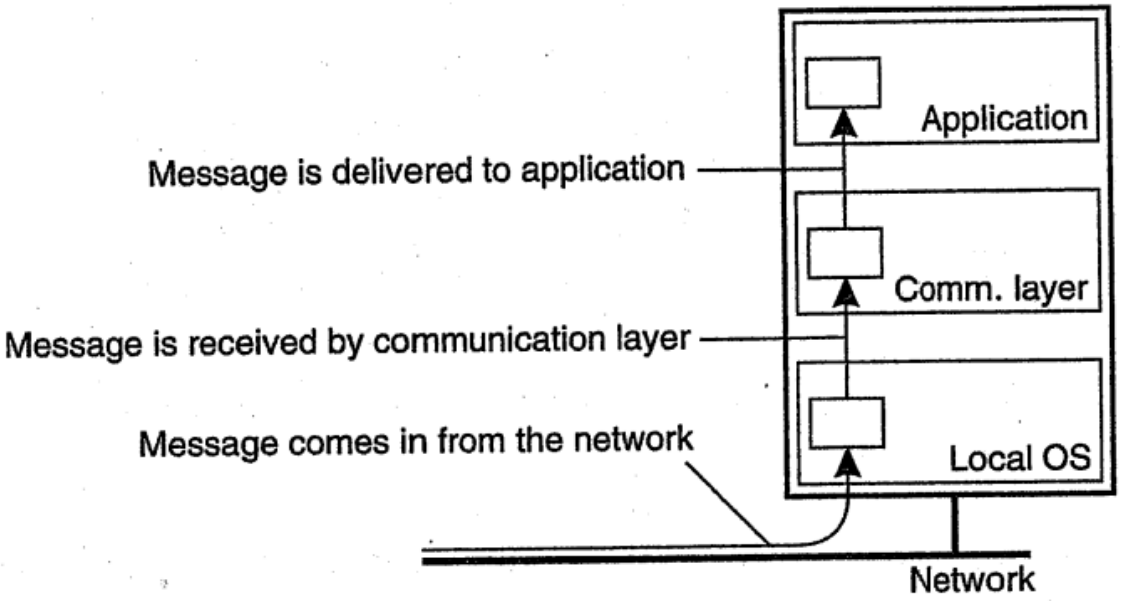
4.3. Atomic Multicast (4)

- **Triển khai Virtual Synchrony**

Mục đích: đảm bảo tất cả các tiến trình (không lỗi) trong view G nhận được thông điệp m trước khi có view change.

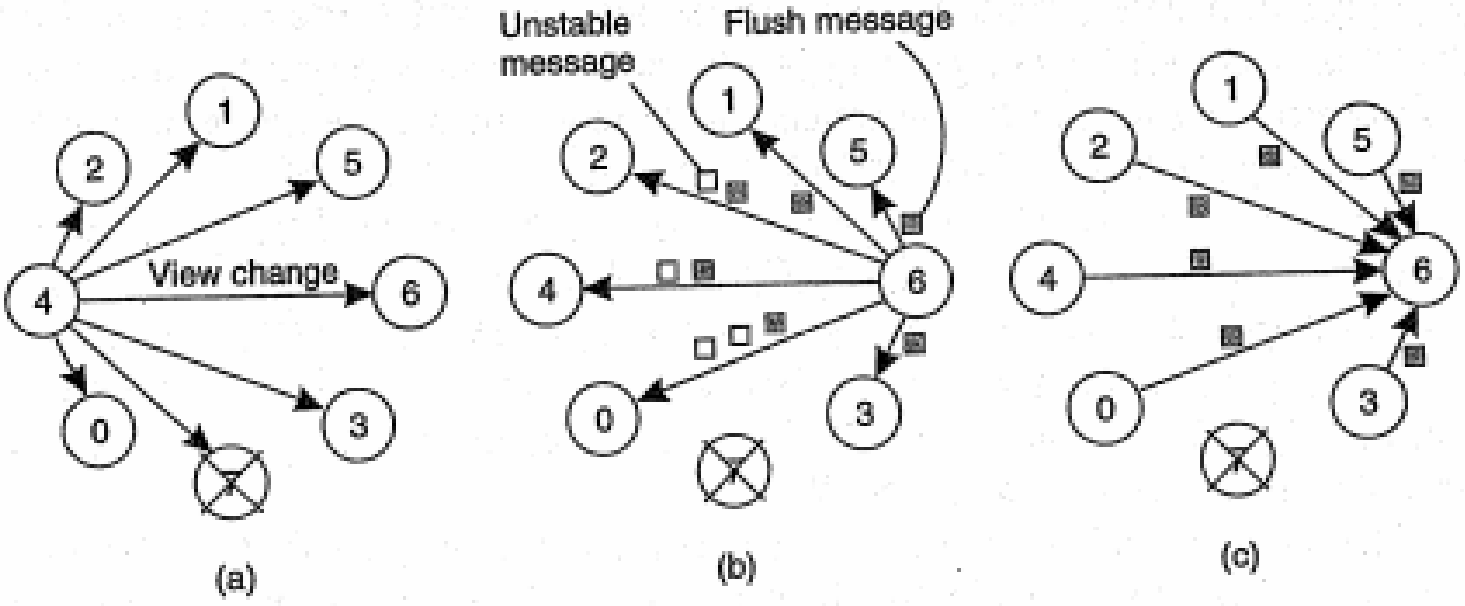
Giải pháp: cho phép mỗi tiến trình trong G giữ m đến khi nào tất cả các thành viên khác trong G đã nhận được m (thông điệp ổn định - stable message)

Chỉ những *thông điệp ổn định* mới được cho phép nhận (delivered: tức là gửi lên trên tầng application).



4.3. Atomic Multicast (5)

- **Triển khai Virtual Synchrony**
- Minh họa:
  a) P4 phát hiện P7 hỏng, gửi thông điệp báo VC
  b) P6 gửi cho tất cả những thông điệp chưa ổn định, sau đó đánh dấu nó là ổn định, kết thúc bởi thông điệp flush.
  c) Khi đã nhận đầy đủ các thông điệp flush từ các tiến trình khác → P6 cài view mới.



5. Distributed Commit

5.1. Commit 2 pha

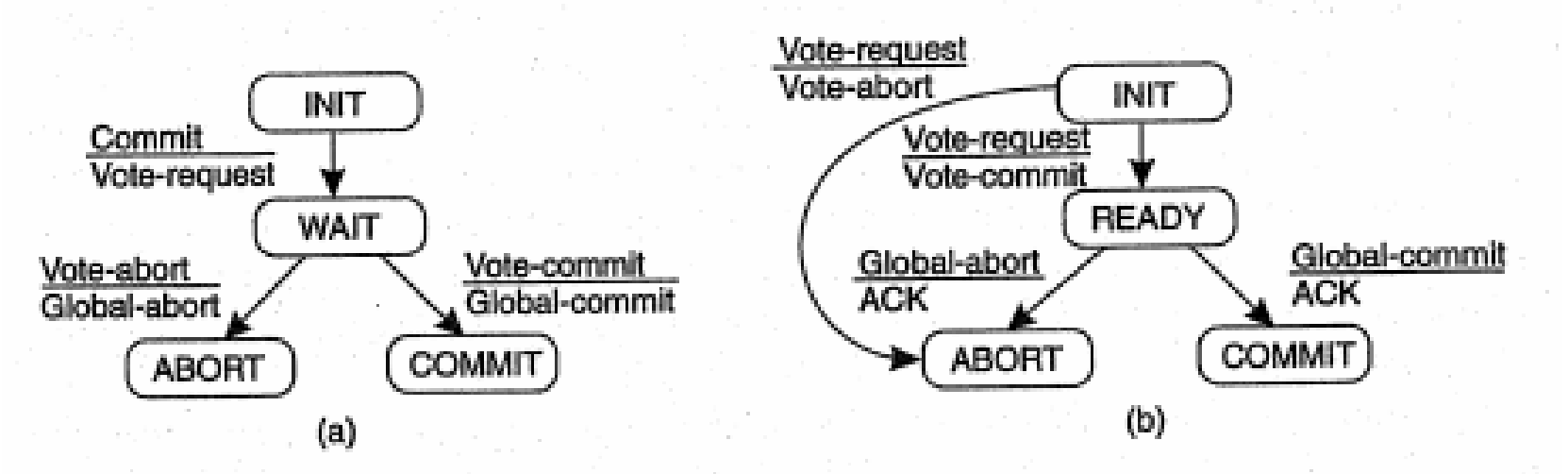
5.2. Commit 3 pha

Distributed Commit

- **Mục đích:** để tất cả các tiến trình trong nhóm cùng thực hiện 1 thao tác gì đó, hoặc không tiến trình nào thực hiện thao tác gì.
- Với giao thức **one-phase commit**: Tiến trình coordinator sẽ gửi yêu cầu thực hiện thao tác cho tất cả các tiến trình thành viên.
- **Nhược điểm:** trường hợp nếu 1 thành viên không thực hiện được thao tác đó → không làm cách nào để báo lại được cho coordinator.

- => **Cần có kiểu giao thức khác!**

5.1. Two-Phase Commit - 2PC (1/5)



Coordinator **Participant**

5.1. Two-Phase Commit - 2PC (2/5)

- **Giải pháp của các tiến trình thành viên:**
  - Dùng timeouts khi đang ở trạng thái READY
  - Khi chờ lâu quá timeouts: tham khảo thông tin của tiến trình Q khác:

Trạng thái của Q	P thực hiện
COMMIT	Chuyển sang COMMIT
ABORT	Chuyển sang ABORT
INIT	Chuyển sang ABORT
READY	Liên lạc tiến trình khác

5.1. Two-Phase Commit - 2PC (3/5)

- Pseudocode thực thi của tiến trình thành viên:

```
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
      write VOTE_ABORT to local log;
      exit;
}
if participant votes COMMIT {
      write VOTE_COMMIT to local log;
      send VOTE_COMMIT to coordinator;
      wait for DECISION from coordinator;
      if timeout {
            multicast DECISION_REQUEST to other participants;
            wait until DECISION is received; /* remain blocked */
            write DECISION to local log;
      }
      if DECISION == GLOBAL_COMMIT
            write GLOBAL_COMMIT to local log;
      else if DECISION == GLOBAL_ABORT
            write GLOBAL_ABORT to local log;
} else {
      write VOTE_ABORT to local log;
      send VOTE_ABORT to coordinator;
}
```

5.1. Two-Phase Commit - 2PC (4/5)

- Mã thực hiện của participant khi bị 1 tiến trình khác hỏi ý kiến (DECISION_REQUEST)

```
Actions for handling decision requests: /* executed by separate thread */

    while true {
        wait until any incoming DECISION_REQUEST is received; /* remain blocked */
        read most recently recorded STATE from the local log;
        if STATE == GLOBAL_COMMIT
            send GLOBAL_COMMIT to requesting participant;
        else if STATE == INIT or STATE == GLOBAL_ABORT
            send GLOBAL_ABORT to requesting participant;
        else
            skip; /* participant remains blocked */
    }
```

5.1. Two-Phase Commit - 2PC (5/5)

- **Giải pháp của Coordinator**

```
write START_2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
        wait for any incoming vote;
        if timeout {
                write GLOBAL_ABORT to local log;
                multicast GLOBAL_ABORT to all participants;
                exit;
        }
        record vote;
}
if all participants sent VOTE_COMMIT and coordinator votes COMMIT {
        write GLOBAL_COMMIT to local log;
        multicast GLOBAL_COMMIT to all participants;
} else {
        write GLOBAL_ABORT to local log;
        multicast GLOBAL_ABORT to all participants;
}
```

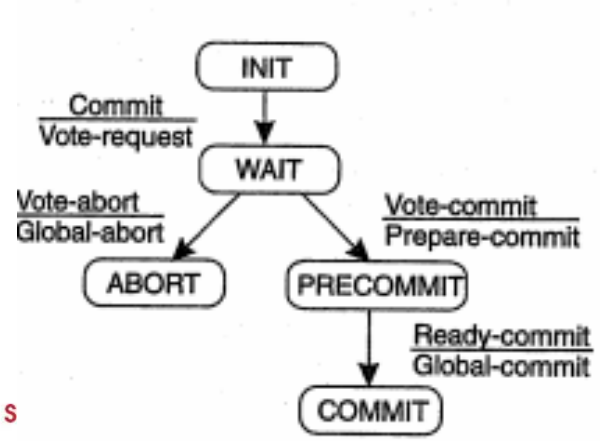
5.2. Three-Phase Commit (1/2)

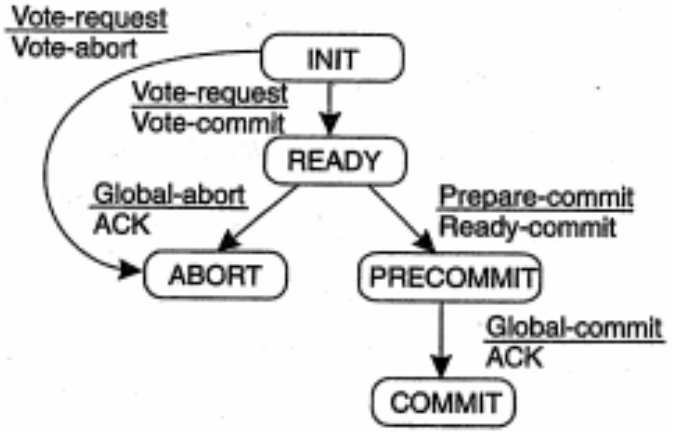
Vấn đề cuar 2PC: Trường hợp Coordinator bất ngờ hỏng, các participant cứ phải chờ vô thời hạn cho đến khi coordinator phục hồi.

⇒ Sử dụng 3PC

3PC tuân thủ 2 nguyên tắc:

- Không có state nào được chuyển trực tiếp đến COMMIT hay ABORT (mà bắt buộc phải chờ 1 thông điệp nào đó mới đc chuyển)
- Không có state nào mà ko thể ra quyết định cuối, và từ đó có thể thực hiện được 1 chuyển trạng thái đến COMMIT





5.2. Three-Phase Commit (2/2)

- Actions taken by Participant in different cases

State of Participant P	State of Participant Q	State of all other participants	Action
INT			VOTE_ABORT
READY	INT		VOTE_ABORT
READY	READY	READY	VOTE_ABORT
READY	PRECOMMIT	PRECOMMIT	VOTE_COMMIT
PRECOMMIT	READY	READY	VOTE_ABORT
PRECOMMIT	PRECOMMIT	PRECOMMIT	VOTE_COMMIT
PRECOMMIT	COMMIT	COMMIT	VOTE_COMMIT

- Actions taken by Coordinator in different cases

State of Coordinator	Action
WAIT	GLOBAL_ABORT
PRECOMMIT	GLOBAL_COMMIT

- Main difference with 2PC: if any participant is in READY state, no crashed process will recover to a state other than INT, ABORT or PRECOMMIT

6. Phục hồi

6.1. Mở đầu

6.2. Checkpointing

6.1. Mở đầu (1/2)

- **Hồi phục quay lui:**
  - trở về trạng thái trước (trước khi bị lỗi)
  - Cần thực hiện ghi lại các trạng thái trong quá trình hoạt động, gọi là các checkpoint.
  - VD: thực hiện gửi lại gói tin khi bị mất gói tin đối với các cơ chế truyền tin tin cậy
  - Vấn đề:
    - tốn kém tài nguyên hệ thống
    - Không đảm bảo được là lỗi đó sẽ không xảy ra nữa sau khi thực hiện quay lui
    - Những trạng thái nào chưa được thực hiện ghi lại checkpoint thì sẽ không phục hồi về đó được

- **Hồi phục tiến:**
  - Tiến lên trạng thái không còn bị lỗi nữa
  - Cần phải biết trước là lỗi nào sẽ xảy ra
  - VD: erasure correction: được thực hiện khi có lỗi mất gói tin, thì gói tin bị mất sẽ được phục hồi nhờ gói tin gửi tiếp theo đó (chứ không cần phải gửi lại)

6.1. Mở đầu (2/2)

- **Lưu trữ ổn định (stable storage)**

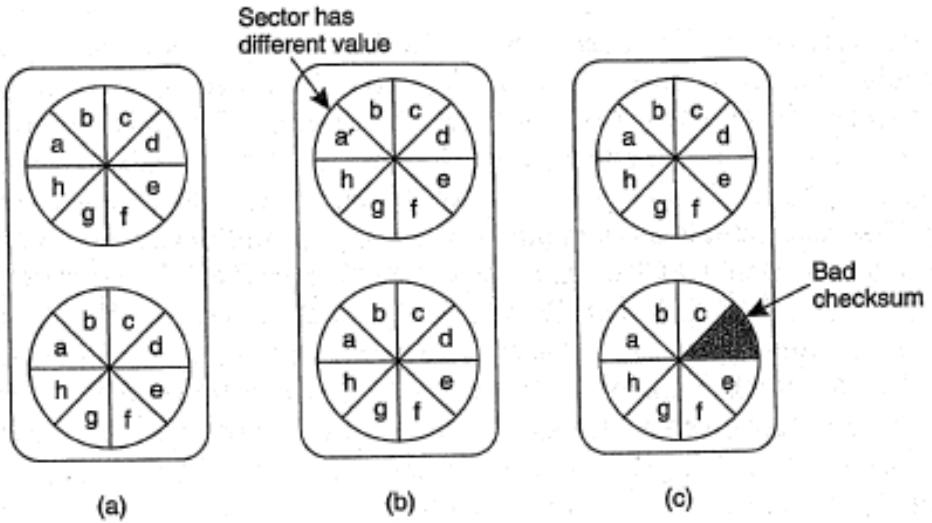
■ Các kiểu lưu trữ:

- RAM: mất thông tin khi mất điện
- Ổ đĩa: mất thông tin khi ổ đĩa bị hỏng
- Stable storage: cơ chế lưu trữ thông tin bền vững mãi mãi

(a) Stable storage

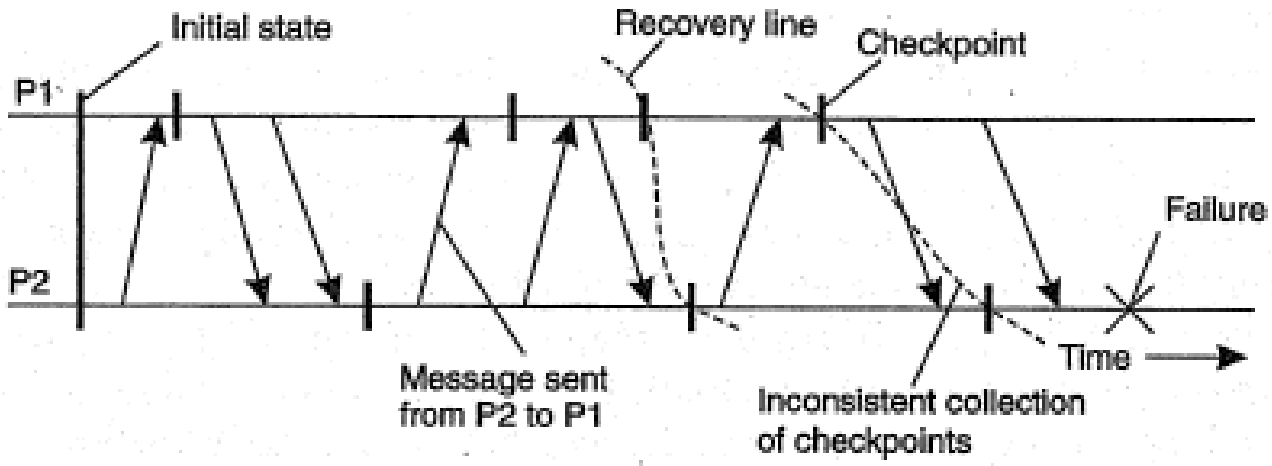
(b) Crash after drive 1 is updated

(c) Bad spot



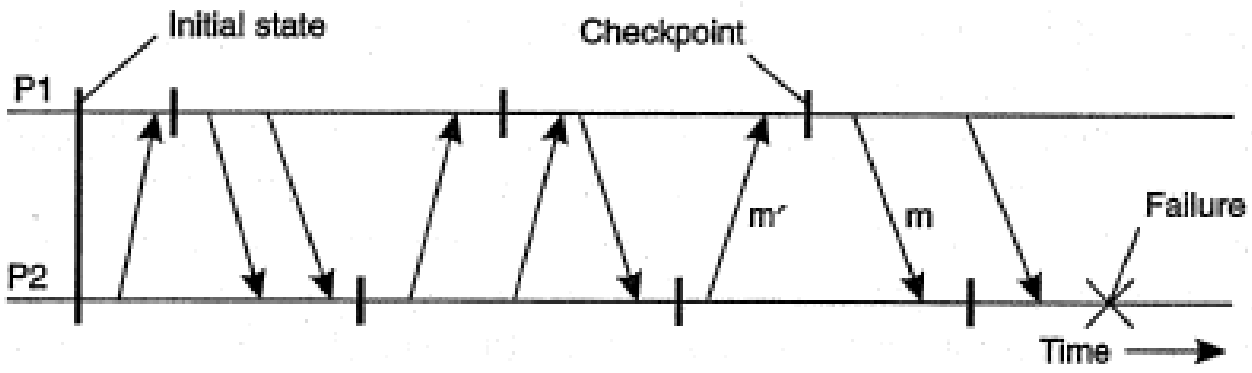
6.2. Checkpointing (1)

- **Phục hồi quay lui:** lưu lại các checkpoint -->trạng thái toàn cục (*global state* or *global snapshot*)
- **Distributed snapshot**: nếu một triến trình P đã lưu lại việc nhận 1 thông điệp, thì phải có 1 tiến trình Q lưu lại việc gửi thông điệp đó.
- Khi gặp lỗi, thì hệ thống sẽ quay lui lại *distributed snapshot* ngay trước đó → đường phục hồi (recovery line)



6.2. Checkpointing (2)

- thực hiện lưu checkpoint một cách độc lập giữa các tiến trình → hiệu ứng domino (domino effect)
- VD: (hình vẽ) không tìm được *consistent global state*
- → Thực hiện **đồng bộ hóa việc lưu checkpoint**, tuy nhiên rất tốn tài nguyên hệ thống



ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Câu hỏi?