# Bài tập Chương 5

## Bài tập 5.1

Các lịch H1 và H4 là tương đương xung đột (conflict equivalent).

## Bài tập 5.2

Các lịch H1, H3, và H4 là khả tuần tự (serializable). Lịch H2 không khả tuần tự vì nó có một chu trình trong đồ thị ưu tiên (T1 → T2 và T2 → T1).

## Bài tập 5.3

Lịch sau đây được phép bởi khóa 2 pha cơ bản (basic 2PL) nhưng không được phép bởi khóa 2 pha nghiêm ngặt (strict 2PL):

H = { W1(x), U1(x), R2(x), W2(x), C2, A1 }

Trong đó U1(x) là thao tác mở khóa của T1. Lịch này không được phép bởi strict 2PL vì T1 mở khóa x trước khi nó kết thúc (Abort).

## Bài tập 5.4

Lịch H4 là có thể phục hồi (recoverable). Các lịch H1, H2, và H3 không thể phục hồi vì chúng chứa các giao dịch đọc từ một giao dịch khác chưa commit và lại commit trước giao dịch đó.

## Bài tập 5.5

(Đây là bài tập thiết kế thuật toán, đáp án là mã giả)

**Thuật toán Distributed 2PL Transaction Manager (D2PL-TM):**

begin  
 // Xử lý Begin\_transaction:  
 Gán ID cho giao dịch.  
  
 // Xử lý Read(x) hoặc Write(x):  
 Xác định site Si chứa x.  
 Gửi yêu cầu khóa (lock request) đến LM tại Si.  
 // LM tại Si sẽ gửi thao tác đến DP sau khi cấp khóa.  
  
 // Xử lý Commit:  
 Bắt đầu giao thức 2PC với tất cả các LM tham gia.  
 // Đợi xác nhận từ 2PC.  
 // Gửi yêu cầu mở khóa đến tất cả các LM tham gia.  
  
 // Xử lý Abort:  
 // Gửi thông báo Abort đến tất cả các LM tham gia.  
 // Gửi yêu cầu mở khóa đến tất cả các LM tham gia.  
end

**Thuật toán Distributed 2PL Lock Manager (D2PL-LM):**

begin  
 // Xử lý yêu cầu khóa từ TM điều phối:  
 Kiểm tra tương thích khóa.  
 Nếu được, cấp khóa, ghi vào bảng khóa, và gửi thao tác đến DP cục bộ.  
 Nếu không, đưa giao dịch vào hàng đợi.  
  
 // Xử lý yêu cầu mở khóa từ TM điều phối:  
 Mở các khóa được giữ bởi giao dịch.  
 Kiểm tra hàng đợi để cấp khóa cho các giao dịch đang chờ.  
end

## Bài tập 5.6

Để xử lý đọc phantom (phantom read), thuật toán 2PL cần được sửa đổi để khóa các vị từ (predicate locking) thay vì chỉ khóa các mục dữ liệu riêng lẻ. Khi một giao dịch T1 thực hiện một truy vấn với một vị từ (ví dụ: WHERE special\_meal = 'yes'), Lock Manager sẽ cấp một khóa trên vị từ đó. Bất kỳ giao dịch T2 nào khác muốn chèn một tuple mới thỏa mãn vị từ này sẽ bị chặn cho đến khi T1 kết thúc. Điều này ngăn chặn việc “bóng ma” xuất hiện nếu T1 thực hiện lại truy vấn.

## Bài tập 5.7

* Một ngày có 24 \* 60 \* 60 = 86400 giây.
* Tần suất của đồng hồ là 1 / 0.1 = 10 ticks/giây.
* Tổng số tick trong 24 giờ là 86400 \* 10 = 864000 ticks.
* Để đảm bảo tính duy nhất của nhãn thời gian, không có hai đồng hồ nào được phép có cùng một giá trị tick tại cùng một thời điểm. Độ trôi tối đa cho phép là **dưới 0.1 giây** trong mỗi 24 giờ. Nếu độ trôi bằng hoặc lớn hơn 0.1 giây, một đồng hồ chạy nhanh có thể “bắt kịp” và tạo ra một giá trị tick trùng với đồng hồ chạy chậm hơn từ chu kỳ trước.

## Bài tập 5.8

(Đây là bài tập thiết kế thuật toán, đáp án là mã giả)

Để tích hợp chiến lược phát hiện bế tắc phân tán (thuật toán của Obermarck) vào D2PL:

**Tại mỗi Lock Manager (LM):**

1. **Xây dựng Local Wait-For Graph (LWFG):** Duy trì đồ thị chờ cục bộ như bình thường.
2. **Thêm các cạnh “external”:** Tạo một nút đặc biệt T\_ex để biểu diễn các giao dịch bên ngoài. Nếu một giao dịch cục bộ Ti đang chờ một giao dịch từ xa Tj, thêm cạnh Ti → T\_ex. Nếu một giao dịch từ xa Tk đang chờ một giao dịch cục bộ Tl, thêm cạnh T\_ex → Tl.
3. **Gửi thông tin chu trình tiềm năng:** Định kỳ, LM gửi thông tin về các đường đi trong LWFG có dạng T\_ex → ... → T\_ex đến các LM khác có liên quan. Thuật toán của Obermarck sử dụng nhãn thời gian của giao dịch để chỉ gửi thông tin đi nếu timestamp(T\_start) < timestamp(T\_end).
4. **Phát hiện bế tắc toàn cục:** Khi một LM nhận được thông tin chu trình từ LM khác, nó sẽ thêm thông tin này vào LWFG của mình. Nếu một chu trình được hình thành, một bế tắc toàn cục đã được phát hiện. LM sẽ chọn một “nạn nhân” (thường là giao dịch có nhãn thời gian gần nhất) để hủy bỏ.

## Bài tập 5.9

Với thuật toán điều khiển tương tranh thứ tự nhãn thời gian lạc quan (optimistic timestamp ordering), yêu cầu lưu trữ của transaction manager tỷ lệ thuận với **số lượng giao dịch đang hoạt động đồng thời** và **kích thước của tập đọc (read set) và tập ghi (write set) của chúng**.

Transaction manager phải lưu giữ RS(T) và WS(T) của tất cả các giao dịch đang trong giai đoạn đọc và thực thi để có thể thực hiện kiểm tra xác thực (validation check) khi một giao dịch muốn commit. Giao dịch càng lớn (nhiều thao tác đọc/ghi) thì yêu cầu lưu trữ càng cao.

## Bài tập 5.10

(Đây là bài tập thiết kế thuật toán, đáp án là mã giả)

**Thuật toán Distributed Optimistic TM:**

begin  
 // Giai đoạn đọc và thực thi:  
 Gửi các thao tác R/W đến các site tham gia. Các site thực hiện trên các bản sao cục bộ/tạm thời.  
  
 // Giai đoạn xác thực (Validation):  
 Gán một nhãn thời gian ts(T).  
 Gửi yêu cầu "VALIDATE" với ts(T), RS(T), WS(T) đến tất cả các site tham gia.  
 Đợi phản hồi từ tất cả các site.  
 Nếu tất cả đều "VALIDATED", bắt đầu giai đoạn ghi (commit).  
 Nếu có ít nhất một "NOT VALIDATED", gửi "ABORT" đến tất cả các site.  
end

**Thuật toán Distributed Optimistic Scheduler:**

begin  
 // Xử lý yêu cầu VALIDATE cho T\_si:  
 Kiểm tra các quy tắc 1, 2, và 3 so với các giao dịch T\_sk đã commit cục bộ mà có ts(T\_sk) < ts(T\_si).  
 Nếu một trong các điều kiện xung đột (ví dụ WS(T\_sk) ∩ RS(T\_si) ≠ ∅) bị vi phạm, gửi "NOT VALIDATED" về TM điều phối.  
 Nếu tất cả các quy tắc được thỏa mãn, gửi "VALIDATED" về TM điều phối.  
end

## Bài tập 5.11

Nếu sử dụng mô hình thực thi phân tán, một giao dịch sẽ được chia thành các giao dịch con, mỗi giao dịch con thực thi tại một site.

* **Transaction Manager:** TM điều phối ban đầu sẽ chỉ khởi tạo các giao dịch con tại các site tham gia. Sau đó, các TM cục bộ tại mỗi site sẽ quản lý việc thực thi của giao dịch con đó. TM điều phối chỉ tham gia vào cuối giai đoạn commit.
* **Lock Manager:** Sẽ không có thay đổi lớn. Mỗi LM cục bộ sẽ xử lý các yêu cầu khóa từ TM cục bộ cho giao dịch con tương ứng. Bế tắc có thể xảy ra giữa các giao dịch con tại các site khác nhau.

## Bài tập 5.12

Có, ví dụ về lịch không khả tuần tự nhưng vẫn đúng đắn là các lịch sử dụng ngữ nghĩa của thao tác.

Ví dụ: Giao dịch T1: x ← x + 5 và T2: x ← x \* 10.

Lịch: R1(x), R2(x), W1(x), W2(x). Lịch này không khả tuần tự (T1→T2 và T2→T1).

Tuy nhiên, nếu các thao tác cộng và nhân có tính giao hoán đối với một số ứng dụng, kết quả có thể được chấp nhận. Một ví dụ khác là **snapshot isolation**, nó cho phép một số lịch không khả tuần tự nhưng vẫn đảm bảo tính nhất quán trong nhiều trường hợp.

## Bài tập 5.13

Giao thức kết thúc (termination protocol) cho 2PC với topo truyền thông phân tán (distributed communication topology):

Khi một participant Pi bị timeout trong trạng thái READY:

1. Pi gửi một thông điệp “inquiry” (hỏi trạng thái) đến tất cả các participant khác.
2. Các participant khác Pj phản hồi như sau:
   * Nếu Pj đã COMMIT hoặc ABORT, nó sẽ gửi lại quyết định đó cho Pi. Pi sẽ tuân theo quyết định này.
   * Nếu Pj cũng đang ở trạng thái READY, nó không thể giúp Pi ra quyết định.
   * Nếu Pj chưa nhận được “prepare” (đang ở INITIAL), nó có thể tự quyết định ABORT và thông báo cho Pi.
3. Pi ra quyết định dựa trên các phản hồi:
   * Nếu nhận được ít nhất một COMMIT hoặc ABORT, nó sẽ theo quyết định đó.
   * Nếu tất cả các participant khác đều ở trạng thái READY, Pi vẫn bị **block**.

## Bài tập 5.14

Giao thức 3PC với topo truyền thông tuyến tính (linear communication topology):

1. **Phase 1 (Vote Request):** Coordinator gửi “prepare” cho participant 2. Participant i sau khi sẵn sàng sẽ gửi “prepare” cho i+1.
2. **Phase 2 (Pre-Commit):** Participant N sau khi sẵn sàng, gửi “vote-commit” về cho N-1. Participant i nhận “vote-commit”, ghi log pre-commit, và gửi “vote-commit” về cho i-1. Quá trình này tiếp tục cho đến khi coordinator nhận được “vote-commit”.
3. **Phase 3 (Commit):** Coordinator ghi log commit, gửi “global-commit” cho participant 2. Participant i nhận “global-commit”, commit, và gửi tiếp cho i+1.

## Bài tập 5.15

Giao thức kết thúc 3PC có thể được sửa đổi để participant gửi trạng thái của mình cho coordinator.

Khi một participant Pi bị timeout trong trạng thái READY hoặc PRECOMMIT:

1. Pi gửi trạng thái hiện tại của mình (READY hoặc PRECOMMIT) đến một coordinator mới được bầu (hoặc đến tất cả các participant khác để họ chuyển tiếp đến coordinator mới).
2. Coordinator mới thu thập trạng thái từ tất cả các participant còn hoạt động.
3. Coordinator mới ra quyết định:
   * Nếu có ít nhất một participant đã ABORT, quyết định toàn cục là ABORT.
   * Nếu có ít nhất một participant đã COMMIT, quyết định toàn cục là COMMIT.
   * Nếu có ít nhất một participant ở trạng thái PRECOMMIT (và không có ai đã commit), coordinator sẽ gửi “global-commit”.
   * Nếu tất cả các participant còn lại đều ở READY, coordinator sẽ gửi “global-abort”.

## Bài tập 5.16

**Chứng minh:**

1. Một scheduler sử dụng thuật toán điều khiển tương tranh nghiêm ngặt (strict) chỉ cho phép một giao dịch T2 đọc hoặc ghi một mục dữ liệu x sau khi giao dịch T1 (mà đã ghi x) đã COMMIT hoặc ABORT.
2. Khi coordinator gửi thông điệp “prepare” cho participant của giao dịch T, điều đó có nghĩa là T đã hoàn thành tất cả các thao tác đọc/ghi của nó.
3. Vì T đã hoàn thành các thao tác, nó sẽ không yêu cầu thêm bất kỳ khóa nào.
4. Do tính nghiêm ngặt, T không thể đang chờ bất kỳ giao dịch nào khác commit (vì nếu nó đọc dữ liệu từ một giao dịch khác, giao dịch đó phải đã commit rồi).
5. Do đó, tại thời điểm nhận được “prepare”, scheduler của participant biết rằng giao dịch có thể commit cục bộ mà không vi phạm tính nhất quán. Nó đã có tất cả các khóa cần thiết và không phụ thuộc vào các giao dịch khác đang chạy. Vì vậy, nó luôn sẵn sàng để commit.

## Bài tập 5.17

(Đây là bài tập thiết kế thuật toán, đáp án là mã giả ở mức cao)

**TM (Coordinator):**

* Quản lý việc bắt đầu/kết thúc giao dịch.
* Gửi các yêu cầu R/W đến các Scheduler tham gia.
* Khởi tạo và điều phối giao thức 2PC khi nhận lệnh Commit.
* Giao tiếp với các Scheduler (participants) và LRM (để ghi log).

**Scheduler (Participant):**

* Triển khai D2PL (yêu cầu/cấp/chờ khóa).
* Triển khai vai trò participant của 2PC.
* Khi nhận “prepare”, nếu đã sẵn sàng, gọi LRM để ghi log “ready”.
* Khi nhận quyết định toàn cục, gọi LRM để ghi log “commit/abort” và thực hiện hành động.
* Khi LRM phục hồi, nhận thông tin trạng thái giao dịch và thực hiện giao thức kết thúc (termination protocol).

**LRM (Local Recovery Manager):**

* Triển khai recovery protocol (ví dụ: ARIES).
* Cung cấp giao diện cho Scheduler để ghi các log của 2PC (begin\_commit, ready, commit, abort).
* Khi phục hồi sau sự cố, đọc log, tái tạo trạng thái các giao dịch và thông báo cho Scheduler về các giao dịch đang trong giai đoạn commit để Scheduler có thể bắt đầu termination protocol.

## Bài tập 5.18

(Đây là bài tập thiết kế thuật toán, đáp án là mã giả)

**Thuật toán LRM no-fix/no-flush:**

Ghi log record <T, x, old\_val, new\_val>.  
Viết new\_val vào trang chứa x trong buffer.  
// Không cần force-write trang ra đĩa.  
  
Ghi log record <T, commit>.  
Force-write tất cả các log record của T ra đĩa (log).  
// Không cần force-write các trang dữ liệu (no-flush).  
Gửi "commit ack".  
  
Ghi log record <T, abort>.  
Dùng log để UNDO các thay đổi của T trong buffer.  
Gửi "abort ack".  
  
// RECOVERY sau sự cố:  
Đọc log từ điểm checkpoint cuối cùng.  
Tạo danh sách UNDO\_list (các giao dịch đã bắt đầu nhưng chưa commit) và REDO\_list (các giao dịch đã commit).  
// Giai đoạn REDO:  
Quét log về phía trước, áp dụng lại tất cả các thay đổi của các giao dịch trong REDO\_list.  
// Giai đoạn UNDO:  
Quét log về phía sau, hoàn tác tất cả các thay đổi của các giao dịch trong UNDO\_list.

## Bài tập 5.19

**TM (Coordinator):**

* Gửi tất cả các yêu cầu khóa R/W đến Scheduler trung tâm.
* Sau khi nhận được “lock granted”, gửi thao tác đến DP tại site tương ứng.
* Khi nhận Commit, bắt đầu 2PC. Vì chỉ có một Scheduler, nó đóng vai trò là participant duy nhất. Quá trình này đơn giản hơn 2PC phân tán.

**Scheduler (Centralized):**

* Quản lý tất cả các khóa cho toàn bộ hệ thống.
* Phát hiện và xử lý bế tắc (cục bộ tại site trung tâm).
* Đóng vai trò participant duy nhất trong 2PC.

**LRM (tại mỗi site):**

* Triển khai no-fix/no-flush.
* Tương tác với TM điều phối (không phải Scheduler) cho các hoạt động ghi log liên quan đến commit (ready, commit, abort).
* Khi phục hồi, giao tiếp với TM điều phối để xác định trạng thái cuối cùng của các giao dịch chưa hoàn thành.