Tạp chí Tin học và Điều khiển học, T. 18, S. 1 (2002), 22–28

THUẬT TOÁN QUI HOẠCH ĐỘNG CHO BÀI TOÁN LẬP LỊCH TỐI ƯU TRONG CƠ SỞ DỮ LIỆU SONG SONG

NGUYỄN XUÂN HUY, NGUYỄN MẬU HÂN

Abstract. This paper suggests an algorithm to define an optimization schedule for the pipelined operator trees in multiprocessor computing system in which the dynamic distribution method will be considered.

Tóm tắt. Bài báo này đưa ra một phương pháp tìm kiếm lịch truy vấn tối ưu cho cây toán tử dạng ống trong môi trường đa xử lí bằng cách sử dụng phương pháp qui hoạch động.

1. GIỚI THIỆU

Khi một câu truy vấn SQL được chuyển đến, bộ tối ưu của các hệ quản trị cơ sở dữ liệu trước tiên phải tiến hành sắp xếp các phép toán theo các chiến lược tối ưu định sẵn để thời gian trả lời truy vấn là ít nhất. Trong môi trường đa xử lí, ngoài việc sắp xếp một cách hợp lý các phép toán, bộ tối ưu còn phải giải quyết tiếp bài toán lập lịch tối ưu nghĩa là tìm một kế hoạch phân công các công việc cho các bộ xử lí để thời gian hoàn thành là ngắn nhất. Bài toán lập lịch tối ưu cho cây truy vấn là bài toán NP-Khó [9], nhiều tác giả đã giải quyết bài toán này bằng cách đưa bài toán về dạng đơn giản hơn, có độ phức tạp đa thức, bằng cách thực hiện các phép gộp các nút (collapse) và xóa các cạnh (cut) để chuyển một cây toán tử phức tạp thành cây toán tử đơn điệu [8], sau đó sẽ tìm một phân hoạch liên thông tối ưu cho các nút của cây toán tử và chuyển các cây con vào các bộ xử lí tương ứng. Trong bài báo này chúng tôi đề xuất cách tìm kiếm lịch truy vấn tối ưu cho cây toán tử dạng ống (pipeline operator tree) bằng phương pháp qui hoạch động và các kết quả của lí thuyết đồ thị hữu hạn. Giả sử $T = (V, E)$ là cây toán tử dạng ống, ta có thể xem T là một đồ thị có hướng, không khuyên, liên thông, có trọng số, các toán tử là các nút của cây, các cạnh với chi phí truyền thông của cây là các cung với trọng số tương ứng. Việc xác định lịch tối ưu cho cây toán tử đã cho đồng nghĩa với việc tìm một phân hoạch các nút của cây $F_1, \ldots, F_p$, với tập F_k là các nút được phân cho bộ xử lí thứ k, sao cho $L = \max\limits_{1 \le i \le p} \text{cost}(F_i)$ đạt cực tiểu.

2. MỘT SỐ ĐỊNH NGHĨA VÀ KHÁI NIỆM LIÊN QUAN

Định nghĩa 1.

• *Cây truy vấn chú giải* (annotated query tree) là cây truy vấn cho biết thứ tự thực hiện mỗi phép toán và phương pháp tính toán mỗi toán tử. Mỗi nút trên cây đại diện cho một (hay nhiều) phép toán quan hệ. Những ghi chú trên mỗi nút mô tả cách nó được thực hiện chi tiết như thế nào.

• *Cây toán tử* (operator tree) dùng để mô tả các phép toán song song để thực hiện cây truy vấn tương ứng cũng như các ràng buộc về thời gian giữa chúng. Trường hợp các toán tử trên cây là toán tử dạng ống (pipelined operator) thì ta có cây toán tử dạng ống [8].

Định nghĩa 2. Cho p bộ xử lí và cây toán tử $T = (V, E)$, trong đó V là tập các nút, E là tập các cạnh. Lịch truy vấn của T là một phân hoạch từ các nút thành p tập $F_1, \ldots, F_p$ với tập F_k là các nút được phân cho bộ xử lí thứ k.

Định nghĩa 3. Phát biểu bài toán lập lịch cây toán tử dạng ống:

Input: Cây toán tử $T = (V, E)$; t_i là trọng số của nút thứ i; c_{ij} là trọng số của cạnh $(i, j) \in E$; p là

số bộ xử lí.

Output: Một lịch truy vấn với thời gian trả lời cực tiểu. Nghĩa là, một phép phân hoạch V thành các tập $F_1, \ldots, F_p$ sao cho $\max_{1\le k\le p} \sum_{i\in F_k}(t_i + \sum_{j\notin F_k} c_{ij})$ là cực tiểu.

Thời gian trả lời L của một lịch truy vấn được tính từ thời gian các toán tử dạng ống khởi động đồng thời cho đến toán tử cuối cùng hoàn tất công việc. Khi đó các toán tử thực hiện nhanh phải "đợi" các toán tử thực hiện chậm.

Định nghĩa 4. Nếu F là một tập toán tử thì chi phí tải của một bộ xử lí để thực hiện F được xác định bởi $\text{cost}(F) = \sum_{i\in F_k}(t_i + \sum_{j\notin F_k} c_{ij})$.

3. TÌM LỊCH TRUY VẤN TỐI ƯU BẰNG PHƯƠNG PHÁP LẬP TRÌNH ĐỘNG

Dựa vào tư tưởng lập trình và thuật toán xác định xác định truy vấn bằng đường tăng luồng sẽ được mô tả dưới đây, chúng ta sẽ xây dựng thuật toán tổng hợp bằng phương pháp lập trình động, có độ phức tạp đa thức, để xác định lịch truy vấn tối ưu cho cây toán tử dạng ống.

3.1. Thuật toán phân phối luồng [6,7]

Lịch truy vấn của một cây toán tử với p bộ vi xử lí là một phân hoạch $F_1, \ldots, F_p$ gồm các nút của cây toán tử, trong đó F_i chứa các nút được định vị cho bộ xử lí thứ i. Thời gian trả lời L của lịch truy vấn là khoảng thời gian mà một bộ xử lí nào đó thực hiện công việc của mình chậm nhất, nghĩa là $L = \max_{1\le i\le p} \cos t(F_i)$, đây chính là hàm đích mà ta cần điều chỉnh để nó tiến về giá trị bé hơn nếu có thể. Trong tất cả các chỉ số $i \in \{1, \ldots, p\}$ mà $\text{cost}(F_i)$ đạt max, ta sẽ chọn một giá trị i^* nào đó mà tồn tại một dãy $i_1, j_1, \ldots, i_r, j_r, i_{r+1}$ thỏa các điều kiện sau:

- $i_1 = i^*$.
- $j_k \in F_{i_k}$, $1 \le k \le r$.
- $\text{cost}(F_i \cup \{j_{k-1} \mid i_k = i,\ 2 \le k \le r+1\} \setminus \{j_k \mid i_k = i,\ 1 \le k \le r\}) < \text{cost}(F_{i^*})$ với $i = 1, \ldots, p$ và qui ước $j_0 \notin F_{i_k}$.

Ta thấy rằng nếu tìm được một dãy như trên thì có thể giảm $\text{cost}(F_i)$ xuống mà không làm tăng giá trị L hiện có bằng cách thay các tập F_{i_k} bằng các tập mới $F_{i_k} \cup \{j_{k-1}\} \setminus \{j_k\}$ cùng với các điều kiện trên. Như vậy mỗi lần được một i như thế thì sẽ giảm mất một F_i có $\text{cost}(F_i)$ đạt max, quá trình này cứ tiếp tục thì sẽ có xu hướng làm giảm giá trị của L. Trong trường hợp không tìm được một giá trị i^* thỏa tính chất đã nêu thì thuật toán kết thúc. Bởi vì mỗi F_i như thế chỉ được xét nhiều nhất một lần nên sau một số hữu hạn bước thuật toán sẽ dừng.

Chúng ta sẽ dùng thuật toán phân chia công việc **Dividing** được mô tả dưới đây để tạo nên một lịch truy vấn ban đầu. Thuật toán này chỉ bảo đảm về mặt cân bằng tải mà không bảo đảm chi phí truyền thông.

Phát biểu bài toán: Giả sử có p bộ xử lí, N công việc $x_1, \ldots, x_N$ có thời gian thực hiện lần lượt là $t_1, \ldots, t_N$. Mỗi công việc có thể thực hiện trên một bộ xử lí bất kỳ nhưng phải thực hiện trọn vẹn. Hãy tìm cách phân chia N công việc cho p bộ xử lí sao cho thời gian hoàn thành là nhanh nhất.

Thuật toán 3.1. *Dividing*

Input: N công việc $x_1, \ldots, x_N$ và thời gian thực hiện tương ứng $t_1, \ldots, t_N$, p là số bộ xử lí.

Output: Phân hoạch $F_1, \ldots, F_p$ sao cho các F_i có tải *gần* bằng nhau.

Method:

1. $(F_1, \ldots, F_p) := \emptyset$
2. JOBS := $\{x_1, \ldots, x_N\}$
   repeat
3. Chọn F_i thỏa $\text{cost}(F_i) = \min_{1\le k\le p} \text{cost}(F_k)$;
4. Chọn x_j thỏa $t_j = \max_{x_k \in JOBS} t_k$;

5. $F_i := F_i \cup \{x_j\}$;
6. JOBS:=JOBS$\setminus\{x_j\}$;
until (JOBS $= \emptyset$);
7. *return* $(F_1, \ldots, F_p)$;
end.

Thuật toán tìm đường tăng luồng được mô tả chi tiết như sau.

Procesure *Flow_Distribution*$(F_1, \ldots, F_p)$
Begin
1. Gọi hàm Dividing $\rightarrow F_1, \ldots, F_p$ /*xác định lịch ban đầu*/
2. *While* true
Begin
3. Find_Increasing_Flow /*hàm tìm cách tăng luông*/
4. If (Find_Increasing_Flow$(F_1, \ldots, F_2) =$ null) *then*
return $(F_1, \ldots, F_p)$ /*nếu không tìm thấy đường tăng luồng thì kết thúc*/
5. *Else*
6. *For* $i := 1$ *to* p *do*
7. $F_i = F_i \cup \{j_{k-1} \mid i_k = i,\ 2 \le k \le r+1\} \setminus \{j_k \mid i_k = i,\ 1 \le k \le r\}$
End
End

Từ thuật toán ta thấy rằng yếu tố quyết định cho hiệu quả của thuật toán chính là hàm tìm đường tăng luồng $i_1, j_1, \ldots, i_r, j_r, i_{r+1}$ $(r > 0)$. Ta sẽ xây dựng hàm tăng luồng Find_Increasing_Flow có độ phức tạp là đa thức. Trước tiên ta dựa vào thuật toán tìm đường đi không có chu trình để xây dựng dãy tăng luồng như sau:

Function *Find_Increasing_Flow*$(F_1, \ldots, F_p)$.

Procedure *FindPath*(i)
Begin /*Trở về chương trình con gọi nó khi có tín hiệu dừng*/
1. *If* STOP *then* return:
2. $i_{r+1} = 1$;
/*Nếu dãy hiện thời thỏa mãn thì bật tín hiệu dừng và trở về chương trình con gọi nó*/
3. *If* cost$(F_i \cup \{j_{k-1} \mid i_k = i,\ 2 \le i \le r+1\} \setminus \{j_k \mid i_k = i,\ 1 \le i \le r\}) < L$ *then*
Begin
4. STOP=true;
5. *Return*;
End
6. *Else* /*Ngược lại nối thêm nếu có thể*/
7. *Begin* /*tăng độ dài của dãy lên*/
8. $r = r + 1$;
/*chọn một phần tử của tập F_i chuyển cho tập khác để F_i sau khi điều chỉnh luồng có cost$(F_i) < L$*/
9. *For* each (op $\in F_i$) *and* (op $\neq j_r$)
10. If cost$(F_i \cup \{j_{k-1} \mid i_k = i,\ 2 \le k \le r+1\} \setminus \{j_k \mid i_k = i, 1 \le k \le r\} \setminus \{\text{op}\}) < L$ *then*
11. *Begin*
12. $j_r =$ op;
/*Chọn một tập chưa được chuyển phần tử để chuyển từ F_i sang tập đó*/
13. *For* each $q \in \{1, \ldots, p\} \setminus \{i_1, \ldots, i_r\}$ *do*
Begin
14. FindPath(q);
15. *If* (STOP) *then* return;
End

```
            End
16.     r = r − 1;
      End
  End procedure

  Begin
1.    STOP = false
2.    L = max_i(cost(F_i)), i = 1, ..., p
3.    For i* = 1 to p do
4.      If cost(F_i*) = L then
5.        Begin
6.          r = 0;
7.          Find_Path (i*)
8.          If STOP then return (i_1, j_1, ..., i_r, j_r, i_{r+1})
          End
9.      Return null;
  End function
```

Nhận xét: Trong trường hợp không tìm được đường tăng luồng thì kết quả là phân hoạch của thuật toán Dividing. Thuật toán này có độ phức tạp là $O(n\log_2 n)$.

3.2. Thuật toán qui hoạch động cho bài toán lập lịch tối ưu

Trong thuật toán này chúng ta sẽ phân phối từng toán tử cho mỗi bộ xử lí nhưng phải theo thứ tự liên thông của cây. Đầu tiên cho các tập $F_1, \ldots, F_p$ được gán bằng rỗng. Giả sử tại mỗi thời điểm phân phối nút m, $F_1, \ldots, F_p$ là các tập chứa các toán tử đã được phân phối tương ứng cho các bộ xử lí $1, \ldots, p$. Khi đó sẽ có p sự lựa chọn phân phối toán tử m cho p bộ xử lí, trong p sự lựa chọn đó ta sẽ chọn cách nào làm cho $L = \max_{1\le i\le p} \text{cost}(F_i)$ đạt giá trị nhỏ nhất. Thuật toán sẽ dừng khi không còn nút nào để phân phối nữa.

```
/* Thuật toán sẽ được khởi tạo với F_i = φ, i = 1, ..., p.
   Gọi thủ tục với m = 1, với qui ước 1 là đỉnh gốc
   Dynamic_Distribution(p, 1)
   Xem cây truy vấn là biến toàn cục */
Proceddure Dynamic_Distribution(p, m)
  begin
1.    min = cost(F_1 ∪ {m})
2.    cho n = 1
3.    for i = 2 to p do
4.      if cost(F_i ∪ {m}) < min then
            begin
5.            min = cost(F_i ∪ {m})
6.            chon= i
            end
7.    F_chon = F_chon ∪ {m}
8.    for each i thuộc tập các nút con của m
9.    Dynamic_Distribution(p, i)
  end procedure
```

3.3. Thuật toán tổng hợp

Trong thuật toán Dynamic_Distrution ta thấy rằng sau khi phân phối xong một nút tại một phân phối nào đó ta có một lịch truy vấn cho cây truy vấn là cây con của cây truy vấn đang tìm.

Từ nhận xét này ta có thể lồng ghép thuật toán phân phối luồng ở phần trên vào thuật toán, cụ thể là ta gọi hàm Flow_Distrubution cho các tập $F_1, \ldots, F_p$. Việc phân phối này sẽ làm cải thiện giá trị $\max_i(\text{cost}(F_i))$ rất nhiều và do đó nó sẽ cho kết quả tốt hơn.

Sau đây là đoạn chương trình mô tả thuật toán:

/* Thuật toán sẽ được khởi tạo với $F_i = \phi$, $i = 1, \ldots p$
Gọi thủ tục với $m = 1$, với qui ước 1 là đỉnh gốc
Dynamic_Distrubution$(p, 1)$ */

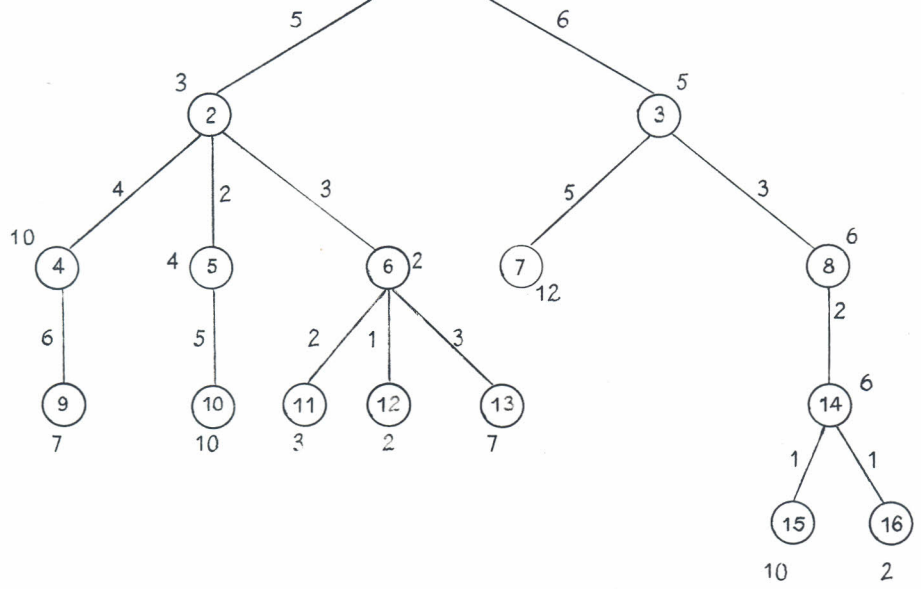
Proceddure Dynamic_Distribution(p, m)

begin

1. min = cost$(F_1 \cup \{m\})$
2. cho $n = 1$
3. *for* $i = 2$ to p *do*
4. *if* cost$(F_i \cup \{i\}) <$ min *then*
   begin
5. min = cost$(F_i \cup \{i\})$
6. chon$= i$
   end
7. $F_{\text{chon}} = F_{\text{chon}} \cup \{m\}$
8. Flow_Distribution$F_1, \ldots, F_p)$
9. *for* each i thuộc tập các nút con của m *do*
10. Dynamic_Distribution(p, i)

end procedure

Ví dụ. Xét cây toán tử 16 đỉnh dưới đây, với $p = 4$



Áp dụng thuật toán tổng hợp, kết quả các bước thực hiện được mô tả chi tiết như sau:

Kết quả phân phối động $m = 1$ $F(1) = \{1\}$; $F(2) = \{\}$; $F(3) = \{\}$; $F(4) = \{\}$;

Kết quả sau khi điều chỉnh bằng thuật toán luồng:

$F(1) = \{1\}$; $F(2) = \{\}$; $F(3) = \{\}$; $F(4) = \{\}$;
Kết quả phân phối động $m = 2$ $F(1) = \{1\}$; $F(2) = \{2\}$; $F(3) = \{\}$; $F(4) = \{\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{1\}$; $F(2) = \{2\}$; $F(3) = \{\}$; $F(4) = \{\}$;
Kết quả phân phối động $m = 4$ $F(1) = \{1\}$; $F(2) = \{2\}$; $F(3) = \{4\}$; $F(4) = \{\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{1\}$; $F(2) = \{2\}$; $F(3) = \{4\}$; $F(4) = \{\}$;
Kết quả phân phối động $m = 9$ $F(1) = \{1\}$; $F(2) = \{2\}$; $F(3) = \{4\}$; $F(4) = \{9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{1\}$; $F(2) = \{2\}$; $F(3) = \{4\}$; $F(4) = \{9\}$;
Kết quả phân phối động $m = 5$ $F(1) = \{1\}$; $F(2) = \{2,5\}$; $F(3) = \{4\}$; $F(4) = \{9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{5\}$; $F(2) = \{1\}$; $F(3) = \{2\}$; $F(4) = \{4,9\}$;
Kết quả phân phối động $m = 10$ $F(1) = \{5,10\}$; $F(2) = \{1\}$; $F(3) = \{2\}$; $F(4) = \{4,9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{5,10\}$; $F(2) = \{1\}$; $F(3) = \{2\}$; $F(4) = \{4,9\}$;
Kết quả phân phối động $m = 6$ $F(1) = \{5,10\}$; $F(2) = \{1\}$; $F(3) = \{2,6\}$; $F(4) = \{4,9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{5,10\}$; $F(2) = \{1\}$; $F(3) = \{2,6\}$; $F(4) = \{4,9\}$;
Kết quả phân phối động $m = 11$ $F(1) = \{5,10,11\}$; $F(2) = \{1\}$; $F(3) = \{2,6\}$; $F(4) = \{4,9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{5,10,11\}$; $F(2) = \{1\}$; $F(3) = \{2,6\}$; $F(4) = \{4,9\}$;
Kết quả phân phối động $m = 12$ $F(1) = \{5,10,11\}$; $F(2) = \{1,12\}$; $F(3) = \{2,6\}$; $F(4) = \{4,9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{5,10,11\}$; $F(2) = \{1,12\}$; $F(3) = \{2,6\}$; $F(4) = \{4,9\}$;
Kết quả phân phối động $m = 13$
$F(1) = \{5,10,11\}$; $F(2) = \{1\}$; $F(3) = \{2,6,13\}$; $F(4) = \{4,9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{6,11,12,13\}$; $F(2) = \{1,2\}$; $F(3) = \{5,10\}$; $F(4) = \{4,9\}$;
Kết quả phân phối động $m = 3$
$F(1) = \{6,11,12,13\}$; $F(2) = \{1,2,3\}$; $F(3) = \{5,10\}$; $F(4) = \{4,9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{2,6,13\}$; $F(2) = \{5,10,11,12\}$; $F(3) = \{1,3\}$; $F(4) = \{4,9\}$;
Kết quả phân phối động $m = 7$
$F(1) = \{2,6,13\}$; $F(2) = \{5,10,11\}$; $F(3) = \{1,3,7\}$; $F(4) = \{4,9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{1,2,6\}$; $F(2) = \{5,10,12,13\}$; $F(3) = \{3,7\}$; $F(4) = \{4,9,11\}$;
Kết quả phân phối động $m = 8$
$F(1) = \{1,2,6\}$; $F(2) = \{5,10,12,13\}$; $F(3) = \{3,7,8\}$; $F(4) = \{4,9,11\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{1,3,7\}$; $F(2) = \{2,6,12,13\}$; $F(3) = \{5,8,10\}$; $F(4) = \{4,9,11\}$;
Kết quả phân phối động $m = 14$
$F(1) = \{1,3,7\}$; $F(2) = \{2,6,12,13\}$; $F(3) = \{5,8,10,14\}$; $F(4) = \{4,9,11\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{1,3,7\}$; $F(2) = \{6,8,11,12,13\}$; $F(3) = \{5,10,14\}$; $F(4) = \{2,4,9\}$;
Kết quả phân phối động $m = 15$
$F(1) = \{1,3,7\}$; $F(2) = \{6,8,11,12,13\}$; $F(3) = \{5,10,14,15\}$; $F(4) = \{2,4,9\}$;
Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{5,6,10,11,13\}$; $F(2) = \{8,12,14,15\}$; $F(3) = \{1,3,7\}$; $F(4) = \{2,4,9\}$;
Kết quả phân phối động $m = 16$
$F(1) = \{5,6,10,11,13\}$; $F(2) = \{8,12,14,15,16\}$; $F(3) = \{1,3,7\}$; $F(4) = \{2,4,9\}$;

Kết quả sau khi điều chỉnh bằng thuật toán luồng:
$F(1) = \{5, 6, 10, 11, 13\}$; $F(2) = \{8, 12, 14, 15, 16\}$; $F(3) = \{1, 3, 7\}$; $F(4) = \{2, 4, 9\}$;
Lịch truy vấn kết quả tìm được:
$F(1) = \{5, 6, 10, 11, 13\}$; $\text{cost}(F_1) = 32$; $F(2) = \{8, 12, 14, 15, 16\}$; $\text{cost}F(2) = 30$; $F(3) = \{1, 3, 7\}$; $\text{cost}F(3) = 30$; $F(4) = \{2, 4, 9$; $\text{cost}F(4) = 30\}$.

Vậy chi phí thực hiện cây toán tử ở trên là $L = \max_{1 \le i \le 4} \text{cost}(F_i) = 32$.

Với cây toán tử này, chúng tôi đã thử nghiệm [7] khi dùng thuật toán tối ưu của Hasan là 40.

4. KẾT LUẬN

Bài báo đã tiếp cận bài toán tìm lịch truy vấn tối ưu cho cây toán tử dạng ống theo hướng sử dụng phương pháp qui hoạch động và tư tưởng của thuật toán tìm đường tăng luồng trong lí thuyết đồ thị hữu hạn. Mặc dù kết quả này đã được công bố bởi Hasan [8] năm 1997 nhưng phương pháp này có thể xây dựng những heuristic nhằm tìm kiếm lời giải tối ưu cho các cây toán tử phức tạp và lớp các cây toán tử hình sao.

TÀI LIỆU THAM KHẢO

[1] Bhaskar, Himatsingkar, Jaideep, Srivastara, *Tradeoffs in Parallel Processing and its Implication for Query Optimization*, Dept. of Computer Science University Minnesota Minneapolis MN 55455, 1997.

[2] Đỗ Xuân Lôi, *Cấu trúc dữ liệu và giải thuật*, NXB Giáo dục, Hà Nội, 1996.

[3] Hong, *Parallel Query Processing Using Shared Mamory Multiprocessors and Disk Array*, Univestity of California, Berkeley, August 1992.

[4] Kien A. Hua, *Parallel Database Technology*, University of Central Florida Orlande FL 32846-2362, 1997.

[5] Nguyễn Đức Nghĩa, Nguyễn Tô Thành, *Toán rời rạc*, NXB Giáo dục, Hà Nội, 1997.

[6] Nguyễn Xuân Huy, Nguyễn Mậu Hân, Lập lịch tối ưu trong cơ sở dữ liệu song song, *Tạp chí Tin học và Điều khiển học* **17** (3) (2001) 87–96.

[7] Nguyễn Xuân Huy, Nguyễn Mậu Hân, "Thuật toán tìm đường tăng luồng cho bài toán lập lịch tối ưu", Báo cáo toàn văn của Hội nghị kỷ niệm 25 năm thành lập Viện Công nghệ thông tin 12/2001.

[8] Waqar Hasan, *Optimization of SQL for Parallel Machines*, Springer, 1995.

[9] Weiyi Meng and Clement T. Yu, *Principles of Database Query Processing for Advanced Applications*, Morgan Kaufman Inc., 1998.

Nhận bài ngày 4 - 12 - 2001
Nhận lại sau khi sửa ngày 28 - 2 - 2002

Nguyễn Xuân Huy - Viện Công nghệ thông tin.
Nguyễn Mậu Hân - Trường Đại học Khoa học Huế.