LẬP TRÌNH ĐỒNG THỜI & PHÂN TÁN

BÀI 7:
BÀI TOÁN SẮP THỨ TỰ THÔNG ĐIỆP

Giảng viên: Lê Nguyễn Tuấn Thành
Email: thanhlnt@tlu.edu.vn

Tính không xác định (1)

- Các chương trình phân tán khó thiết kế và kiểm thử bởi tính chất không xác định của nó
  - **Nguyên nhân**: do thứ tự khác nhau của các thông điệp trong mỗi lần thực thi
- Một tính toán bất đồng bộ hoàn toàn không có bất kỳ giới hạn nào về thứ tự thông điệp
  - Cho phép tối đa sự đồng thời
- **Tuy nhiên**: KHÓ thiết kế những thuật toán cho thứ tự giao tiếp bất đồng bộ hoàn toàn
  - Do các thuật toán này phải tính đến tất cả thứ tự có thể có trong việc truyền thông điệp

Tính không xác định (2)

Mong muốn: kiểm soát tính chất không xác định của các chương trình phân tán

- Bằng cách kiểm soát các kiểu thứ tự thông điệp có thể có trong hệ thống phân tán

NỘI DUNG

- Thứ tự FIFO
- Thứ tự nhân quả
- Thứ tự đồng bộ
- Thứ tự toàn bộ cho thông điệp multicast
  - Thuật toán tập trung
  - Thuật toán của Lamport
  - Thuật toán của Skeen

Bài giảng có sử dụng hình vẽ trong cuốn sách "***Concurrent and Distributed Computing in Java***, *Vijay K. Garg, University of Texas, John Wiley & Sons, 2005*"

Thứ tự FIFO

FIFO ordering

Thứ tự FIFO

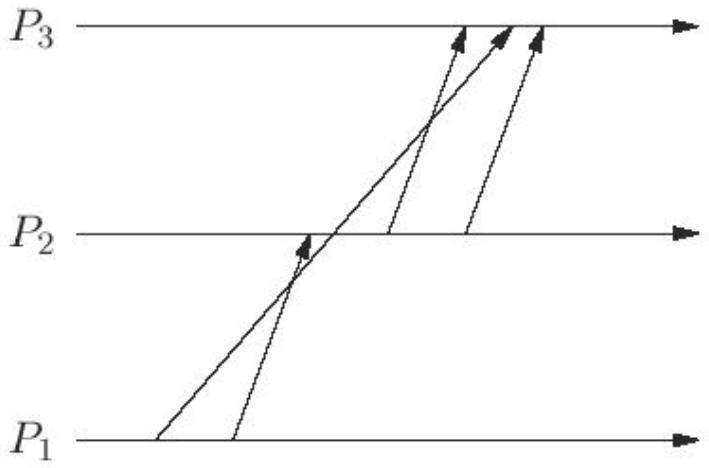
- Nhiều hệ thống phân tán giới hạn việc phân phối thông điệp theo thứ tự FIFO
  - Giúp đơn giản hoá thiết kế thuật toán
  - Ví dụ: chúng ta đã sử dụng giả thiết thứ tự FIFO trong thuật toán của Lamport cho bài toán truy cập tài nguyên chia sẻ
- **Tuy nhiên:** chương trình sẽ mất đi một vài tính chất đồng thời
  - Khi nhận được một thông điệp không tuân theo thứ tự FIFO, việc xử lý phải bị trì hoãn

Thứ tự FIFO
Định nghĩa

Hai thông điệp được gửi từ tiến trình P_i đến tiến trình P_j được nhận theo cùng một thứ tự như khi gửi đi

$$s_1 \prec s_2 \Rightarrow \neg(r_2 \prec r_1)$$

- $e \prec f$: **e xảy ra trước f <u>trong cùng một tiến trình</u>**
- $s_i \rightsquigarrow r_i$: **r_i** là sự kiện nhận tương ứng của sự kiện gửi **s_i**



Thứ tự FIFO
Thuật toán (1)

- Mỗi tiến trình lưu một ma trận M[1..N, 1..N]
- Phần tử M[j,k] tại tiến trình P_i lưu lại số lượng thông điệp được gửi từ tiến trình P_j cho tiến trình P_k, được biết bởi tiến trình P_i

Thứ tự FIFO
Thuật toán (2)

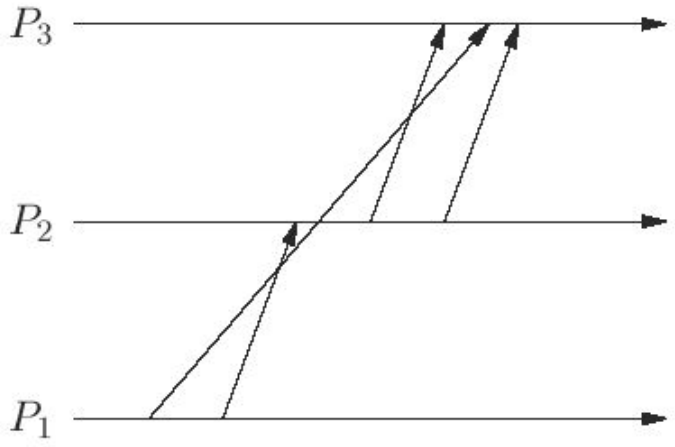
1. Khi P_i gửi một thông điệp tới P_j :
   - M[i,j] = M[i,j] + 1
   - Ma trận M được gắn cùng với thông điệp gửi đi
2. Khi P_i nhận được một thông điệp cùng với ma trận W từ P_j:
   - Nếu $W[j,i] = M[j,i] + 1$ thì tiếp nhận thông điệp này và cập nhật: $M = max(M, W)$
   - Nếu không, khóa thông điệp này lại, chờ đến khi điều kiện trên được thoả mãn

Thứ tự nhân quả

Causal ordering

Thứ tự nhân quả

- Một thứ tự thông điệp mạnh hơn FIFO
- **Trực quan**: một thông điệp không bị vượt qua bởi một chuỗi các thông điệp khác
- Ví dụ: một thứ tự FIFO nhưng không phải thứ tự nhân quả



Thứ tự Nhân quả
Định nghĩa

- Gọi r_1, r_2 là 2 sự kiện nhận trên cùng một tiến trình và s_1, s_2 là 2 sự kiện gửi tương ứng của r_1, r_2
  - s_1, s_2 có thể ở trên 2 tiến trình khác nhau!
- Nếu s_1 xảy ra trước s_2 thì r_2 không thể nhận được trước r_1

$$s_1 \rightarrow s_2 \Rightarrow \neg(r_2 \prec r_1)$$

Thứ tự nhân quả
Thuật toán (1)

- Tại mỗi tiến trình lưu một ma trận M[1..N, 1..N]
- Phần tử M[j,k] tại P_i lưu lại số lượng thông điệp được gửi từ tiến trình P_j cho tiến trình P_k, được biết bởi tiến trình P_i

Thứ tự nhân quả Thuật toán (2)

1. Khi P_i gửi một thông điệp tới P_j :
   - M[i,j] = M[i,j] + 1
   - Ma trận M được gắn cùng với thông điệp gửi đi
2. Khi P_i nhận được một thông điệp cùng với ma trận W từ P_j:
   - Nếu $W[j,i] = M[j,i] + 1$ và $\forall k \neq j$: $M[k,i] \geq W[k,i]$ thì tiếp nhận thông điệp này & cập nhật: $M = max(M, W)$
   - Nếu không, khóa thông điệp lại, chờ đến khi 2 điều kiện trên được thoả mãn

Mã giả cho Thuật toán thứ tự nhân quả tại P_i

P_i::

var

M:array[1..N, 1..N] of integer initially $\forall j, k : M[j, k] = 0$;

To send a message to P_j:

$M[i, j] := M[i, j] + 1$;

piggyback M as part of the message;

To receive a message with matrix W from P_j

enabled if $(W[j, i] = M[j, i] + 1) \wedge (\forall k \neq j : M[k, i] \geq W[k, i])$

$M := max(M, W)$;

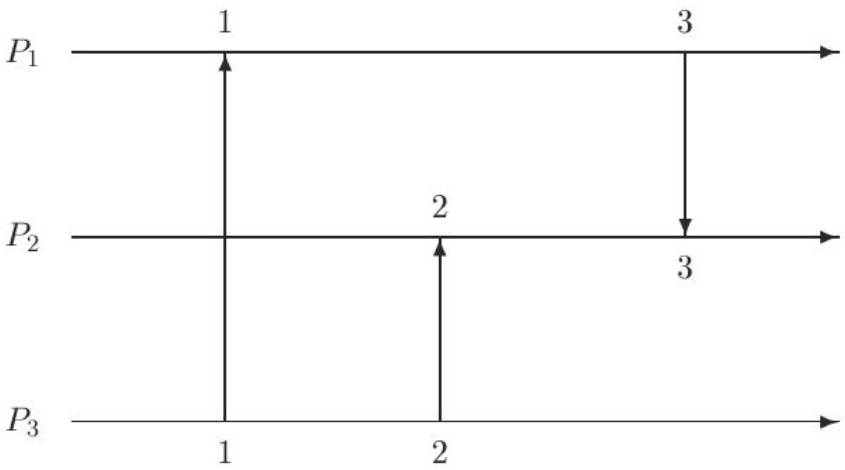
Thứ tự đồng bộ

Synchronous ordering

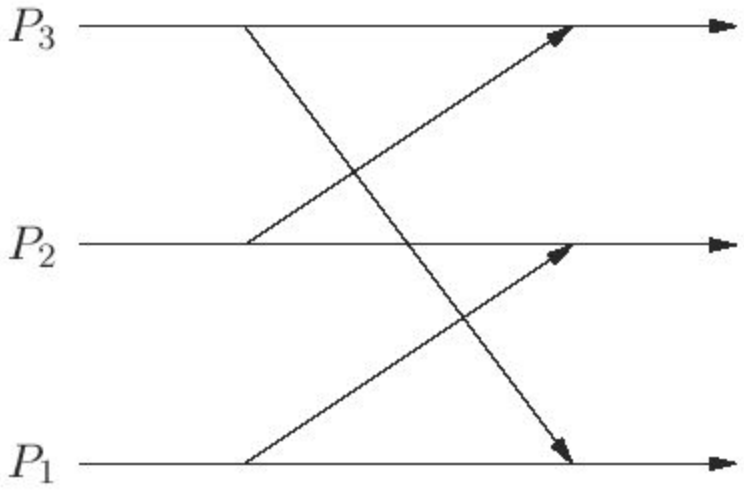
Thứ tự đồng bộ (1)

- Thứ tự này mạnh hơn thứ tự nhân quả
- Một tính toán thoả mãn thứ tự đồng bộ nếu tất cả thông điệp được gửi & nhận *một cách tức thời*
  - Dấu thời gian của sự kiện gửi và nhận thông điệp là giống nhau
  - Truyền thông *điểm* (point-to-point)

Thứ tự Đồng bộ



Thứ tự không đồng bộ



Thứ tự đồng bộ (2)

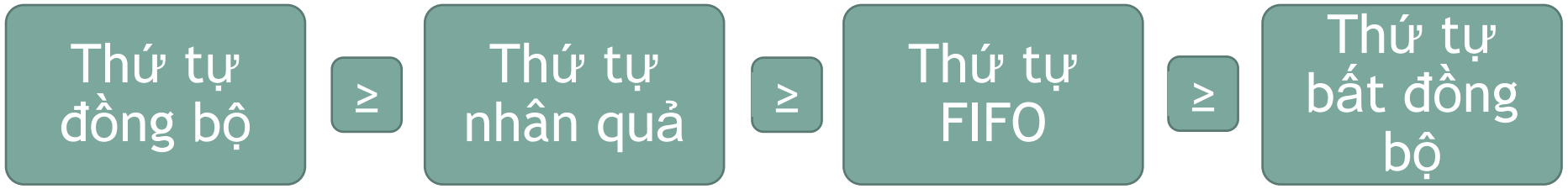
Gọi E là tập các sự kiện. Một tính toán là được gọi đồng bộ khi và chỉ khi tồn tại một ánh xạ T từ E tới tập các số tự nhiên sao cho:

$$\forall s, r, e, f \in E: s \rightsquigarrow r \Rightarrow T(s) = T(r)$$
$$e \prec f \Rightarrow T(e) < T(f)$$

- Điều kiện SYNC

Chứng minh: với bất kỳ 2 sự kiện e và f trong hệ thống phân tán được sắp thứ tự đồng bộ, thì:

$$(e \rightarrow f) \wedge \neg(e \rightsquigarrow f) \Rightarrow T(e) < T(f)$$


Thứ tự đồng bộ
≥
Thứ tự nhân quả
≥
Thứ tự FIFO
≥
Thứ tự bất đồng bộ

Thứ tự đồng bộ
Thuật toán

- Thêm các thông điệp điều khiển (*ack*, *request*, *permission*) để đảm bảo thứ tự đồng bộ
  - Những thông điệp điều khiển này không cần tuân theo thứ tự đồng bộ !
- Sử dụng 2 loại thông điệp
  1. ***Thông điệp lớn***: được gửi bởi tiến trình có định danh lớn hơn tới tiến trình có định danh nhỏ hơn.
  2. ***Thông điệp nhỏ***: được gửi bởi tiến trình có định danh nhỏ hơn tới tiến trình có định danh lớn hơn.
- Một tiến trình có thể ở trong 2 trạng thái:
  1. *active* (*chủ động*)
  2. *passive* (*thụ động*)
- Ban đầu tất cả tiến trình đều ở trạng thái *active*

Thứ tự đồng bộ
Thuật toán (3)

1. Tiến trình P_i ở trạng thái *active* có thể gửi một *thông điệp lớn* (tới tiến trình P_j có định danh nhỏ hơn, i > j)
   - Sau khi gửi, P_i chuyển sang trạng thái *passive* cho đến khi nhận được thông điệp *ack* từ tiến trình P_j

- Chú ý:
  - Tiến trình ở trạng thái *passive* không thể gửi hoặc nhận thông điệp, trừ khi đó là thông điệp *ack*

Thứ tự đồng bộ Thuật toán (4)

2. Để tiến trình P_j gửi 1 *thông điệp nhỏ* tới tiến trình P_i có định danh lớn hơn ($j < i$), P_j cần sự cho phép từ P_i
   - P_j có thể gửi thông điệp *request* bất kỳ lúc nào
   - P_i chỉ có thể cho phép (gửi thông điệp *permission*) khi P_i đang ở trạng thái *active*
   - Sau đó, P_i chuyển sang trạng thái *passive* cho đến khi P_i nhận thông điệp nhỏ từ P_j

P_i ::

var

$state$: $\{active, passive\}$ initially $active$;

To send m to P_j, $(j < i)$

enabled if $(state = active)$:

send m to P_j

$state := passive$;

Upon receive m from P_j, $(j > i)$

enabled if $(state = active)$:

send *ack* to P_j;

Upon receive *ack*:

$state := active$;

To send a message $(message\_id, m)$ to P_j, $(j > i)$

send *request(message_id)* to P_j;

Upon receive *request(message_id)* from P_j, $(j < i)$

enabled if $(state = active)$:

send *permission(message_id)* to P_j

$state := passive$;

Upon receive *permission(message_id)* from P_j, $(j > i)$

enabled if $(state = active)$:

send m to P_j;

Upon receive m from P_j, $(j < i)$

$state := active$;

25

Thứ tự toàn bộ cho thông điệp đa hướng

Total ordering for multicast messages

Thứ tự toàn bộ cho thông điệp đa hướng

- **Thứ tự toàn bộ:**
  - Nếu tiến trình P_i gửi 2 thông điệp đa hướng *x, y* tới các tiến trình P_j, P_k,... thì tất cả những tiến trình này sẽ nhận thông điệp theo cùng một thứ tự (*x,y hoặc y, x*)
- Lưu ý:
  - Điều này không ám chỉ thứ tự đó là thứ tự *Nhân quả* hoặc thậm chí *FIFO*
  - Xét trường hợp khi P_i gửi 2 thông điệp: m_1 và sau đó là m_2
  - Nếu tất cả tiến trình nhận m_2 trước m_1, thì thứ tự toàn bộ được thỏa mãn, tuy nhiên không thoả mãn thứ tự FIFO

Thứ tự toàn bộ: Thuật toán

- Sử dụng lại các thuật toán cho truy cập tài nguyên chia sẻ
  - Thuật toán mutex tập trung
  - Thuật toán của Lamport
- Thuật toán của Skeen

Thuật toán tập trung cho thứ tự toàn bộ

1. Khi tiến trình P_i muốn multicast 1 thông điệp
   - P_i gửi thông điệp đó cho tiến trình điều phối
   - Tiến trình điều phối lưu 1 hàng đợi các yêu cầu
2. Khi một yêu cầu, được gửi từ 1 tiến trình P_j nào đó, có thể được thực hiện, tiến trình điều phối sẽ multicast thông điệp đó

Thuật toán của Lamport cho thứ tự toàn bộ

- Thay đổi thuật toán mutex của Lamport cho bài toán sắp thứ tự toàn bộ, với giả thiết:
  - Thứ tự FIFO giữa các thông điệp
  - Một thông điệp được broadcast tới tất cả tiến trình khác
- Mô phỏng multicast bằng cách sử dụng các thông điệp broadcast
- Mỗi tiến trình sẽ gồm có 2 vector:
  - 1 đồng hồ vector (sử dụng cho dấu thời gian)
  - 1 hàng đợi (lưu những thông điệp chưa được phân phối)

Thuật toán của Lamport: Các bước thực hiện

1. Khi P_i muốn gửi thông điệp broadcast:
   - P_i gửi thông điệp với dấu thời gian tới mọi tiến trình khác.
2. Khi P_j nhận được 1 thông điệp broadcast
   - Thông điệp và dấu thời gian được lưu trong hàng đợi của P_j
   - Một thông điệp *ack* và dấu thời gian tương ứng được gửi ngược về tiến trình gửi P_i
3. Một tiến trình P_k có thể phân phối thông điệp với dấu thời gian *t* nhỏ nhất trong hàng đợi của nó nếu:
   - P_k đã nhận được thông điệp *ack* có dấu thời gian lớn hơn *t* từ tất cả các tiến trình khác

Thuật toán của Skeen cho thứ tự toàn bộ

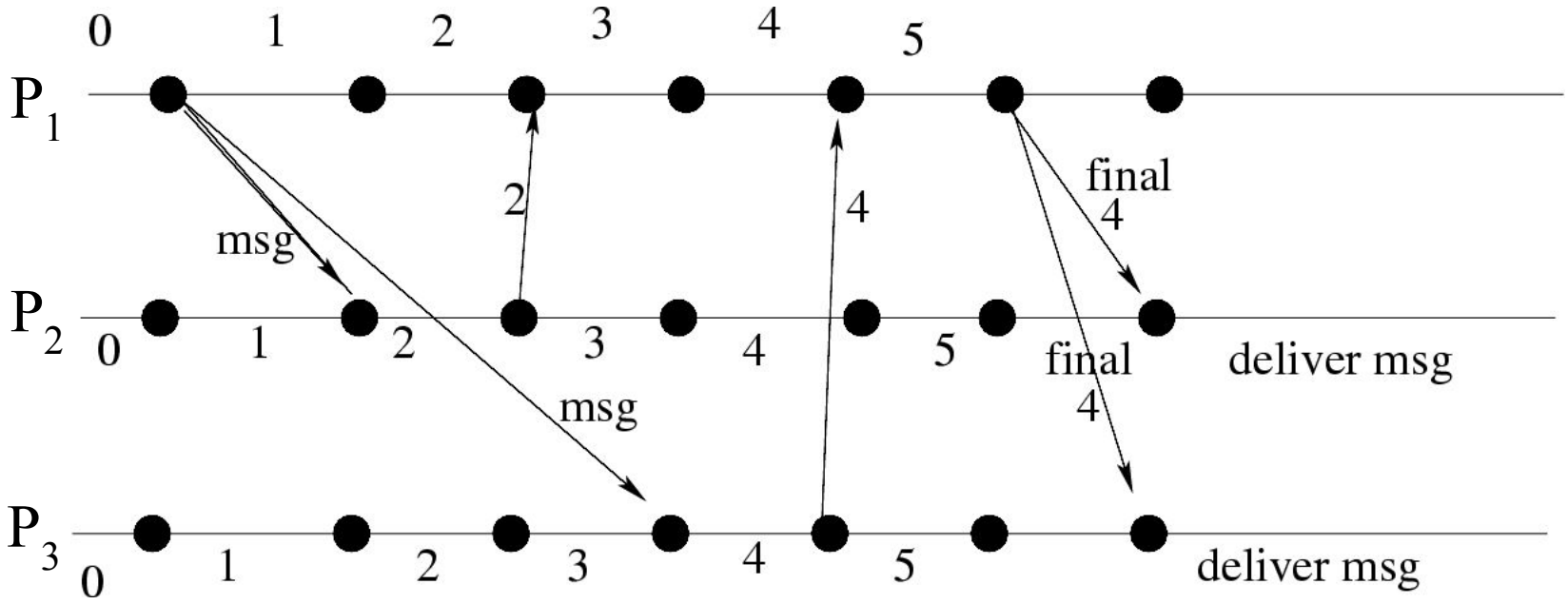
- Thuật toán chỉ yêu cầu số lượng thông điệp tương ứng với số lượng tiến trình nhận (m)
  - Thay vì N-1 tiến trình như thuật toán của Lamport
- Mỗi tiến trình lưu 1 đồng hồ logic

Thuật toán của Skeen: Các bước thực hiện (1)

1. Khi tiến trình P_i muốn gửi 1 thông điệp multicast
   - P_i gửi thông điệp với dấu thời gian tới những tiến trình đích
2. Khi tiến trình P_k nhận được 1 thông điệp
   - P_k đánh dấu thông điệp đó là *chưa-thể-chuyển-đi* (*undeliverable*)
   - P_k gửi ngược giá trị của đồng hồ logic của nó như 1 lời đề xuất cho tiến trình gửi (P_i)

Thuật toán của Skeen: Các bước thực hiện (2)

3. Khi tiến trình gửi P_i nhận được các lời đề xuất gửi từ các tiến trình đích
   - P_i lấy giá trị max của dấu thời gian trong tất cả các đề xuất, gọi giá trị này là *đề-xuất-cuối-cùng*
   - P_i gửi đề xuất cuối cùng này đến những tiến trình đích
4. Khi P_k nhận được đề xuất cuối cùng của một thông điệp
   - P_k đánh dấu thông điệp là *có-thể-chuyển-đi* (*deliverable*)
   - Một thông điệp có thể được xử lý nếu nó có dấu thời gian nhỏ nhất trong hàng đợi của P_k



1. Tiến trình P_0 multicast thông điệp *msg* tới tiến trình P_1 và P_2
2. Khi P_1 và P_2 nhận được thông điệp, chúng đánh dấu nó là *chưa-thể-chuyển-đi* và gửi đề xuất với giá trị dấu thời gian tương ứng là 2 và 4
3. P_0 lấy giá trị lớn nhất của dấu thời gian đề xuất được gửi từ P_1, P_2 và gửi *đề-xuất-cuối-cùng* (4) cho P_1, P_2
4. P_1, P_2 đánh dấu *msg* là *có-thể-chuyển-đi* và sẽ xử lý thông điệp nếu nó có dấu thời gian nhỏ nhất trong hàng đợi

Tài liệu tham khảo

- *Concurrent and Distributed Computing in Java*, Vijay K. Garg, University of Texas, John Wiley & Sons, 2005
- Tham khảo:
  - *Principles of Concurrent and Distributed Programming*, M. Ben-Ari, Second edition, 2006
  - *Foundations of Multithreaded, Parallel, and Distributed Programming*, Gregory R. Andrews, University of Arizona, Addison-Wesley, 2000
  - *The SR Programming Language: Concurrency in Practice*, Benjamin/Cummings, 1993
  - *Xử lý song song và phân tán*, Đoàn văn Ban, Nguyễn Mậu Hân, Nhà xuất bản Khoa học và Kỹ thuật, 2009