MỘT GIẢI PHÁP PHÁT HIỆN CÁC SỰ KIỆN SONG SONG TRONG CÁC TIẾN TRÌNH CỦA ỨNG DỤNG PHÂN TÁN

Nguyễn Hoàng Hà

Khoa Công nghệ thông tin, Trường Đại học Khoa học, Đại học Huế

Email: nhha@husc.edu.vn, nhha76@gmail.com

Ngày nhận bài: 27/4/2020; ngày hoàn thành phản biện: 4/5/2020; ngày duyệt đăng: 14/7/2020

TÓM TẮT

Khác với hệ thống tập trung, dữ liệu và các chức năng trên hệ phân tán được lưu trữ trên các máy tính thuộc các vùng địa lý khác nhau và tại một thời điểm có nhiều công việc được thực hiện một cách đồng thời. Vì vậy, làm sao để phát hiện các sự kiện song song trong các tiến trình nhằm tối ưu thời gian thực hiện của hệ thống là một thách thức lớn. Trước đây, người ta sử dụng thời gian thực để phát hiện ra các sự kiện song song. Khi truyền nhận dữ liệu giữa các nút trên hệ phân tán, thời gian thực có độ trễ lớn nên độ chính xác không cao. Bài báo này nghiên cứu đồng bộ hóa thời gian logic trong hệ phân tán nhằm phát hiện ra các sự kiện có thể thực hiện song song trong các ứng dụng phân tán.

Từ khóa: đồng bộ hóa thời gian, xử lý phân tán, thuật toán Lamport, thuật toán Vector Clock.

1. MỞ ĐẦU

Hệ phân tán là một hệ thống có chức năng và dữ liệu phân tán trên các trạm (máy tính) được kết nối với nhau bởi một mạng máy tính [2].

Trong hệ phân tán, dữ liệu và các chức năng được lưu trữ trên các máy tính ở các vị trí địa lý khác nhau và nhiều công việc có thể thực hiện đồng thời. Vì vậy, hiện nay hệ phân tán gặp một số thách thức về đồng bộ như sau: làm sao đồng bộ về thời gian trong hệ thống, trong khi mỗi quốc gia có các múi giờ khác nhau; tại một thời điểm có thể có nhiều tiến trình cộng tác cùng nhau; các sự kiện trên các tiến trình cùng trao đổi thông tin với nhau. Vì vậy, làm sao để xác định được sự kiện nào trên mỗi tiến trình có thể thực hiện đồng thời với sự kiện của tiến trình khác. Đây là một thách thức rất lớn.

Để giải quyết vấn đề này, Gusella và Zatti tại Đại học California và Flaviu Cristian đã đưa ra giải thuật Berkeley và Cristian để đồng bộ thời gian thực [5]. Cả hai

giải thuật này sử dụng đồng hồ UTC làm mốc thời gian để đồng bộ. Nhưng cả hai thuật toán chỉ áp dụng trong các mạng nội bộ có độ trễ thấp, nếu sử dụng trong mạng diện rộng thì độ chính xác không cao vì độ trể thời gian lớn.

Năm 2016, Đặng Hồng Vỹ [1] đã sử dụng thuật toán Lamport để đồng bộ hóa thời gian logic trên hệ phân tán. Nghiên cứu này chỉ xác định được quan hệ từng phần của các tiến trình, còn nhiều trường hợp chưa xác định tiến trình nào xảy ra trước, tiến trình nào xảy ra sau cũng như chưa xác định được các sự kiện nào có thể xảy ra đồng thời.

Yan Cai ; W.K. Chan [3] đã sử dụng thuật toán Vector Clock để đồng bộ thời gian logic. Nghiên cứu này đã xác định được quan hệ từng phần, quan hệ trước sau của các tiến trình nhưng chưa xác định các sự kiện xảy ra đồng thời.

Bài báo này nghiên cứu thuật toán Vector Clock [4], từ đó đưa ra mô hình, giải thuật và cài đặt thử nghiệm nhằm đưa ra các sự kiện có thể xảy ra đồng thời. Từ đó xác định được các sự kiện nào có thể cài đặt song song với nhau nhằm tối ưu thời gian của hệ thống.

2. MÔ HÌNH HỆ THỐNG

a. Mô hình của các tiến trình

Một hệ thống xử lý phân tán gồm *n* tiến trình, ký hiệu: P={p_1, p_2, ..., p_n}.

Trên mỗi tiến trình gồm m sự kiện xảy ra trên tiến trình đó, gọi E_i là sự kiện xảy ra trên tiến trình i (i=1..n), ký hiệu E_i={e_{i1}, e_{i2}…, e_{im}}, trong đó $e_{ix} \in E_i$ là sự kiện x (x=1..m) xảy ra trên tiến trình i. Các sự kiện trên tiến trình này có thể trao đổi thông tin với sự kiện trên tiến trình khác. Mỗi sự kiện duy trì một nhãn thời gian [1].

b. Quan hệ "xảy ra trước"

Cho e_{ix} (i=1 … n, x=1…m), e_{jy} (j=1 … n, y=1..m) là hai sự kiện trong một hệ thống xử lý phân tán và ký hiệu: → là quan hệ "xảy ra trước"[1].

- Nếu e_{ix} và e_{jy} xảy ra trên cùng một tiến trình (i=j), và e_{ix} đến trước e_{jy}, thì: $e_{ix} \rightarrow e_{jy}$.

- Nếu e_{ix} là việc gửi gói tin từ tiến trình Pi, và e_{jy} là việc nhận gói tin đó ở một tiến trình Pj, thì $e_{ix} \rightarrow e_{jy}$.

- Nếu $e_{ix} \rightarrow e_{jy}$ và $e_{jy} \rightarrow e_{it}$ (t=1..m) thì $e_{ix} \rightarrow e_{it}$.

c. Nhãn thời gian

Giả sử mỗi tiến trình đều có một đồng hồ C, với C là một hàm số sinh ra nhãn thời gian (timestamp). Đồng hồ này không nhất thiết phải liên quan đến đồng hồ vật

lý. Mỗi sự kiện $e_{ix} \in E_i$ trên tiến trình đều được C gán cho một con số timestamp tương đương $C(e_{ix})$. Bộ đếm $C(e_{ix})$ luôn được tăng trước mỗi sự kiện trong tiến trình. Đối với mỗi sự kiện e_{ix}, e_{jy} bất kỳ ta luôn có: nếu $e_{ix} \rightarrow e_{jy}$ thì $C(e_{ix}) < C(e_{jx})$[1].

Ta định nghĩa lại tình trạng của đồng hồ dựa trên quan hệ "xảy ra trước" bao gồm hai điều kiện như sau:

Nếu e_{ix} và e_{iy} là hai sự kiện trong một tiến trình $p_i \in P$ và $e_{ix} \rightarrow e_{jy}$ thì $C(e_{ix}) < C(e_{jy})$.

Nếu e_{ix} là sự kiện gửi gói tin từ tiến trình $p_i \in P$ và e_{jy} là sự kiện nhận gói tin đó thì $C(e_{ix}) < C(e_{jy})$.

d. Giải pháp trật tự từng phần

Trong các ứng dụng của hệ phân tán, dựa vào quan hệ "xảy ra trước" ta có thể xác định được trật tự từng phần giữa các sự kiện. Trật tự này thỏa mãn điều kiện:

Nếu e_{ix} và e_{jy} là hai sự kiện của cùng một trạm và nếu e_{ix} thực hiện trước e_{jx} thì $e_{ix} \rightarrow e_{jx}$.

Nếu e_{ix} phát thông điệp bởi một trạm nào đó và e_{jx} thu thông điệp này thì $e_{ix} \rightarrow e_{jx}$.

e. Đồng hồ vector

Nếu nhãn thời gian là một số nguyên ta có thể kết luận: nếu $e_{ix} \rightarrow e_{jy}$ thì $C(e_{ix}) < C(e_{jx})$. Nếu $C(e_{ix}) < C(e_{jx})$ chưa chắc a xảy ra trước b, và không thể rút ra quan hệ phụ thuộc nhân quả từ các nhãn thời gian vì có thể hai sự kiện e_{ix}, e_{jy} xảy ra đồng thời. Để khắc phục nhược điểm này ta sử dụng đồng hồ vector [3] .

Mỗi tiến trình sử dụng 1 vector gồm n thành phần chứa các số tự nhiên (nhãn thời gian): Pi duy trì 1 vector: Vi[1,…,n]

Vi[i]: chứa nhãn thời gian của tiến trình i. Ví dụ V1={1,0,0} thì V1[1]=1, V1[2]=0, V1[3]=0;

Vi[j]: chứa nhãn thời gian của tiến trình j xảy ra tại tiến trình i.

3. THUẬT TOÁN VECTOR CLOCK

Ý tưởng của thuật toán

Nếu chúng ta sử dụng nhãn thời gian là một số nguyên thì sẽ không xác định được hai sự kiện có thể xảy ra đồng thời hay không. Đồng hồ vector giải quyết vấn đề này bằng cách sử dụng một bộ đếm vector thay vì bộ đếm là một số nguyên. Đồng hồ vector của một hệ thống có n tiến trình là vector của n bộ đếm, mỗi một sự kiện trên

tiến trình đều duy trì một vector. Khi các tiến trình trao đổi với nhau thì giá trị của các vector này thay đổi [3].

Thuật toán Vector Clock

Đầu vào:

- $P=\{p_1, p_2, ..., p_n\}$;
- $E_i=\{e_{i1}, e_{i2}..., e_{im}\}$;
- Tập các vector Vi (i=1..n);

Đầu ra: Tập vector Vi chứa nhãn thời gian của các sự kiện trên mỗi tiến trình.

Thuật toán:

1.	Khởi tạo: Vi[j]=0, i=1..n, j=1..n.
2.	Foreach $p_i \in P$ do
3.	Foreach $e_{ix} \in E_i$ do
4.	If Pi gửi 1 sự kiện e_{ix} đến Pj then
5.	Pi sẽ thay đổi giá trị Vi[i]=Vi[i]+1 ;
6.	Pi gửi thông báo kèm theo Vi[i] đến Pj;
7.	Khi Pj nhận sự kiện từ Pi, Pj sẽ cập nhật lại Vj:
8.	Vj[k]=max(Vj[k],Vi[k]), k=1..n, j<>k;
9.	Vj[j]=Vj[j]+1;

Phân tích độ phức tạp của thuật toán Vector Clocks.

- Để đưa ra nhãn thời gian cho mỗi sự kiện của tiến trình ta phải duyệt qua các n tiến trình nên độ phức tạp: O(n).

- Mỗi tiến trình ta phải duyệt qua các sự kiện để xác định sự kiện gửi, sự kiện nhận nên độ phức tạp: O(M) với M là số sự kiện lớn nhất của các tiến trình.

- Với mỗi sự kiện ta phải duyệt qua các phần tử trong vector nên độ phức tạp: O(n).

Như vậy, độ phức tạp của thuật toán Vector Clocks là O(n)*O(M)*O(n) =$O(n^2*M)$.

Đầu ra của thuật toán Vector Clock là tập các vector Vi chứa nhãn thời gian của các sự kiện trên mỗi tiến trình. Bài báo này dựa vào đầu ra của thuật toán Vector Clock để phân tích tìm ra sự kiện nào xảy ra trước, sự kiện nào xảy ra sau và xác định các sự kiện nào xảy ra đồng thời. Từ đó, xác định trong các tiến trình sự kiện nào có thể thực

hiện song song để tối ưu thời gian thực hiện của hệ thống. Để giải quyết vấn đề này ta cần tìm ra quan hệ nhân quả giữa các sự kiện trong các tiến trình.

4. QUAN HỆ NHÂN QUẢ GIỮA CÁC SỰ KIỆN

Dựa trên tập các vector Vi chứa nhãn thời gian của các sự kiện trên mỗi tiến trình. Ta xác định được:

- Hai vector bằng nhau nếu mỗi thành phần tương ứng trong 2 vector bằng nhau:

Vi=Vj, nếu Vi[k]=Vj[k], $\forall$ k=1,…,n

- Vi≤Vj, nếu Vi[k] ≤Vj[k], $\forall$ k=1,…,n.

- Sự kiện có nhãn thời gian Vi xảy ra trước sự kiện có nhãn thời gian Vj nếu Vi<Vj, Vi<Vj nếu thỏa mãn hai điều kiện: Vi≤Vj và $\exists$k để Vi[k]<Vj[k].

Như vậy, sự kiện có nhãn thời gian Vi xảy ra đồng thời với sự kiện có nhãn thời gian Vj nếu: Not(Vi≤Vj) và Not(Vj≤Vi).

5. THUẬT TOÁN PVectorClock

Dựa trên mô hình ở phần 4, bài báo xây dựng thuật toán PVectorClock để xác định được các sự kiện xảy ra đồng thời giữa các tiến trình.

Thuật toán PVectorClock:

Đầu vào: Tập các vector Vi là đầu ra của thuật toán Vector Clock;

Đầu ra: Tập S chứa các vector xảy ra đồng thời;

Thuật toán:

```
1.  Function isconcurrent(v[], w[]) //Hàm kiểm tra v,w xảy ra đồng thời hay không
2.  Begin
2.      greater:=false, less:=false;
3.      for i:=0 to n do
4.          if v[i] > w[i] then
5.              greater := true;
6.          else
7.           if v[i] < w[i] then
8.              less = true;
```

```
9.          if greater=true and less=true then
10.                 return true; // v và w là đồng thời
            else
                    return false; // v và w là không đồng thời
11.   End;
12.   S=∅;
13.   Foreach pi ∈ P do
14.         Si=∅;
15.         Foreach pj ∈ P do
16.                 If isconcurrent(Vi, Vj) then
17.                         Si=Si ∪Vj
18.         S=S ∪ Si
```

Phân tích thuật toán độ phức tạp của thuật toán PVectorClock.

- Để tìm ra các vector xảy ra đồng thời ta phải duyệt qua các n tiến trình nên độ phức tạp: O(n).

- Mỗi tiến trình ta phải duyệt qua các tiến trình có trao đổi dữ liệu với nhau nên độ phức tạp: O(n)

- Để kiểm tra hai sự kiện có xảy ra đồng thời hay không, ta phải phải duyệt qua các phần tử trong vector nên độ phức tạp: O(n).

Như vậy, độ phức tạp của thuật toán PVectorClock là O(n)*O(n)*O(n) =O(n^3).

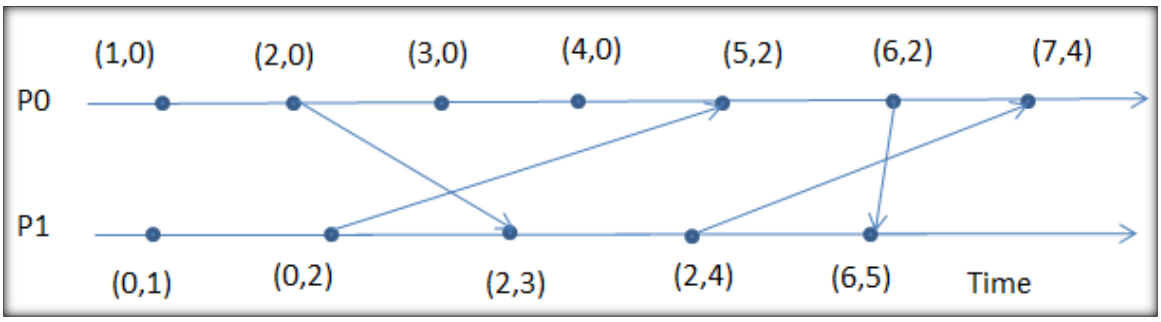
Cài đặt và mô phỏng thuật toán PVectorClock

Thuật toán được cài đặt trên ngôn ngữ lập trình Java (NetBean 8.2, JDK 1.8); hệ điều hành Window 7 Ultimate; bộ xử lý Intel(R) Core ™ i5-4200U CPU @ 1.60 GHz 2.30 GHz; RAM: 4.00Gb.

Phần này sẽ mô phỏng thuật toán trên 2 trường hợp, trường hợp 1 mô phỏng trên 2 tiến trình, 12 sự kiện. Trường hợp 2 mô phỏng trên 4 tiến trình và 19 sự kiên.

Trường hợp 1:

- Thuật toán Vector Clock được mô phỏng trên 02 tiến trình, tiến trình 1 có 7 sự kiện, tiến trình 2 có 5 sự kiện, mỗi quan hệ giữa các sự kiện và kết quả đầu ra của thuật toán Vector Clock được thể hiện như Hình 1.



Hình 1. Kết quả của Thuật toán Vector Clock cho 2 tiến trình.

- Sau khi mô phỏng, thuật toán PvectorClock đã xác định được tập các sự kiện song song trên mỗi tiến trình, cụ thể như sau:

Sự kiện có nhãn thời gian [1, 0] có thể thực hiện song song với các sự kiện có nhãn thời gian [0,1], [0,2]. Sự kiện có nhãn thời gian [5, 2] có thể thực hiện song song với các sự kiện có nhãn thời gian [2,3], [2,4], …

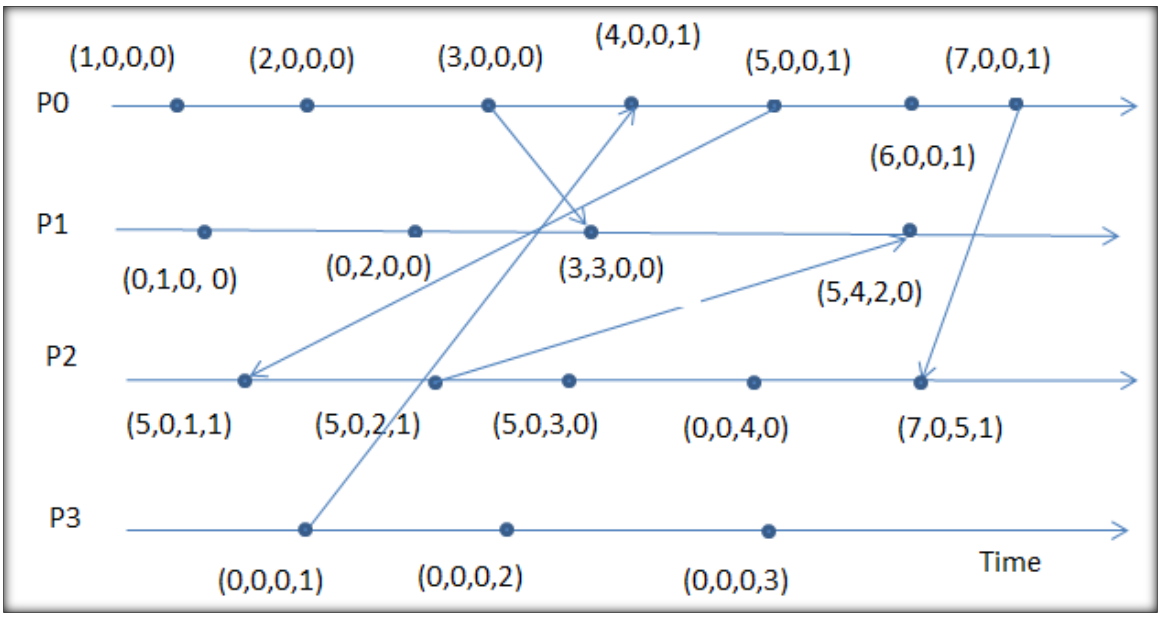
Tập các sự kiện có thể thực hiện song song với nhau được thể hiện ở Hình 2.

Thứ tự sự kiện có nhãn thời gian:		**Xảy ra đồng thời với các sự kiện có nhãn thời gian:**			
1	[1 0]	[0 1]	[0 2]		
2	[2 0]	[0 1]	[0 2]		
3	[3 0]	[0 1]	[0 2]	[2 3]	[2 4]
4	[4 0]	[0 1]	[0 2]	[2 3]	[2 4]
5	[5 2]	[2 3]	[2 4]		
6	[6 2]	[2 3]	[2 4]		
7	[7 4]	[6 5]			
8	[0 1]	[1 0]	[2 0]	[3 0]	[4 0]
9	[0 2]	[1 0]	[2 0]	[3 0]	[4 0]
10	[2 3]	[3 0]	[4 0]	[5 2]	[6 2]
11	[2 4]	[3 0]	[4 0]	[5 2]	[6 2]
12	[6 5]	[7 4]			

Hình 2. Tập các sự kiện có thể thực hiện song song với nhau trên 2 tiến trình

Trường hợp 2:

- Thuật toán Vector Clock được mô phỏng trên 04 tiến trình, số sự kiện trên mỗi tiến trình lần lượt là: 7, 4, 5, 3. Mối quan hệ giữa các sự kiện và kết quả đầu ra của thuật toán Vector Clock được thể hiện như Hình 3.



Hình 3. Kết quả của Thuật toán Vector Clock cho 4 tiến trình

- Trên mỗi sự kiện của tiến trình, thuật toán PvectorClock đã xác định được tập các sự kiện xảy ra song song với nó, cụ thể như sau:

Trên tiến trình 1 (P_0), sự kiện có nhãn thời gian [1,0,0,0] có thể thực hiện song song với các sự kiện có nhãn thời gian [0,1,0,0], [0,2,0,0], [0,0,4,0], [0,0,0,1], [0,0,0,2], [0,0,0,3]

Trên tiến trình 2 (P_1), sự kiện có nhãn thời gian [0,1,0,0] có thể thực hiện song song với các sự kiện có nhãn thời gian [1,0,0,0], [2,0,0,0], [3,0,0,0], [4,0,0,1], …

Kết quả mô phỏng trong Trường hợp 2 được thể hiện như Hình 4, trong đó mỗi sự kiện trên mỗi tiến trình đều xác định được tập các sự kiện xảy ra đồng thời với nó.

Trong trường hợp tổng quát thuật toán PvectorClock có thể mô phỏng cho bất kỳ tập các tiến trình nào, miễn sao có được tập các nhãn thời gian trên mỗi sự kiện của mỗi tiến trình và tập các mối quan hệ giữa các sự kiện.

Như vậy, thuật toán PvectorClock đã xác định được tập các sự kiện xảy ra đồng thời. Dựa vào tập này nhà phát triển ứng dụng phân tán có thể cho thực hiện song song các sự kiện nhằm tối ưu thời gian thực hiện cho cả hệ thống.

Thứ tự sự kiện có nhãn thời gian:		Xảy ra đồng thời với các sự kiện có nhãn thời gian:					
1	[1 0 0 0]	[0 1 0 0]	[0 2 0 0]	[0 0 4 0]	[0 0 0 1]	[0 0 0 2]	[0 0 0 3]
2	[2 0 0 0]	[0 1 0 0]	[0 2 0 0]	[0 0 4 0]	[0 0 0 1]	[0 0 0 2]	[0 0 0 3]
3	[3 0 0 0]	[0 1 0 0]	[0 2 0 0]	[0 0 4 0]	[0 0 0 1]	[0 0 0 2]	[0 0 0 3]
4	[4 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]	[5 0 3 0]	[0 0 4 0]
		[0 0 0 1]	[0 0 0 2]	[0 0 0 3]			
5	[5 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]	[5 0 3 0]	[0 0 4 0]
		[0 0 0 2]	[0 0 0 3]				
6	[6 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]	[5 0 1 1]	[5 0 2 1]
		[5 0 3 0]	[0 0 4 0]	[0 0 0 2]	[0 0 0 3]		
7	[7 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]	[5 0 1 1]	[5 0 2 1]
		[5 0 3 0]	[0 0 4 0]	[0 0 0 2]	[0 0 0 3]		
8	[0 1 0 0]	[1 0 0 0]	[2 0 0 0]	[3 0 0 0]	[4 0 0 1]	[5 0 0 1]	[6 0 0 1]
		[7 0 0 1]	[5 0 1 1]	[5 0 2 1]	[5 0 3 0]	[0 0 4 0]	[7 0 5 1]
		[0 0 0 1]	[0 0 0 2]	[0 0 0 3]			
9	[0 2 0 0]	[1 0 0 0]	[2 0 0 0]	[3 0 0 0]	[4 0 0 1]	[5 0 0 1]	[6 0 0 1]
		[7 0 0 1]	[5 0 1 1]	[5 0 2 1]	[5 0 3 0]	[0 0 4 0]	[7 0 5 1]
		[0 0 0 1]	[0 0 0 2]	[0 0 0 3]			
10	[3 3 0 0]	[4 0 0 1]	[5 0 0 1]	[6 0 0 1]	[7 0 0 1]	[5 0 1 1]	[5 0 2 1]
		[5 0 3 0]	[0 0 4 0]	[7 0 5 1]	[0 0 0 1]	[0 0 0 2]	[0 0 0 3]
11	[5 4 2 0]	[4 0 0 1]	[5 0 0 1]	[6 0 0 1]	[7 0 0 1]	[5 0 1 1]	[5 0 2 1]
		[5 0 3 0]	[0 0 4 0]	[7 0 5 1]	[0 0 0 1]	[0 0 0 2]	[0 0 0 3]
12	[5 0 1 1]	[6 0 0 1]	[7 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]
		[5 0 3 0]	[0 0 4 0]	[0 0 0 2]	[0 0 0 3]		
13	[5 0 2 1]	[6 0 0 1]	[7 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]
		[5 0 3 0]	[0 0 4 0]	[0 0 0 2]	[0 0 0 3]		
14	[5 0 3 0]	[1 0 0 0]	[2 0 0 0]	[3 0 0 0]	[4 0 0 1]	[5 0 0 1]	[6 0 0 1]
		[7 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]	[5 0 1 1]
		[5 0 2 1]	[0 0 4 0]	[0 0 0 1]	[0 0 0 2]	[0 0 0 3]	
15	[0 0 4 0]	[1 0 0 0]	[2 0 0 0]	[3 0 0 0]	[4 0 0 1]	[5 0 0 1]	[6 0 0 1]
		[7 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]	[5 0 1 1]
		[5 0 2 1]	[5 0 3 0]	[0 0 0 1]	[0 0 0 2]	[0 0 0 3]	
16	[7 0 5 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]	[0 0 0 2]	[0 0 0 3]
17	[0 0 0 1]	[1 0 0 0]	[2 0 0 0]	[3 0 0 0]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]
		[5 4 2 0]	[5 0 3 0]	[0 0 4 0]			
18	[0 0 0 2]	[1 0 0 0]	[2 0 0 0]	[3 0 0 0]	[4 0 0 1]	[5 0 0 1]	[6 0 0 1]
		[7 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]	[5 0 1 1]
		[7 0 5 1]	[5 0 2 1]	[5 0 3 0]	[0 0 4 0]		
19	[0 0 0 3]	[1 0 0 0]	[2 0 0 0]	[3 0 0 0]	[4 0 0 1]	[5 0 0 1]	[6 0 0 1]
		[7 0 0 1]	[0 1 0 0]	[0 2 0 0]	[3 3 0 0]	[5 4 2 0]	[5 0 1 1]
		[5 0 2 1]	[5 0 3 0]	[0 0 4 0]	[7 0 5 1]		

Hình 4. Tập các sự kiện có thể thực hiện song song với nhau trên 4 tiến trình

6. KẾT LUẬN

Bài báo tập trung nghiên cứu các giải pháp để đồng bộ hóa thời gian logic, từ đó xây dựng mô hình toán học cho các thành phần trong hệ phân tán. Dựa vào kết quả đầu ra của thuật toán Vector Clock, bài báo đã phân tích và xây dựng thuật toán PvectorClock nhằm xác định tập các sự kiện song song trên các tiến trình. Thông qua việc phân tích và đánh giá kết quả mô phỏng cho thấy thuật toán PvectorClock đã xác định được tập các sự kiện xảy ra đồng thời với nhau. Dựa vào kết quả này các nhà phát triển ứng dụng trên hệ phân tán có thể thực hiện song song các sự kiện với nhau, khi đó sẽ tối ưu tổng thời gian thực hiện của hệ thống.

TÀI LIỆU THAM KHẢO

[1]. Đặng Hùng Vĩ, Lê Văn Sơn (2016), *Giải pháp cung cấp tài nguyên truyền thông phân tán*, Kỷ yếu Hội nghị Quốc gia lần thứ VIII về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR); Hà Nội, ngày 9-10/7/2015. DOI: 10.15625/vap.2015.000159.

[2]. G. Coulouris, J. Dollimore, T. Kinberg, G. Blair, (2012) , *Distributed systems : Concept and Design*, 5th Edition, Addison-Wesley.

[3]. Yan Cai ; W.K. Chan (2012), *LOFT: Redundant Synchronization Event Removal for Data Race Detection*, IEEE, **DOI**: 10.1109/ISSRE.2011.12.

[4]. D.A. Khotimsky ; I.A. Zhuklinets (2003), *Hierarchical vector clock: scalable plausible clock for detecting causality in large distributed systems*, IEEE, 10.1109/ICATM.1999.786798.

[5]. A. S. Tanenbaum, M. V. Steen (2007), *Distributed Systems: Principles and Paradigms* , 2nd Edition, Prentice-Hall.

A SOLUTION TO DETECT PARALLEL EVENTS IN THE PROCESS OF DISTRIBUTION APPLICATIONS

Nguyen Hoang Ha

Faculty of Information Technology, University of Sciences, Hue University

Email: nhha@husc.edu.vn, nhha76@gmail.com

ABSTRACT

Unlike centralized systems, data and functions on distributed systems are stored on computers in different geographic locations and many jobs are solved simultaneously. Therefore, how to detect parallel events in execution processes to optimize system performance time is a big challenge. In the past, people used real time to discover parallel events. On distributed systems, when transmitting data between nodes, the real time has a large delay that leads to the accuracy is not high. This paper investigates logical time synchronization in a distributed system. The purpose of this study is to discover events that can be performed in parallel in distributed applications.

Keywords: Lamport algorithm, time synchronization, distributed processing, VectorClock algorithm.

Nguyễn Hoàng Hà sinh ngày 22/11/1976 tại Quảng Nam. Năm 1999, ông tốt nghiệp đại học ngành Công nghệ Thông tin tại trường Đại học Khoa học, ĐH Huế. Năm 2005, ông nhận bằng thạc sỹ Khoa học Máy tính tại Trường Đại học Khoa học, ĐH Huế. Năm 2017, ông tốt nghiệp tiến sĩ chuyên ngành Khoa học máy tính tại trường Đại học Khoa học, ĐH Huế. Hiện ông công tác tại Trường Đại học Khoa học, Đại học Huế.

Lĩnh vực nghiên cứu: Xử lý song song và phân tán, tính toán lưới và tính toán đám mây.