10. Phát hiện Deadlock trong hệ thống phân tán

10.1. Giới thiệu

Deadlocks là một vấn đề cơ bản trong hệ thống phân tán và việc phát hiện deadlock trong hệ thống phấn tán đã nhận được rất nhiều mối quan tâm trong quá khứ. Trong hệ thống phân tán, một tiến trình có thể yêu cầu tài nguyên mà không theo trật tự nào, hay trong khi tiến trình đang giữ tài nguyên khác. Nếu không quản lý được thứ tự phân bố của nguồn tài nguyên tiến trình trong một hệ thống, deadlock có thể xảy ra. Deadlock được định nghĩa như một điệu kiện khi một tập tài nguyên mà tiến trình yêu cầu đã được thực hiện bởi một tiến trình khác trong tập.

Deadlock có thể được xử lý bằng một trong ba chiến lược dưới đây: phòng ngừa deadlock, tránh deadlock và phát hiện deadlock. Phòng ngừa deadlock được thực hiện bằng cách một tiến trình sẽ có đầy đủ các tài nguyên cần thiết trước khi thực thi hoặc chiếm trước một tiến trình đang giữ tài nguyên cần thiết. Trong cách tiếp cận hệ thống tránh deadlock, một tài nguyên được cấp cho một tiến trình nếu hệ thống chung an toàn. Phát hiện deadlock đòi hỏi việc kiểm tra trạng thái tương tác tài nguyên với tiến trình khi có điều kiện deadlock.

Để giải quyết deadlock, ta phải bỏ tiến trình deadlock.

Trong chương này, chúng ta nghiên cứu về kỹ thuật phát hiện deadlock phân tán chung dựa trên các chiến lược khác nhau

10.2. Mô hình hệ thống

Một hệ thống phân tán bao gồm một bộ tiến trình được kết nối bởi mạng truyền thông. Chậm trễ truyền thông là hữu hạn nhưng không thể đoán trước được. Một hệ thống phân tán bao gồm một bộ n các tiến trình không đồng bộ P_1, P_2,..., P_i, ... , P_n giao tiếp bởi chuyển thông điệp trên mạng truyền thông. Không mất tính tổng quát ta giả sử mỗi tiến trình đang chạy trên một tiến trình khác nhau. Các tiến trình không được chia sẻ bộ nhớ chung và giao tiếp bằng việc chuyển thông điệp trên mạng truyền thông. Khống có đồng hồ vật lý chung trong hệ thống để tiến trình có thể truy cập tức thì. Giao tiếp thông thường có thể chuyển thông điệp ra ngoài vùng , thông điệp có thể bị mất, bị cắt hoặc trùng do hết thời gianvà tiến trình truyền lại có thể bị lỗi và link truyền thông bị down. Hệ thống có thể được mô hình như mô hình định hướng trong trong đó các đỉnh đại diện cho tiến trình và các cạnh đại diện cho một chiều kênh truyền thông

Chúng ta đưa ra những giả thiết như sau:

Hệ thống chỉ có duy nhất một tài nguyên có thể tái sử dụng

Tiến trình chỉ cho phép một quyền truy cập tới tài nguyên

Chỉ có một bản copy của mỗi tài nguyên

Mỗi tiến trình có thể có hai trạng thái, chạy hoặc bị chặn. Trong trạng thái hoạt động (còn gọi là trạng thái active), một tiến trình có tất cả những tài nguyên cần thiêt, lệnh thực thi hoặc các lệnh sẵn sang thực thi. Trong trạng thái blocked, một tiến trình phải đợi để yêu cầu một vài tài nguyên.

10.2.1. Wait for graph (WFG)

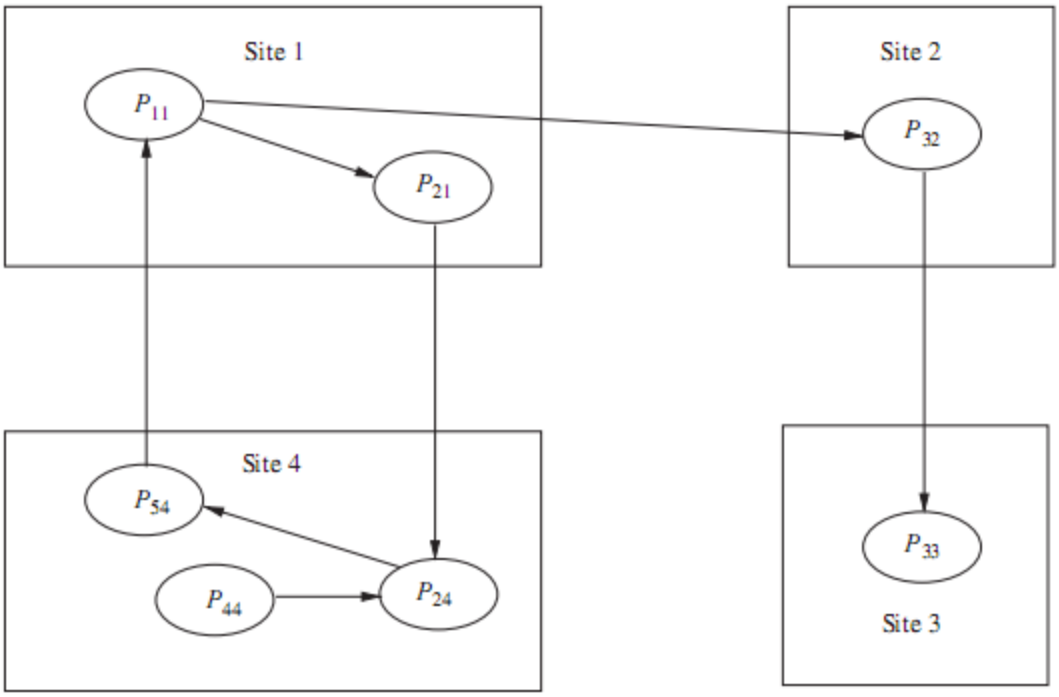
Trong hệ thống phân tán, trạng thái của hệ thống có thể được mô hình bởi đồ thị có hướng, gọi là wait for graph (WFG). Trong một WFG, nút là các tiến trình và có một cạnh có hướng từ nút P_1 tới nút P_2 nếu P_1 bị chặn và chờ P_2 giải phóng tài nguyên. Một hệ thống là deadlock khi và chỉ khi tồn tại một chu kỳ có hướng hoặc nút trong WFG.

Hình 10.1 hiển thị một WFG, trong đó tiến trình P_{11} của trang 1 có một cạnh tới tiến trình P_{21} của trang 1 và một cạnh tới tiến trình P_{32} của trang 2. Tiến trình P_{32} của trang 2 đợi tài nguyên đang được tiến trình P_{33} của trang 3. Tại thời điểm tiến trình P_{21} ở trang 1 đang đợi tiến trình P_{24} ở trang 4 giải phóng tài nguyên và cứ như vậy. Nếu P_{33} bắt đầu đợi tiến trình P_{24}, tiến trình đó nằm trong WFG được gọi trong một deadlock dựa trên trong lúc yêu cầu mô hình.

10.3. Dự bị

10.3.1. Chiến lược xử lý deadlock

Có 3 chiến lược để xử lý deadlock, phòng chống deadlock, tránh deadlock và phát hiện deadlock. Xử lý deadlock là việc phức tạp nhất trong hệ thống phân tán bởi vì không có trang web nào đủ hiểu biết về trạng thái của hệ thống và bởi vì mọi trang web giao tiếp với nhau đều có độ trễ và không dự đoán trước được. Phòng chống deadlock thường đạt được bằng cách một tiến trình có được tất cả tài nguyên cần thiết trước khi bắt đầu thực thi hoặc lấy các tài nguyên cần thiết từ các tiến trình khác. Cách tiếp cận này không có tính thực tế và không đạt hiệu quả cao trong các hệ thống phân tán.



Trong hệ thống phân tán, tránh deadlock bằng cách, một tài nguyên được cấp cho một tiến trình nếu kết quả trạng thái toàn bộ hệ thống an toàn (lưu ý rằng toàn bộ trạng thái bao gồm tất cả các quá trình và các tài nguyên của hệ thống phân tán). Tuy nhiên, do một số vấn đề tránh deadlock là không thực tế trong hệ phân tán.

Phát hiện deadlock đòi hỏi việc kiểm tra trạng thái hoạt động tương tác các tài nguyên tiến trình cho sự xuất hiện của chu kỳ chờ đợ. Phát hiện deadlock trong hệ thống phân tán dường như là phương pháp tốt nhất để xử lý deadlock trong hệ thống phân tán. Trong chương này, chúng tôi hạn chế các cuộc thảo luận các kỹ năng phát hiện deadlock trong hệ thống phân tán.

10.3.2. Vấn đề trong phát hiện deadlock

Xử lý bế tắc bằng cách sử dụng phương pháp phát hiện deadlock cần giải quyết hai vấn đề cơ bản: thứ nhất là phát hiện các deadlock hiện tại, thứ hai là cách giải quyết các deadlock được tìm thấy.

Phát hiện deadlock

Phát hiện các deadlock liên quan đến việc giải quyết hai vấn đề: bảo trì WFG và tìm kiếm WFG để xuất hiện chu kỳ (hoặc nút). Trong hệ thống phân tán, một chu kỳ hoặc nút có thể liên quan tới một số trang web, việc tìm kiếm các chu kỳ phụ thuộc rất nhiều vào cách WFG của hệ thống được biểu diễn trên toàn hệ thống. Tùy theo cách WFG duy trì và tìm kiếm chu kỳ được thực hiện thế nào, có tập trung, phân tán và các thuật toán phân cấp cho việc phát hiện deadlock trong hệ thống phân tán.

Các tiêu chí chính xác

Một thuật toán phát hiện deadlock phải đáp ứng hai điều kiện sau:

- Sự tiến triển (không có deadlock nào là không được phát hiện) thuật toán phải phát hiện tất cả deadlock hiện có trong thời gian có hạn. Một khi deadlock xảy ra, phát hiện deadlock phải thực hiện liên tục, cho tới khi tìm thấy deadlock. Nói cách khác, sau khi chờ đợi các phụ thuộc của deadlock được hình thành, thuật toán không nên thêm chờ bất kỳ sự kiện xảy ra để phát hiện deadlock.
- An toàn (không deadlock nào bị lỗi) thuật toán không báo cáo những deadlock không tồn tại (gọi là ghost hay deadlock lỗi). Trong các hệ thống phân tán khi không có bộ nhớ toàn bộ và không có đồng bộ thời gian, thật khó để thiết kế một thuật toán phát hiện deadlock chính xác bởi vì các trang web có thể có thời gian và WFG không phù hợp với hệ thống. Kết quả là, các trang web có thể phát hiện một chu kỳ không tồn tại nhưng có phân đoạn khác nhau tồn tại trong hệ thống tại các thời điểm khác nhau. Đây là lý do tại sao nhiều thuật toán phát hiện deadlock được báo cáo trong y học không chính xác.

Giải quyết khi phát hiện deadlock

Giải quyết deadlock liên quan đến đứt đoạn phụ thuộc giữa các tiến trình để giải quyết deadlock. Nó liên quan đến quay lại một hoặc nhiều tiến trình deadlock và phân bổ tài nguyên cho các hệ thống bị chặn để chúng tiếp tục thực thi. Lưu ý rằng một số thuật toán phát hiện deadlocks lan truyền thông tin liên quan đến phụ thuộc chờ dọc theo cạnh của đồ thị chờ. Do đó, khi một phụ thuộc chờ bị phá vỡ, các thông tin tương ứng cần làm rõ ngay lập tức trong hệ thống. Nếu thông tin này không được làm rõ kịp thời có thể dẫn đến việc phát hiện ra các deadlock ma . Làm rõ không đúng lúc và không phù hợp của phụ thuộc chờ bị phá vỡ là lý do chính khiến cho các thuật toán phát hiện các deadlock trong y học không chính xác.

10.4. Mô hình của deadlocks

Tính toán phân tán cho phép nhiều kiểu yêu cầu tài nguyên. Một tiến trình có thể yêu cầu một tài nguyên đơn hoặc nhiều tài nguyên kết hợp để thực thi. Phần này sẽ giới thiệu hệ thống cấp bậc của mô hình yêu cầu bắt đầu với những mẫu giới hạn đến những mẫu không giới hạn. Hệ thống này sẽ được dung để phân loại các thuật toán phát hiện deadlock dựa trên độ phức tạp của tài nguyên yêu cầu.

10.4.1. Mô hình tài nguyên đơn

Mô hình tài nguyên đơn là tài nguyên đơn giản nhất trong hệ thống phân tán, nơi một tiến trình có thể có nhiều nhất môt yêu cầu cho một đơn vị tài nguyên. Từ khi ngoài mức độ tối đa của nút trong một WFG cho mô hình tài nguyên đơn là 1, sự xuất hiện của một chu kỳ trong WFG có thể chỉ ra một deadlock. Trong phần sau, một thuật toán phát hiện deadlock trong mô hình tài nguyen đơn sẽ được trình bày.

10.4.2. Mô hình AND

Trong mô hình AND, một tiến trình có thể yêu cầu nhiều hơn một tài nguyên cùng lúc và yêu cầu được thỏa mãn chỉ sau khi tất cả yêu cầu tài nguyên được cấp phát cho tiến trình. Tài nguyên được yêu cầu có thể tồn tại ở một vị trí khác. Ngoài mức độ một nút trong WFG cho mô hình AND có thể lớn hơn 1. Sự xuất hiện của một chu kỳ trong WFG chỉ ra một deadlock trong mô hình AND. Mỗi nút trong WFG trong mô hình như vậy gọi là nút AND.

Xét ví dụ được mô tả trong hình 10.1. Tiến trình P_11 có hai yêu cầu tài nguyên nổi bật. Trong mô hình AND, P_11 có thể hoạt động từ trạng thái idle chỉ sau khi tất cả tài nguyên được cấp phát. Một chu kỳ là P_11 → P_21 → P_24 → P_54 → P_11, tương ứng với một trường hợp deadlock

Trong mô hình AND, nếu một chu kỳ được phát hiện trong WFG, nó sẽ chứa một deadlock nhưng không phải ngược lại. Đó là quá trình có thể không cần là một phần của chu kỳ nhưng vẫn có thể là bị deadlock. Xét tiến trình P_44 trong hình 10.1. Nó không phải một phần của bất kỳ chu kỳ nào nhưng vẫn bị deadlock và not phụ thuộc vào P_24, cái bị deadlock. Từ trong mô hình tài nguyên đơn, một tiến trình có thể có nhiều nhất một yêu cầu nổi bật, và mô hình AND tổng quát hơn mô hình tài nguyên đơn

10.4.3. Mô hình OR

Trong mô hình OR, một tiến trình có thể tạo một yêu cầu cho nhiều tài nguyên cùng lúc và yêu cầu được thỏa mãn nếu bất kỳ một tài nguyên nào được cấp phát. Tài nguyên được yêu cầu có thể nằm trong vị trí khác. Nếu tất cả yêu cầu trong WFG là OR, thì các nút được gọi là nút OR. Sự xuất hiện của một chu kỳ trong

WFG của mô hình OR không bao gồm một deadlock trong mô hình OR. Để làm rõ hơn, xét hình 10.1. Nếu tất cả các nút là nút OR, tiến trình P_11 không bị deadlock bởi vì một khi tiến trình P_33 giải phóng tài nguyên, P_32 sẽ hoạt động vì một yêu câu của nó được cấp phát. Sau khi P_32 thực thi xong và giải phóng tài nguyên, tiến trình P_11 có thể tiếp tục.

Trong mô hình OR, sự xuất hiện của một nút thắt chỉ ra một deadlock. Trong WFG, một vec tơ v là một nút thắt nếu tất cả u::u có thể truy cập từ v:v có thể truy cập từ u. Không có đường bắt đầu một nút thắt có điểm kết thúc.

Một deadlock trong mô hình OR có thể được định nghĩa như sau: một tiến trình P_i bị chặn nếu có một yêu cầu OR chưa giải quyết được đáp ứng. Với mỗi tiến trình bị chặn, có một bộ tiến trình kết hợp gọi là bộ phụ thuộc. Một tiến trình có thể chuyển từ tạng thái idle sang trạng thái active khi nhận một thông điệp cấp phát từ bất kỳ tiến trình nào trong bộ phụ thuộc của nó. Một tiến trình bị chặn vĩnh viễn nếu nó không bao giờ nhận được thông điệp cấp phát từ bất kỳ tiền trình nào trong bộ phụ thuộc. Bằng trực giác, một bộ tiến trình S bị deaclock nếu tất cả tiến trình trong S bị chặn vĩnh viễn. Chính xác là, một bộ tiến trình bị deadlocked khi các điều kiện sau đây là đúng:

Mỗi tiến trình trong bộ S bị chặn

Bộ phụ thuộc của mỗi tiến trình trong S là một bộ nhỏ của S

Không có thông điệp cấp phát nào được chuyền giữa giữa tiến tình bất kỳ trong bộ S

Chúng ta sẽ thấy một tập hợp các tiến trình S bị chặn vĩnh viễn trong mô hình OR nếu các điều kiện trên là đúng. Một tiến trình bị chặn P trong bộ S chỉ hoạt động được khi nhận được thông điệp cấp phát từ một tiến trình trong bộ phụ thuộc, một bộ con của S. Lưu ý rằng không có thông điệp cấp phát nào có thể có được từ bất kỳ tiến trình nào trong S vì chúng đã bị chặn. Ngoài ra, điều kiện thứ ba rằng không có thông điệp cấp phát được chuyển giữa bất kỳ hai tiến trình nào trong S. Vì thấy, tất cả các tiến trình trong S đều bị chặn vĩnh viễn.

Do đó, phát hiện deadlock trong mô hình OR tương được với việc tìm ra nút thắt trong đồ thị. Lưu ý rằng, có thể có một tiến trình deadlock không phải một phần trong nút thắt. Xét hình 10.1, khi P_44 có thể bị deadlock mặc dù nó không phải là một nút thắt. Do đó trong mô hình OR, một tiến trình bị chặn P là deadlock khi nó nằm trong một nút thắt hoặc nó chỉ tiếp cận một tiến trình trong một nút thắt.

10.4.4. Mô hình AND-OR

Khái quát của hai mô hình trước là mô hình AND-OR. Trong mô hình AND-OR, một

yêu cầu có thể chỉ ra kết hợp bất kỳ giữa and và or trong yêu cầu tài nguyên. Ví dụ, trong mô hình AND-OR, một yêu cầu cho đa tìa nguyên có thể có dạng x and (y or z). Tài nguyên được yêu cầu có thể nằm ở vị trí khác. Để phát hiện sự có mặt của deadlock trong mô hình này, không có cấu trúc lý thuyết đồ thị sử dụng WFG. Kể từ khi một deadlock là một thuộc tính ổn định (tức là một khi nó tồn tại, nó không tự mất đi), thuộc tính này có thể được khai thác và một deadlock trong mô hình AND-OR có thể được phát hiện bằng các ứng dụng lặp lại của các thử nghiệm cho mô hình OR. Tuy nhiên, đây là một chiến lược hiệu quả, Các thuật toán hiệu quả để phát hiện deadlock sẽ được thảo luận sau.

10.4.5. Mô hình $\binom{p}{q}$

Một mẫu khác của mô hình AND-OR là mô hình (p¦q) (còn được gọi là mô hình P-out-of-Q), nó cho phép một yêu cầu bao gồm bất kỳ k tài nguyên có sẵn từ một bể của n tài nguyên. Cả hai mô hình đều giống nhau về mặt ý nghĩa. Tuy nhiên, mô hình (p¦q) thích hợp với các yêu cầu có dạng nhỏ gọn hơn.

Mỗi yêu cầu trong mô hình (p¦q) có thể được thể hiện trong mô hình AND-OR và ngược lại. Lưu ý rằng yêu cầu AND cho tài nguyên b có thể được nêu như (p¦q) và yêu cầu OR cho tài nguyên p có thể nêu như (p¦q).

10.4.6. Mô hình không hạn chế

Trong mô hình không hạn chế, không có giả thiết được tạo liên quan đến cấu trúc cơ bản của tài nguyên yêu cầu. Trong mô hình này, chỉ có một giả thiết rằng deadlock có tính ổn định, do đó nó là mô hình tổng quát nhất. Cách nhìn này vào các vấn đề deadlock chia các mối quan tâm: về thuộc tính của vấn đề (ổn định và deadlock) là được tách từ hệ thống tính toán phân tán cơ bản (ví dụ, thông điệp thông qua và truyền thông đồng bộ). Do đó, các thuật toán này được dung để phát hiện các thuộc tính ổn định như thỏa thuận với mô hình chung. Nhưng các thuật toán này có giá trị về lý thuất nhiều hơn cho các hệ thống phân tán vì không có them giả định được đưa ra về các hệ thống tính toán phân tán cơ bản dẫn đến một thỏa thuận tốt như phần trên (cái có thể tránh được trong một mô hình đơn giản hơn như AND hay OR).

10.5. Phân loại các thuật toán phát hiện deadlock phân tán của Knapp

Thuật toán phát hiện deadlock phân tán có thể được chia thành bốn lớp: path pushing, edge chasing, tính toán khuếch tán, và phát hiện trạng thái toàn bộ.ư

10.5.1. Thuật toán path-pushing

Trong thuật toán path-pushing, deadlock được phát hiện bằng cách duy trì một

WFG toàn cục rõ rang. Ý tưởng cơ bản là xây dừng một WFG toàn cục cho mỗi trang web của hệ thống phân tán. Trong lớp thuật toán này, bất cứ khi nào deadlock được thực hiện, mỗi trang web sẽ gửi WFG cục bộ tới tất cả các trang web lân cận. Sauk hi cấu trúc dữ liệu cục bộ của mỗi trang được cập nhật, WFG cập nhật được thông qua các trang web, và các thủ tục được lặp lại cho tới khi một trang web có được một bức tranh đầy đủ về trngj thái cục bộ để thông báo deadlock hoặc thiết lập mà không có deadlock nào có mặt. Tính năng gửi vòng quanh các đường dẫn của WFG toàn cục đã dấn đến thuật toán path-pushing

Ví dụ của thuật toán này là Menasce-Muntz [33], Gligor và Shattuck [11], Ho và Ramamoorthy [18], và Obermarck.

10.5.2. Thuật toán edge-chasing

Trong thuật toán edge-chasing, sự xuất hiện của một chu kỳ trong cấu trúc đồ thị phân tán được xác minh bằng cách truyền bá thông điệp đặc biệt gọi là thăm dò dọc theo các cạnh của đồ thị. Các thông điệp thăm dò khác nhau để yêu cầu và trả lời. Sự hình thành của chu kỳ có thể được phát hiện bởi các trang web nếu nó nhận được thăm dò tương thích được gửi bởi nó trước đó.

Bất cứ khi nào một tiến trình đang thực thi nhận được một thông điệp thăm dò, nó chỉ đơn giản loại bỏ thông điệp này và tiếp tục. Chỉ tiến trình bị chặn truyền bá thông điệp thăm dò dọc theo các cạnh. Một biến thể thú vị của phương thức này được tìm thấy trong Mitchell, trong đó thăm dò được gửi theo yêu cầu và trong hướng ngược lại của cạnh.

Ưu điểm chính của thuật toán là edge-chasing là thông điệp thăm dò có kích thước cố định thường là rất ngắn. Ví dụ về thuật toán này gồm các thuật toán Chandy et al. [6], Choudhary et al. [7], Kshemkalyani–Singhal [27], và Sinha–Natarajan.

10.5.3. Thuật toán dựa trên tính toán khuếch tán

Trong thuật toán phát hiện deadlock phân tán dựa trên tính toán khuếch tán, tính toán phát hiện deadlock được khuếch tán thông qua WFG của hệ thống. Các thuật toán này được dung thuật toán echo để phát hiện deadlock. Sự tính toán này được xếp chồng lên tính toán phân tán phía dưới. Nếu tính toán chấm dứt, người khởi xướng sẽ tuyên bố một deadlock. Tính năng chính của tính toán xếp chồng là WFG cục bộ phản ánh ngầm trong cấu trúc của tính toán. WFG thực tế không bao giờ được xây dựng rõ rang.

Để phát hiện một deadlock, một tiến trình sẽ gửi đi một thông điệp truy vấn dọc theo tất cả các cạnh trong WFG. Những truy vấn này liên tục được truyền bá

(khuếch tán) thông qua các cạnh của WFG. Truy vấn được loại bỏ bởi một tiến trình đang chạy và được lặp lại bởi các tiến trình bị chặn theo cách sau: khi một tiến trình đầu tiên bị chặn nhận được một thông điệp truy vấn cho một khởi tạo phát hiện deadlock đặc biệt, nó không gửi thông điệp trả lời cho đến khi nó nhận được thông điệp trả lời cho mỗi truy vấn mà nó gửi (cho những kế thừa của nó trong WFG). Với tất truy vấn tiếp theo để bắt đầu phát hiện deadlock, nó lập tức gửi lại một thông điệp trả lời. Người khởi xướng của phát hiển deadlock phát hiện một deadlock khi nó nhận được một trả lời cho mọi truy vấn mà nó gửi đi. Ví dụ của thuật toán này gồm Chandy–Misra–Haas cho mô hình OR và the Chandy–Herman

10.5.4. Thuật toán dựa trên trạng thái toàn cục

Thuật toán phát hiện deadlock dựa trên phát hiện trạng thái toàn cục khai thác các vấn đề sau: (i) một bản chụp nhất quán của một hệ thống phân tán có thể thu được mà không cần làm lạnh các tính toán cơ bản, và (ii) một bản chụp phù hợp có thể không đại diện cho trạng thái hệ thống ở bất kỳ thời điểm nào, nhưng nếu một thuộc tính ổn định được lưu trong hệ thống trước khi bản chụp tổng hợp được khởi tạo, thuộc tính này vẫn được lưu trong bản chụp.

Do đó deadlock phân tán có thể được phát hiện bằng cách lấy các bản chụp của hệ thống và kiểm tra nó với điều kiện của một deadlock. Ví dụ của dạng thuật toán này Bracha–Toueg [2], Wang et al. [45], và Kshemkalyani–Singhal [26]

10.6. Mô hình toán học Mitchell và Merritt đối với nguồn tài nguyên đơn (1 nguồn)

Thuật toán Mitchell và Merritt được xếp vào dạng mô hình toán học xác định các điểm khóa chết (nút chặn-deadlock) mà trong đó các nút tín hiệu dò (tín hiệu dò – probes) được gửi ngược lại đến các cạnh của mô hình đợi (WFG). Trong trường hợp này, khi các nút tín hiệu dò được gửi đi bởi tiến trình (process) trước đó mà trờ lại là chính nó, thì tiến trình đó được xem như bị chặn (process ở đây được coi là một nút trong mô hình WFG). Đặc điểm hữu dụng của mô hình thuật toán này như sau:

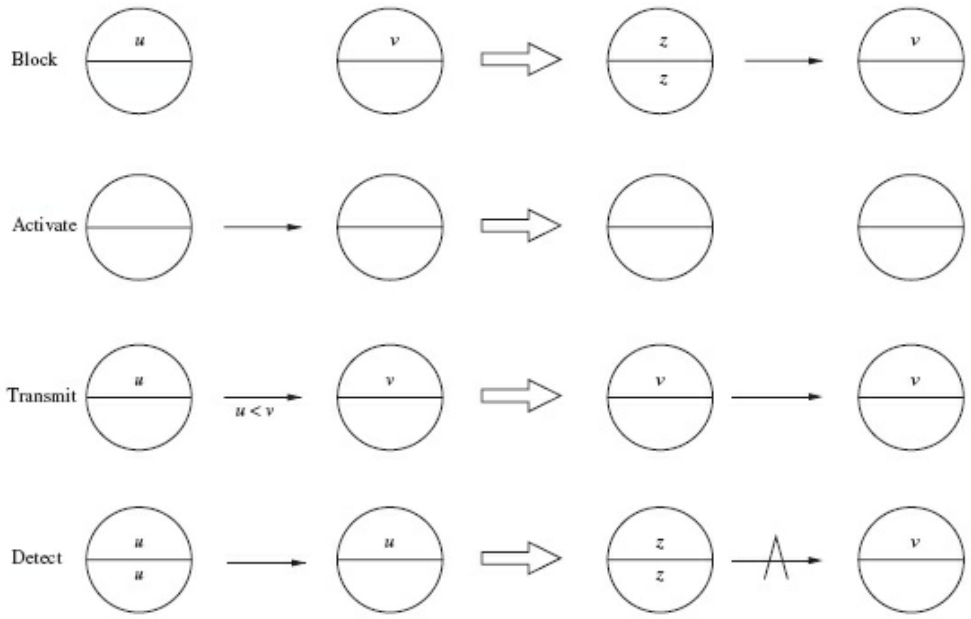
- Chỉ một tiến trình trong chu kỳ xác định điểm khóa chết. Đây là giải pháp đơn giản nhất để xác định điểm khóa chết – tiến trình này có thể bỏ qua (hủy bỏ - abort) tín hiệu đi qua bản thân nó để xác định lời giải xác định điểm khóa chết. Thuật toán này có thể thực hiện việc xác định điểm khóa chết bằng cách tính toán gồm tất cả những nút tiến trình ưu tiên, và nút tiền trình ưu tiên thấp nhất trong một chu kỳ truyền tín hiệu xác định được điểm khóa chết hoặc hủy bỏ.

- Trong thuật toán này, mặc dù điểm khóa chết ảo có thể không bị loại trừ, nhưng nút tiến trình đã được xác định tại điểm khóa chết sẽ tự động bị bỏ qua. Chỉ có điểm khóa chết thực sẽ được xác định, tuy nhiên, nút tiến trình này có thể được biểu hiện chỉ trong trưởng hợp không xuất hiện hiện tượng tự hủy bỏ

Mỗi nút trên đồ thị đợi (WFG) được gán hai loại nhãn: nhãn cá thể (private label) là loại nhãn gán duy nhất cho nút trên đồ thị tại tất cả các lần tín hiệu truyền qua, mặc dù loại nhãn này có thể không phải là một hằng số; nhãn tập thể (public label) là loại nhãn gán cho nút trên đồ thị mà tín hiệu truyền qua nút tiến trình này có thể đọc được tín hiệu truyền qua nút tiến trình khác và nhãn này không phải là nhãn duy nhất. Với mỗi tiến trình được đặc trưng bằng tham số u/v trong đó u và v lần lượt là nhãn tập thể và nhãn cá thể. Tại thời điểm ban đầu, nhãn tập thể và cá thể là cân bằng tại mỗi nút tiến trình.

Một đồ thị đợi tổng quát được duy trì và xác định toàn bộ các trạng thái của hệ thống phân tán. Thuật toán này được xác định bởi 4 trạng thái truyền tín hiệu biểu diễn trên hình 10.2, trong đó z = inc (u,v) và inc(u,v) sinh ra nhãn duy nhất lớn hơn cả u và v. Loại nhãn này không được chỉ ra và không thay đổi. Nút khóa tạo trên một cạnh của đồ thị đợi. Hai tín hiệu cần cho quá trình: một tín hiệu yêu cầu đến và một tín hiệu hồi đáp đến nút tiến trình bị khóa để thông báo đến nhãn tập thể của nút tiến trình đang đợi trên đồ thị. Các biểu hiện được kích hoạt là một quá trình đã thu thập được tín hiệu đến từ các nút tiến trình đang đợi sẵn trên đồ thị đợi. Quá trình truyền tin hiệu trên các nút tiến trình được gán các nhãn lớn hơn gửi ngược đến các cạnh của đồ thị được thực hiện bằng cách gửi các tín hiệu dò. Khi đó, nút tiến trình nhận được tín hiệu dò sẽ ít hơn nút tiến trình được gán nhãn tập thể có thể bỏ qua. Việc xác định này có nghĩa là số nút tín hiệu dò kèm theo số nút tiến trình được gán nhãn cá thể của một số quy trình truyền tín hiệu đã trả về những nút đó, điều này chỉ ra rằng nút tiến trình là một nút khóa chết.

Mitchell và Merritt đã chỉ ra rằng, mọi nút khóa chết đều được xác định, đồng thời trong trường hợp không xuất hiện hiện tượng tự hủy bỏ, chỉ có nút khóa chết thực được xác định. Trong trường hợp này, có các chuỗi bất biến u/v; v<=u (phần hình tự đọc nhé, vì nó chỉ diễn giải)



Hình 10.2: Bốn thể chuyển trạng thái

Chứng minh: Ban đầu u = v cho tất cả các quá trình. Các yêu cầu duy nhất mà thay đổi u hoặc v là:

1. Chặn: u và v được thiết lập như vậy mà u = v.
2. Chuyển giao: u tăng lên.

   Do đó, các bất biến sau.

   Từ bất biến trước đó, chúng ta có bổ đề sau đây.

   Bổ đề 10.1: Đối với quá trình bất kỳ u / v, nếu u> v, sau đó u đã được thiết lập bởi một bước chuyển giao.

   Định lý 10.1: Nếu một bế tắc được phát hiện, một chu kỳ của các nút chặn tồn tại.

   Chứng minh: Một bế tắc được phát hiện nếu sau cạnh p → p' tồn tại:

Chúng tôi sẽ chứng minh những tuyên bố sau đây:

1. u đã được nhân giống từ p để p' thông qua một chuỗi các đã truyền đi.
2. p đã liên tục bị chặn vì nó truyền giao tới u.
3. Tất cả các nút trung gian trong đường dẫn của chuyển giao (l), bao gồm cả p ', đã liên tục bị chặn kể từ khi chúng được truyền giao tới u.

Từ những tuyên bố trên, các bằng chứng cho định lý sau khi thảo luận dưới đây.

Từ bất biến và sự độc đáo của riêng nhãn v của p':v <u. Từ bổ đề 10.1, u đã được thiết lập bởi một bước chuyển giao. Từ ngữ nghĩa của chuyển giao, có một số p'' với nhãn u riêng biệt và nhãn w công cộng. Nếu u=w thì p = p'', và nó là một thành công. Nếu không thì, nếu w <u, chúng ta lặp lại các đối số. Do chỉ có một quá trình với u = v, nó là p. Nếu p là tích cực, thì nó chỉ ra rằng nó đã chuyển giao tới u ngược lại nó bị chặn nếu phát hiện bế tắc. Do đó khi ngăn chặn nó tăng lên nhãn hiệu riêng của mình. Nhưng nhãn hiệu riêng và nhãn hiệu công cộng không thể bằng nhau. Hãy xem xét một quá trình đã được hoạt động kể từ khi nó được chuyển giao u. Rõ ràng, người tiền nhiệm của nó là cũng đang hoạt động, như truyền di chuyển theo hướng ngược lại. Bằng cách lặp lại lập luận này, chúng ta có thể thấy rằng p đã hoạt động kể từ khi nó được truyền tới u.

Các thuật toán trên có thể dễ dàng được mở rộng để bao gồm các ưu tiên, do đó bất cứ khi nào một bế tắc xảy ra, quá trình ưu tiên thấp nhất bị hủy bỏ. Thuật toán này có hai giai đoạn. Giai đoạn đầu tiên là gần giống như các thuật toán. Trong giai đoạn thứ hai ưu tiên nhỏ nhất là tuyên truyền xung quanh vòng tròn. Tuyên truyền dừng lại khi một quá trình nhận thức được ưu tiên nhân giống như là của riêng của nó.

Độ phức tạp:

Bây giờ chúng ta tính toán phức tạp của thuật toán. Nếu chúng ta giả định rằng một bế tắc vẫn tồn tại đủ lâu để được phát hiện, sự phức tạp trường hợp xấu nhất của thuật toán là s(s-1)/2 bước chuyển giao, trong đó s là số lượng của các quá trình trong chu kỳ.

10.7. Thuật toán Chandy-Misra-Haas đối với mô hình AND: trên cơ sở mô hình toán học điểm dò – cạnh đồ thị đợi

Thuật toán này sử dụng tín hiệu đặc biệt được gọi là tín hiệu dò (Probe) với ba đầu tín hiệu (i,j,k). Quá trình xác định nút khóa chết được thực hiện ban đầu đối

với nút tiến hình Pi và được gửi đi by nguồn của nút tiến trình Pj đến nguồn của nút tiến trình Pk. Tín hiệu dò sẽ đi một vòng dọc theo các cạnh của đồ thị chờ tổng quát, khi tín hiệu dò quay trả lại nút tiến trình ban đầu truyền đi thì nút khóa chết được xác định.

Nút tiến trình Pj được xem như phụ thuộc vào nút tiến trình Pk nếu tồn tại một chuỗi các nút tiến trình P_j, P_{i1}… P_{im}, P_k trong mỗi một quy trình truyền tín hiệu, ngoài trừ Pk trong chuỗi tiến trình sẽ bị khóa, ngoại trừ nút tiến trình Pj được giữ lại tại nguồn phát, các nút tiến trình khác trong tiến trình truyền tín hiệu trước đó sẽ phải đợi. Nút tiến trình Pj được xem như phụ thuộc cục bộ vào nút tiến trình Pk nếu Pj phụ thuộc vào Pk và cả hai nút tiến trình đều thuộc cùng một vị trí nguồn phát.

Cấu trúc dữ liệu

Mỗi nút tiến trình Pi duy trì một hệ thống logic phụ thuộc thứ i, trong đó Pj là giá trị thực chỉ khi Pj phụ thuộc vào chính nó. Ban đầu Pi luôn sai với mọi giá trị i và j.

Thuật toán:

Thuật toán 10.1 được thực hiện để xác định xem một quá trình chặn được bế tắc. Do đó, một tin nhắn thăm dò tiếp tục được lưu hành dọc theo các cạnh của WFG đồ thị toàn cầu và một bế tắc được phát hiện khi một tin nhắn thăm dò trở lại quá trình khởi đầu của nó.

Nếu Pi là địa phương phụ thuộc vào chính nó

Thì khai báo một khóa chết (deadlock)

Ngược lại cho tất cả P¬¬j và Pk như sau:

a. *Pi là địa phương phụ thuộc vào Pj, và*
b. *Pj đang chờ vào Pk, và*
c. *Pj và Pk là trên các trang khác nhau,*

   gửi một tin thăm dò (i, j, k) cho trang chính của Pk

   Trên biên nhận của một tin thăm dò (i, j, k), trang web phải thực hiện các hành động sau đây:

   Nếu

d. *Pk bị khóa, và*

e. Phụ thuộc k (i) là sai, và

f. Pk đã không trả lời tất cả các yêu cầu của Pj,

Thì

Bắt đầu

Phụ thuộc k (i) = đúng;

Nếu k=i

Thì khai báo Pi là khóa chết (deadlock)

Ngược lại tất cả Pm và Pn như sau:

a'. Pk là địa phương phụ thuộc và Pm, và

b'. Pm đang chờ Pn, và

c'. Pm và Pn là trên các trang khác nhau,

Gửi tin nhắn thăm dò (i, m, n) tới trang chính của Pn

Kết thúc.

Phân tích hiệu suất:

Trong các thuật toán, một tin nhắn thăm dò (mỗi bắt đầu phát hiện bế tắc) được gửi trên mỗi cạnh của WFG mà kết nối các quy trình trên hai trang. Do đó các thay đổi thuật toán hầu hết m(n-1)/2 tin nhắn để phát hiện một bế tắc có liên quan đến quá trình m và kéo dài trên các trang của n. Kích thước của tin nhắn là cố định và là (chỉ có ba chữ số nguyên) rất nhỏ. Sự chậm trễ trong việc phát hiện một bế tắc là 0(n).

3. Thuật toán Chandy-Misra-Haas cho mô hình OR.

Bây giờ chúng ta thảo luận về phân phối các thuật toán phát hiện bế tắc Chandy-Misra-Haas cho các mô hình OR [6], mà là dựa trên cách tiếp cận của khuếch tán tính toán (xem Algorithm 10.2).

Một quá trình bị chặn xác định nếu nó được bế tắc bằng cách khởi xướng một tính toán khuếch tán. Hai loại thông điệp được sử dụng trong một tính toán khuếch tán:

truy vấn (i, j, k) và trả lời (i, j, k), biểu thị rằng chúng thuộc về một tính toán khuếch tán khởi xướng bởi một quá trình Pi và đang được gửi từ quá trình Pj để xử lý Pk.

Ý tưởng nền tảng

Một quá trình bị chặn khởi phát hiện bế tắc bằng cách gửi thông điệp truy vấn cho tất cả các quá trình trong tập hợp phụ thuộc của nó (tức là, các quy trình mà từ đó nó đang chờ đợi để nhận được một tin nhắn). Nếu một quá trình hoạt động nhận được một câu hỏi hoặc trả lời tin nhắn, nó loại bỏ nó. Khi một tiến trình bị chặn P_k nhận được một truy vấn (i, j, k) tin nhắn, nó có các hành động sau đây:

1. Nếu đây là thông điệp truy vấn đầu tiên nhận được P_k cho các phát hiện bế tắc khởi xướng bởi Pi (gọi là truy vấn hấp dẫn), sau đó nó truyền các truy vấn cho tất cả các quá trình trong tập hợp phụ thuộc của nó và đặt một biến địa phương *number(i)* với số lượng tin nhắn gửi truy vấn.
2. Nếu đây không phải là truy vấn hấp dẫn, sau đó Pk trả về một tin nhắn trả lời nó ngay lập tức cung cấp P_k đã liên tục bị chặn kể từ khi nó nhận được các truy vấn tham gia tương ứng. Nếu không, nó loại bỏ các truy vấn.

Quy trình P_k duy trì một biến Boolean $wait_k(i)$ biểu thị một thực tế là nó đã được liên tục bị chặn kể từ khi nó nhận được các truy vấn cuối cùng hấp dẫn từ quá trình P_i. Khi một tiến trình bị chặn P_k nhận được một thư trả lời (i, j, k) thông báo, nó giảm $number_{k\neg}(i)$ chỉ khi nếu $wait_k(i)$ nắm giữ. Một quá trình gửi một tin nhắn trả lời để đáp ứng với một truy vấn tham gia chỉ sau khi nó đã nhận được trả lời mọi tin nhắn truy vấn đã được gửi ra cho truy vấn hấp dẫn này.

Quá trình khởi đầu phát hiện một bế tắc khi nó đã nhận được tin nhắn trả lời cho tất cả những thông điệp truy vấn nó đã gửi đi.

Thuật toán:

Các thuật toán làm việc như trong thuật toán 10.2. Để dễ trình bày, chúng ta đã giả định rằng chỉ có một tính toán khuếch tán được bắt đầu cho một quá trình. Trong thực tế, một số tính toán khuếch tán có thể được bắt đầu cho một quá trình (một tính toán khuếch tán được bắt đầu mỗi lần quá trình bị chặn), nhưng bất cứ lúc nào chỉ có một khuếch tán tính là hiện tại cho bất kỳ quá trình. Tuy nhiên, các thông điệp cho các tính toán khuếch tán lỗi thời vẫn có thể quá cảnh. Việc tính toán khuếch tán hiện tại có thể được phân biệt với những người lỗi thời bằng cách sử dụng số thứ tự.

Initiate a diffusion computation for a blocked process P_i:
send *query*(i, i, j) to all processes P_j in the dependent set DS_i of P_i;
$num_i(i) := |DS_i|$; $wait_i(i) := true$;

When a blocked process P_k receives a *query*(i, j, k):
if this is the engaging *query* for process P_i then
send *query*(i, k, m) to all P_m in its dependent set DS_k;
$num_k(i) := |DS_k|$; $wait_k(i) := true$
else if $wait_k(i)$ then send a *reply*(i, k, j) to P_j.

When a process P_k receives a *reply*(i, j, k):
if $wait_k(i)$ then
$num_k(i) := num_k(i) - 1$;
if $num_k(i) = 0$ then
if $i = k$ then **declare a deadlock**
else send *reply*(i, k, m) to the process P_m
which sent the engaging query.

Thuật toán 10.2: Chandy-Misra-Haas cho mô hình OR.

Khởi xướng một tính toán khuếch tán cho một quá trình Pi bị chặn
gửi truy vấn (i, i, j) cho tất cả các quá trình Pj trong tập hợp phụ thuộc DSi của Pi;

Khi một tiến trình bị chặn P_k nhận được một truy vấn (i, j, k):
nếu điều này là truy vấn hấp dẫn cho quá trình Pi sau đó
gửi truy vấn (i, k, m) cho tất cả Pm trong tập hợp phụ thuộc của nó DS_k;

Khi một quá trình Pk nhận được một thư trả lời (i, j, k):
Ngược lại gửi trả lời (i, k, m) đến quá trình Pm
mà gửi các truy vấn hấp dẫn.

Phân tích hiệu suất:
Đối với mỗi phát hiện bế tắc, các thuật toán trao đổi e thông điệp truy vấn và trả lời e tin nhắn, trong đó e = n (n-1) là số cạnh.