Tạp chí Tin học và Điều khiển học, T. 17, S. 3 (2001), 87–96

LẬP LỊCH TỐI ƯU TRONG CƠ SỞ DỮ LIỆU SONG SONG

NGUYỄN XUÂN HUY, NGUYỄN MẬU HÂN

Abstract. Parallel execution offers a solution to the problem of reducing the response time queries against large database. As receiving a SQL query, the parallel DBMS first find a procedural plan to execute the query that delivers the query result in minimal time. There is two phase to execute the plan: the first phase applies the tactic of minimizing total work while the second applies the tactic of partitioning work among processors. More specialists in computer science are now researching the otimization schedule on parallel database problems to divide effectively the work into the processors. The paper suggests an optimization pipelined parallelism schedule algorithm in which communication cost among nodes of operator tree will be take care. Waqar Hasan has solved the same problem in non-communication cost case in 1995.

Tóm tắt. Xử lý song song cho phép giảm thiểu thời gian hỏi đáp của câu truy vấn trên các cơ sở dữ liệu (CSDL) lớn. Khi nhận một câu truy vấn SQL gửi đến, hệ quản trị CSDL trước tiên sẽ tìm phương án thi hành tối ưu để thời gian trả lời truy vấn là nhỏ nhất. Phương án này phải trải qua hai giai đoạn chính: giai đoạn làm cực tiểu khối lượng công việc và giai đoạn phân bố công việc cho các bộ xử lý. Bài toán lập lịch tối ưu để phân chia công việc một cách hợp lý cho các bộ xử lý là một bài toán được nhiều nhà tin học quan tâm. Bài báo này đề xuất một thuật toán lập lịch song song dạng ống (pipelined parallelism schedule) có tính đến chi phí truyền thông giữa các trạm. Trường hợp không tính đến chi phí truyền thông đã được Hasan giải quyết năm 1995 [5].

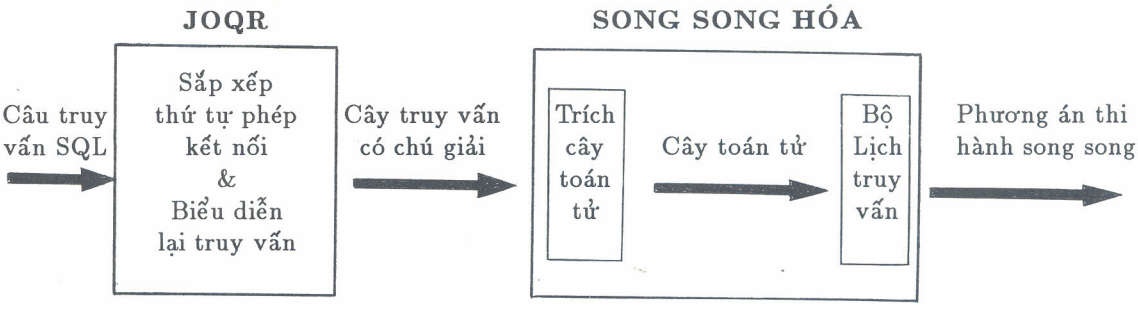
1. GIỚI THIỆU

Tối ưu hóa truy vấn là một đề tài được nhiều người quan tâm khi bắt đầu phát triển các hệ quản trị CSDL. Hiệu quả và tính khả thi của việc tổ chức khai thác CSDL trên môi trường đa xử lý đã thu hút sự quan tâm nghiên cứu của nhiều nhà tin học. Một yếu tố dẫn đến sự thành công của các hệ quản trị CSDL đa xử lý này là tính hiệu quả của bộ tối ưu hóa. Trước khi trả lời một câu truy vấn, bộ tối ưu hóa tiến hành hai giai đoạn [2]:

- *Giai đoạn JOQR* (Join Ordering and Query Rewriting). Mục đích chính của giai đoạn này là xây dựng các chiến lược để thực hiện các phép nối có hiệu quả nhằm giảm thiểu khối lượng công việc. Các chiến lược tối ưu đã lựa chọn được thể hiện qua cây truy vấn có chú giải (annotated query tree).

- *Giai đoạn song song hóa* (Parallelization). Cây truy vấn có chú giải được biến đổi để đưa ra một phương án thi hành song song. Mục đích chính của giai đoạn này là hình thành lịch truy vấn tối ưu để phân chia công việc một cách hợp lý cho các bộ vi xử lý.

Có thể mô tả quá trình tối ưu hóa câu hỏi trong CSDL song song như sau:



Bài báo này, tập trung vào bài toán lập lịch cho cây toán tử dạng ống (pipelined operator tree), nghĩa là một số toán tử của cây có thể thực hiện đồng thời, dữ liệu sản xuất ra của toán tử này có thể là dữ liệu tiêu thụ của toán tử kia. Lập lịch cho cây toán tử như thế là một bài toán phức tạp. Chúng ta sẽ đưa bài toán về dạng đơn giản hơn, có độ phức tạp đa thức, bằng các phép xóa cạnh và gộp các nút để chuyển cây toán tử phức tạp thành cây toán tử đơn điệu. Cuối cùng sẽ tìm một phân hoạch liên thông tối ưu cho các nút của cây toán tử để chuyển các cây con vào các bộ xử lý tương ứng. Trước tiên, chúng ta sẽ xây dựng thuật toán trong trường hợp các cạnh của cây có trọng số bằng 0, nghĩa là chi phí truyền thông giữa các nút không tính đến [5]. Sau đó chúng ta sẽ xét bài toán trong trường hợp có tính đến chi phí truyền thông để tìm kiếm lời giải tối ưu cho bài toán lập lịch.

2. MỘT SỐ ĐỊNH NGHĨA VÀ KHÁI NIỆM LIÊN QUAN

Định nghĩa 2.1.

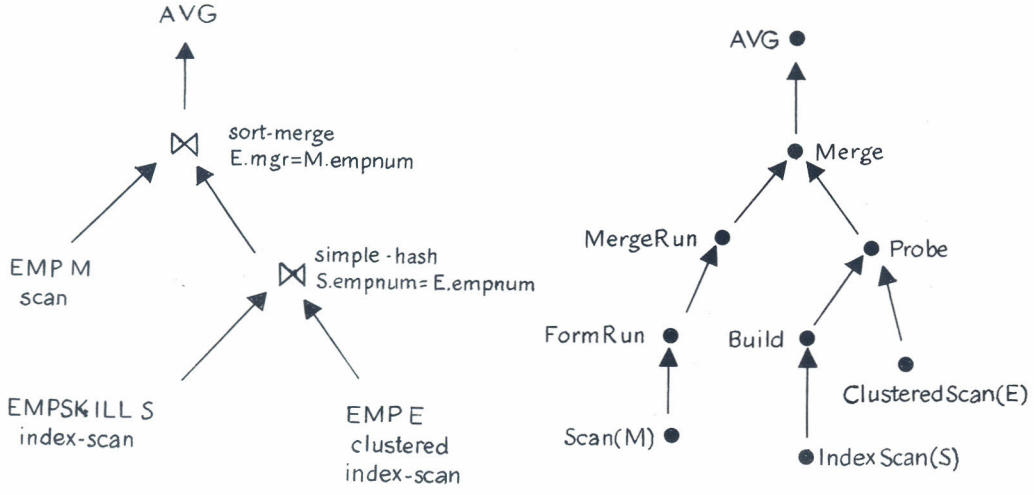
• *Cây truy vấn chú giải* (annotated query tree) là cây truy vấn cho biết thứ tự thực hiện mỗi phép toán và phương pháp tính toán mỗi toán tử. Mỗi nút trên cây đại diện cho một (hay nhiều) phép toán quan hệ. Những ghi chú trên mỗi nút mô tả cách nó được thực hiện chi tiết như thế nào (hình 1).

• *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 các toán tử dạng ống (pipelined operator) thì gọi là cây toán tử dạng ống (pipelined operator tree).

Ví dụ. Truy vấn sau đây để tìm lương trung bình của các nhân viên có kỹ năng "lập trình" và có tiền lương lớn hơn thủ trưởng của họ:

SECLECT	**avg** (E.salary)
FROM	Emp E, Emp M, EmpSkill S
WHERE	E.EmpNum=S.EmpNum **and** E.Mgr=M.EmpNum
	and E.Salary>M.Salary **and** S.Skill="Lập trình"

Ta có cây truy vấn chú giải và cây toán tử tương ứng là:



Cây truy vấn chú giải Cây toán tử tương ứng

Hình 1

Định nghĩa 2.2. Cho p bộ xử lý và cây toán tử $T = (V, E)$, trong đó V là tập các nút, E là tập các cạnh của cây. 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 công việc được phân cho bộ xử lý thứ k.

Định nghĩa 2.3. Bài toán lập lịch cây toán tử dạng ống được phát biểu như sau:
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 lý.
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 $\max_{1 \le i \le p} \left[\sum_{i \in F_i} t_i + \sum_{j \notin F_i} c_{ij} \right]$ là cực tiểu.

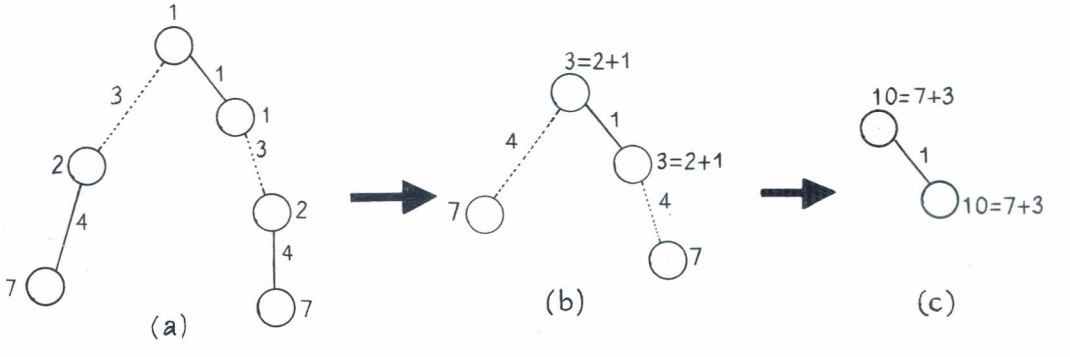
Định nghĩa 2.4. Tải (load) L_k lên bộ xử lý k là $\sum_{i \in F_k} \left[t_i + \sum_{j \notin F_k} c_{ij} \right]$.

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. Giả sử toán tử i được định vị đến bộ xử lý k thì tỉ lệ sử dụng của bộ xử lý này là $f_i = (1/L) \sum_{j \notin F_k} c_{ij}$. Như vậy, tỉ lệ sử dụng của một bộ xử lý là tổng tỉ lệ sử dụng các toán tử được thực hiện trên nó. Vì vậy, một lịch tối ưu sẽ tồn tại ít nhất một bộ xử lý ở tình trạng bão hòa theo nghĩa nó sẽ được tận dụng tối đa hay tỉ lệ tận dụng bằng 1. Ta có:

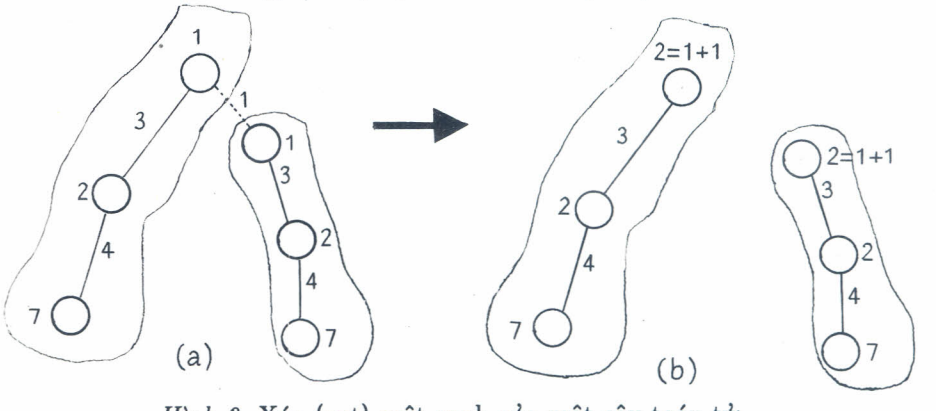
$$\max_{1 \le k \le p} \sum_{i \in F_k} f_i = 1 \Rightarrow L = \max_{1 \le k \le p} \left[\sum_{i \in F_k} t_i + \sum_{j \notin F_k} c_{ij} \right] = \max_{1 \le k \le p} L_k .$$

Định nghĩa 2.5. Nếu F là một tập các toán tử thì chi phí tải một bộ xử lý để thực hiện F được xác định bởi $\text{cost}(F) = \sum_{i \in F} \left[t_i + \sum_{j \notin F} c_{ij} \right]$.

Chúng ta sẽ sử dụng các thao tác *gộp* các nút và *xóa* cạnh của một toán tử để quyết định vị trí các nút kề nhau nên đặt trên cùng một bộ xử lý hay khác bộ xử lý.



Hình 2. Gộp (collapse) các nút của một cây toán tử



Hình 3. Xóa (cut) một cạnh của một cây toán tử

Định nghĩa 2.6. Collapse(i, j) là gộp hai nút i và j của một cây để có một nút mới i' có trọng số $t_{i'} = t_i + t_j$. Khi đó các cạnh nối với i và j được chuyển thành nối với i'.

Định nghĩa 2.7. Cut(i,j) là xóa đi cạnh (i,j) và thêm trọng số của nó vào nút i và j, nghĩa là $t_i^{\text{new}} = t_i^{\text{old}} + c_{ij}$ và $t_j^{\text{new}} = t_j^{\text{old}} + c_{ij}$.

Nếu một lịch truy vấn đặt cả i lẫn j trên cùng bộ xử lý k thì việc tải dữ liệu trên các bộ xử lý là bất biến khi i và j bị gộp, và nút mới sẽ được định vị trên bộ xử lý k.

Nếu một lịch truy vấn đặt i và j trên các bộ xử lý riêng biệt, thì việc tải dữ liệu là bất biến khi cạnh (i,j) bị xóa.

Các hình 2 và 3 cho thấy việc gộp nút và xóa cạnh trên một cây toán tử.

3. CẠNH KHÔNG CHẤP NHẬN VÀ CÂY ĐƠN ĐIỆU

Phần này khảo sát sự liên hệ giữa chi phí thực hiện song song và chi phí truyền thông.

Định nghĩa 3.1. Một cạnh $(i,j) \in E$ được gọi là *không chấp nhận* nếu $c_{ij} \geq t_i + \sum_{k \neq j} c_{ik}$ hoặc $c_{ik} \geq t_j + \sum_{k \neq i} c_{ik}$.

Như vậy, một cạnh là không chấp nhận nếu chi phí truyền thông của nó khá lớn và vượt quá lợi ích của việc song song hóa. Bằng thuật toán tiền xử lý **Pre_Processing** chúng ta sẽ tìm và loại bỏ các cạnh không chấp nhận và xây dựng một lớp các cây đơn điệu mà trên đó không tồn tại các cạnh không chấp nhận. Chúng ta cũng chỉ ra rằng lặp lại nhiều lần việc xóa các cạnh không chấp nhận sẽ cho một cây đơn điệu.

Định lý 3.1. *Cho p bộ xử lý và cây toán tử $T = (V,E)$, cạnh $(i,j) \in E$. Khi đó sẽ tồn tại một lịch truy vấn tối ưu của T cho p bộ xử lý mà trong đó nút i và j được gộp lại trên cùng một bộ xử lý* [5].

Định nghĩa 3.2. Một cây toán tử T được gọi là đơn điệu (monotone) nếu với hai tập liên thông các nút bất kỳ X, Y trên T thì chi phí thực hiện X sẽ thấp hơn chi phí thực hiện của một tập liên thông Y chứa nó. Nghĩa là, nếu $X \subset Y$ thì $\text{cost}(X) < \text{cost}(Y)$.

Từ định nghĩa, dễ thấy rằng T là một cây đơn điệu nếu và chỉ nếu T không có các cạnh không chấp nhận. Điều quan trọng là việc xây dựng một lịch truy vấn cho một cây đơn điệu dễ hơn nhiều so với việc xây dựng lịch truy vấn cho cây ban đầu. Điều này chỉ chấp nhận được khi việc gộp các cạnh không chấp nhận mà không làm mất ý nghĩa của việc tối ưu hóa. Hơn thế nữa, lịch truy vấn cho cây ban đầu có thể tìm lại được từ lịch truy vấn của cây đã chuyển đổi.

Thuật toán 3.1. *Pre_Processing*

Input: Một cây toán tử.

Output: Một cây toán tử đơn điệu.

Method:

While tồn tại cạnh không chấp nhận (i,j)
  Collapse(i,j)
End while

End

Do vậy cây đang xét là hữu hạn và mỗi lần gộp sẽ làm giảm số các nút, nên thuật toán sẽ dừng. Việc kiểm tra sự tồn tại của cạnh không chấp nhận là yếu tố xác định chủ yếu của thời gian thực hiện thuật toán. Thuật toán thực hiện với độ phức tạp là $O(nd)$, với n là số các nút và d là giá trị lớn nhất của bậc các nút.

Bổ đề 3.1. *Cho $R_i = [t_i + \sum_{j \in V} c_{ij}]$ là trọng số của nút i. Thời gian trả lời của lịch truy vấn bất kỳ của một toán tử đơn điệu có một giới hạn thấp hơn $R = \max_{i \in V} R_i$.*

Bổ đề 3.2. *Thời gian trả lời của một lịch truy vấn với p bộ xử lý của một toán tử bất kỳ luôn luôn lớn hơn $\overline{W} = W/p$ với $W = \sum_{i \in V} t_i$ là tổng trọng số của các nút của cây.*

4. LỊCH TRUY VẤN LIÊN THÔNG

Một lịch truy vấn được gọi là liên thông nếu những nút được định vị trên một bộ xử lý nào đó phải là một tập liên thông. Ràng buộc này tương đương với việc chỉ xét các cây truy vấn chỉ chịu chi phí truyền thông trên $p-1$ cạnh khi sử dụng p bộ xử lý. Bởi vì bài toán xác định truy vấn tối ưu tổng quát là NP-khó nên chúng ta sẽ tìm lịch truy vấn liên thông tối ưu tương ứng có độ phức tạp đa thức.

4.1. Lịch truy vấn liên thông không có chi phí truyền thông

Chúng ta đưa ra một thuật toán để tìm kiếm lịch truy vấn liên thông tối ưu cho các cây có trọng số ở các cạnh đều bằng 0 ($c_{ij}=0$). Thuật toán được xây dựng hai bước:

Bước thứ nhất là với một cận B và số bộ xử lý là p, ta sẽ xây dựng một thuật toán hiệu quả để tìm một phân hoạch liên thông gồm p tập $F_1,\ldots,F_p$ với $\max_{1\le i\le p}\text{cost}(F_i)\le B$, nếu nó tồn tại.

Bước thứ hai sẽ xây dựng thuật toán tổng thể bằng cách lần lượt dùng thuật toán ở bước thứ nhất. Chúng ta bắt đầu từ giá trị B được đặt bằng cận dưới thời gian trả lời và tăng dần giá trị B cho đến khi thuật toán ở bước thứ nhất cho lời giải hợp lý và khi đó giá trị B là cũng là giá trị nhỏ nhất.

Định nghĩa 4.1.1. Một lịch truy vấn được gọi (B,p)-bị chặn nếu nó là một lịch truy vấn liên thông và sử dụng tối đa p bộ xử lý và có một thời gian trả lời tối đa là B.

Định nghĩa 4.1.2. Một nút được gọi là nút mẹ nếu các nút kề của nó là các nút lá với nhiều nhất là một ngoại lệ (nghĩa là có thể có một nút không là nút lá).

Bổ đề 4.1.1. *Giả sử m là một nút mẹ với các nút con $r_1,\ldots,r_d$ được sắp xếp không giảm theo trọng số, nghĩa là $t_{r_1}\le\ldots\le t_{r_d}$. Nếu một lịch truy vấn (B,p)-bị chặn S có các nút m và r_j đặt trong cùng một phân đoạn và các nút r_i được đặt trong một phân đoạn khác, với $i<j$ (nghĩa là $t_{r_i}\le t_{r_j}$) thì lịch truy vấn S' được tạo từ S bằng cách đổi chỗ r_j và r_i cũng là (B,p)-bị chặn.*

Sử dụng lặp lại bổ đề trên ta có kết quả sau:

Bổ đề 4.1.2. *Nếu tồn tại một lịch truy vấn (B,p)-bị chặn thì cũng tồn tại một lịch truy vấn (B,p)-bị chặn khác sao cho:*

(1) *Nếu hai nút m, r_j được gộp thì hai nút m,r_{j-1} cũng được gộp.*

(2) *Nếu cạnh (m,r_j) bị xóa thì cạnh (m,r_{j+1}) cũng bị xóa.*

Gọi s là số lớn nhất các nút con mà có thể được gộp với nút mẹ m mà không vượt quá cận B. Nghĩa là từ s ta có $t_m+\sum_{1\le i\le s}t_{r_i}\le B$.

Định lý 4.1.1. *Nếu tồn tại một lịch truy vấn (B,p)-bị chặn thì cũng tồn tại một lịch truy vấn (B,p)-bị chặn sao cho:*

(1) *Hai nút m,r_j được gộp với $1<j<s$.*

(2) *Cạnh (m,r_j) bị xóa với $s<j<d$.*

Định lý 4.1.1 cho phép tìm lịch (B,p)-*bị chặn* hoặc chỉ ra rằng không tồn tại lịch nào như vậy. Chọn một nút mẹ bất kỳ và xếp lại các nút con theo thứ tự không tăng của trọng số. Ta gộp các nút con thành nút mẹ sao cho trọng số của nút mẹ phải ở dưới cận B và tách phần còn lại. Lặp lại tiến trình trên cho đến khi nào không còn nút nào được gộp hay chúng ta đã thực hiện xóa $p-1$ cạnh. Nếu trọng số của phân đoạn cuối cùng không hơn B, thì chúng ta tìm ra một lịch (B,p)-*bị chặn*, ngược lại không có lịch truy vấn nào như thế tồn tại.

Thuật toán 4.1.1. *BpSchedule*

Input: cây toán tử T với các cạnh có trọng số 0, cận B.

Output: Phân hoạch T thành các phân đoạn $F_1, ..., F_p$ sao cho $\text{cost}(F_i) \leq B$ với $i = 1, ..., p-1$.

Method:

1. *While* tồn tại một nút mẹ m
2. Gọi $r_1, ..., r_d$ là d nút con của m sao cho: $t_{r_1} \leq ... \leq t_{r_d}$
3. Chọn $s \leq d$ sao cho s là giá trị lớn nhất thỏa mãn $t_m + \sum_{1 \leq i \leq s} t_{r_i} \leq B$
4. *For* $j = 1$ *to* s *do*
5. Collapse(m, r_j)
6. *For* $j = s+1$ *to* s *do*
7. Cut(m, r_j)
8. *If* tổng cộng số cut là $p-1$ *goto* 10
9. *End while*
10. *Return* (kết quả phân hoạch $F_1, ..., F_p$)

Ta sẽ tìm một lịch truy vấn liên thông tối ưu bằng cách gán B đến một cận dưới nào đó, sau đó tăng B một lượng lớn nhất có thể được và xét lại B nhiều lần nhưng phải bảo đảm rằng không vượt quá giá trị tối ưu cần tìm. Với mỗi giá trị B như thế, thực hiện Bpschedule và kiểm tra phân hoạch liên thông tìm được có thỏa mãn điều kiện $\max_{1 \leq i \leq p}(\text{cost}(F_i)) \leq B$ hay không. Dựa vào Bổ đề 4.1.1 và Bổ đề 4.1.2 ta chọn cận B ban đầu là $\max(\overline{W}, R_{\max})$, ở đây với giả thiết trọng số của các cạnh đều bằng 0 nên $R_{\max} = \max_{i \in V}(t_i + \sum_{j \in V} c_{ij}) = \max_{i \in V} t_i$. Trong trường hợp phân hoạch liên thông tìm được không thỏa mãn thì ta sẽ dựa vào phân hoạch đó để tìm cách tăng giá trị B.

Ta gọi nút *liền kề* của một tập F_i là một nút có trọng số nhỏ nhất trong các nút không phụ thuộc F_i nhưng lại nối tới một đỉnh trong F_i, gọi $B_i = \text{cost}(F_i)$ + trọng số của nút liền kề. Mỗi lần thực hiện lại thành công thuật toán thì phân hoạch chứa các nút gộp sẽ lớn hơn và khi đó giá trị mà B sẽ tăng lên để không vượt quá giá trị nhỏ nhất cần tìm là $B^* = \min_{j \in V} B_i$. Từ đây ta có thuật toán để tìm phân hoạch liên thông có $\max_{1 \leq i \leq p} \text{cost}(F_i)$ là nhỏ nhất.

Thuật toán 4.1.2. *BalancedCuts*

Input: Cây toán tử có trọng số của các cạnh bằng 0, số bộ xử lý là p.

Output: Phân hoạch liên thông $F_1, ..., F_p$ sao cho $\max_{1 \leq i \leq p} \text{cost}(F_i)$ nhỏ nhất.

Method:

1. $B = \max\left((1/p)\sum_{i \in V} t_i, \max_{i \in V} t_i\right)$
2. *While* true
3. $F_1, ..., F_p = \text{BpSchedule}(T, B)$
4. *If* $\text{cost}(F_p) \leq B$ *then return* $F_1, ..., F_p$
5. $B_i = \text{cost}(F_i)$ + trọng số của nút liền kề của F_i
6. $B = \min_i B_i$

End while

End

4.2. Lịch truy vấn liên thông có tính đến chi phí truyền thông

Đến đây ta đã giải quyết được tìm lịch tối ưu trong trường hợp trọng số các cạnh của cây toán tử bằng 0. Tuy nhiên, trong thực tế thì chi phí truyền thông giữa các nút không thể bỏ qua được. Tức là trọng số của các cạnh khác 0 ($c_{ij} \neq 0$). Trong thuật toán BpSchedule ta thấy rằng việc thêm một nút đến một phân đoạn mà vẫn bảo đảm tính liên thông sẽ làm tăng chi phí của phân đoạn đó. BpSchedule làm cho các phân đoạn lớn lên do việc gộp các nút con với nút mẹ miễn sao chi phí trên phân đoạn đó vẫn bị chặn. Các nút con vẫn được sắp xếp theo thứ tự không giảm của trọng số. Với các cạnh có trọng số khác 0, thì nút mẹ phải chịu chi phí truyền thông cho nút con khi nó ở một phân đoạn khác. Vì vậy việc gộp nút con i với nút mẹ m sẽ làm tăng thêm chi phí của phân

đoạn là $t_i - c_{im}$. Việc sắp xếp các nút con của nút mẹ theo thứ tự không giảm của $t_i - c_{im}$ là cần thiết để áp dụng được các Bổ đề 4.1.1, 4.1.2 và Định lý 4.1.1.

Thuật toán BpSchedule trong trường hợp có chi phí truyền thông được viết lại như sau:

Thuật toán 4.2.1 *BpSchedile_Cost*

Input: Cây toán tử T; cận B.

Output: Phân hoạch T thành các $F_1, \ldots, F_p$ sao cho $\text{cost}(F_i) \leq B$ với $i = 1, \ldots, p-1$.

Method:

1. số_nhát_cắt = 0
2. *While* (tồn tại một nút mẹ m) và (số_nhát_cắt $< p - 1$)
3. Giả sử m có d nút con $r_1, \ldots, r_d$ với $t_{r_1} - c_{r_1 m} \leq \ldots \leq t_{r_d} - c_{r_d m}$
   Gọi n nút là nút mẹ của m
4. Chọn $s \leq d, s$ là giá trị lớn nhất thỏa: $t_m + c_{mn} + \sum_{1 \leq i \leq s} t_{r_i} + \sum_{s+1 \leq i \leq d} c_{r_i m} \leq B$
5. *For* $j = 1$ *to* s *do*
6. collapse$(m; r_j)$
7. *For* $j = s + 1$ *to* d *do*
   Begin
8. Cut$(m; r)$
9. số_nhát_cắt = số_nhát_cắt + 1
   End
   End while
10. *Return* (phân hoạch liên thông $F_1, \ldots, F_p$)

Từ đây ta có thuật toán để tìm phân hoạch liên thông có $\max_{1 \leq i \leq p} \text{cost}(F_i)$ là nhỏ nhất trong trường hợp có chi phí truyền thông như sau:

Thuật toán 4.2.2. *BalancedCuts_Cost*

Input: . Cây toán tử có trọng số của các cạnh khác 0
. Số bộ xử lý là p

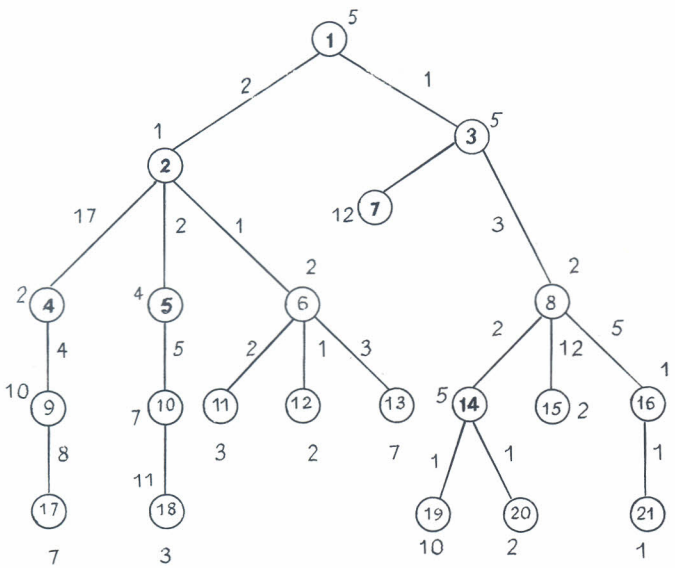
Output: Phân hoạch liên thông $F_1, \ldots, F_p$ sao cho $\max_{1 \leq i \leq p} \text{cost}(F_i)$ nhỏ nhất.

Method:

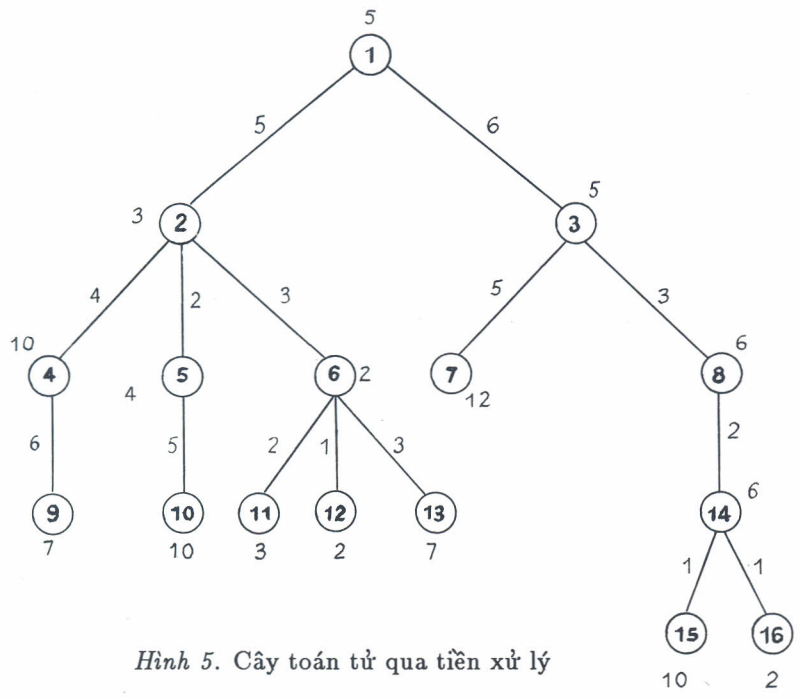
1. $B = \max(\frac{1}{p} \sum_{i \in V} t_i, \max_{i \in V} t_i)$
2. *While* true
3. $F_1, \ldots, F_p = \text{BpSchedule}(T, B)$
4. *If* $\text{cost}(F_p) \leq B$ *then return* $F_1, \ldots, F_p$
5. *For* $i = 1$ *to* p
   Begin
6. $k =$ nút liên kề của F_i
7. $cc =$ trọng số của cạnh nối F_i với k
8. $n =$ nút mẹ của k
9. $B_i = \text{cost}(F_i) + t_k - 2cc + c_{kn}$
   End
10. $B = \min_i B_i$
    End while

End

Ví dụ: Xét cây toán tử ban đầu có 21 đỉnh (Hình 4). Sau khi qua giai đoạn tiền xử lý, bằng cách gộp các nút tạo bởi các cạnh không chấp nhận được: (2,4), (10,18), (8,15), (8,16), (16,21) ta được cây đơn điệu (Hình 5).



Hình 4. Cây toán tử ban đầu



Hình 5. Cây toán tử qua tiền xử lý

Sau khi thực hiện quá trình để tìm phân hoạch liên thông bằng cách sử dụng các thuật toán đã nêu ở trên ta được các kết quả sau:

$p = 4$, $\sum_{i \in V} t_i = 94$.

- $B = 94/4 = 23{,}5$

$F_1 = \{6, 11, 12, 13\}$, $\text{cost}(F_1) = 17$, $B_1 = \text{cost}(F_1) + t_2 - 2c_{26} + c_{21} = 17 + 12 - 6 + 5 = 28$

$F_2 = \{5, 10\}$ $\text{cost}(F_2) = 16$, $B_2 = \text{cost}(F_2) + t_2 - 2c_{25} + c_{21} = 16 + 12 - 4 + 5 = 29$

$F_3 = \{4, 9\}$, $\text{cost}(F_3) = 21$, $B_3 = \text{cost}(F_3) + t_2 - 2c_{24} + c_{21} = 21 + 12 - 8 + 5 = 30$

$F_4 = \{1, 2, 3, 7, 8, 14, 15, 16\}$, $\text{cost}(F_4) = 58 > B_4 = \text{cost}(F_4) + t_6 - 2c_{26} = 58 + 14 - 6 = 66$

$\max_{1 \le i \le 4} \text{cost}(F_i) = 53 > B$, tiếp tục.

- $B = minB_i = 28$

  Lặp $B = 28$

$F_1 = \{5, 10\}$, $\text{cost}(F_1) = 16$, $B_1 = \text{cost}(F_1) + t_2 - 2c_{25} + c_{21} = 16 + 23 - 4 + 5 = 40$

$F_2 = \{5, 10\}$, $\text{cost}(F_2) = 16$, $B_2 = \text{cost}(F_2) + t_2 - 2c_{25} + c_{21} = 16 + 12 - 4 + 5 = 29$

$F_3 = \{8, 14, 15, 16\}$, $\text{cost}(F_3) = 27$, $B_3 = \text{cost}(F_3) + t_6 - 2c_{38} + c_{13} = 27 + 14 - 6 + 6 = 41$

$F_4 = \{1, 2, 3, 6, 7, 11, 12, 13\}$, $\text{cost}(F_4) = 39$, $B_4 = \text{cost}(F_4) + t_6 - 2c_{25} = 39 + 14 - 4 = 49$

$\max_{1 \le i \le 4} \text{cost}(F_i) = 39 > B$, tiếp tục.

- $B = \min B_i = 40$

$F_1 = \{4, 9\}$ $\text{cost}(F_1) = 21$

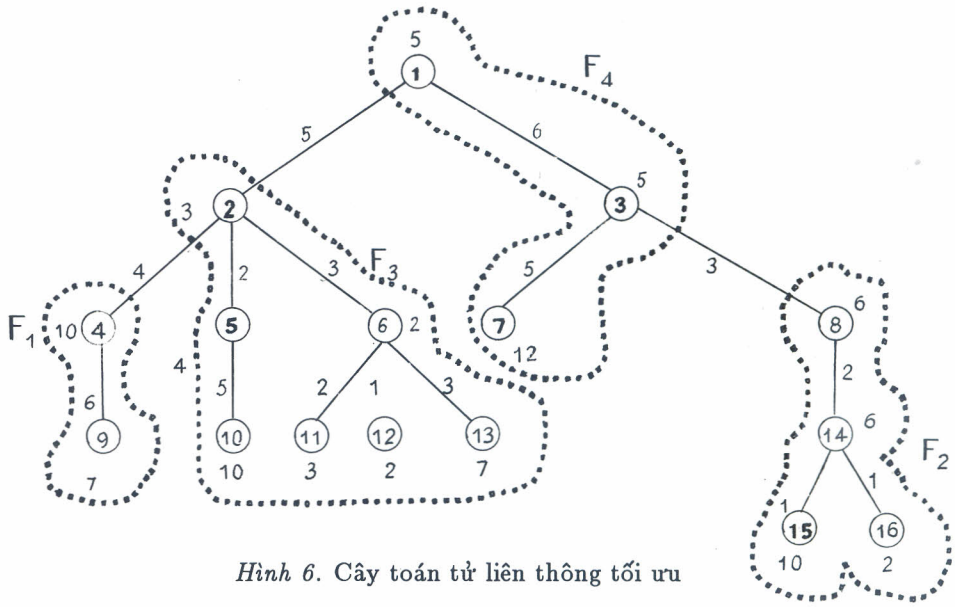
$F_2 = \{8, 14, 15, 16\}$ $\text{cost}(F_2) = 24$

$F_3 = \{2, 5, 6, 10, 11, 12, 13\}$ $\text{cost}(F_3) = 40$

$F_4 = \{1, 3, 7\}$ $\text{cost}(F_4) = 30$

$L = \max_{1 \le i \le 4} \text{cost}(F_i) = 40 = B$, dừng.

Các tập này ngăn cách nhau như ở hình dưới đây.



Hình 6. Cây toán tử liên thông tối ưu

Nhận xét. Thuật toán sẽ cho kết quả tốt nhất trong trường hợp cây toán tử *path* (path là cây toán tử chỉ có hai nút lá) và cho kết quả xấu nhất trong trường hợp cây toán tử là *star* (star là cây toán tử chỉ có một mút không phải là nút lá còn toàn bộ các nút khác là nút lá). Thông thường thì nếu cây toán tử mà bậc các đỉnh càng bé thì thuật toán càng hiệu quả.

6. KẾT LUẬN

Bài báo đã đề xuất thuật toán lập lịch truy vấn tối ưu cho cây toán tử dạng ống có tính đến chi phí truyền thông. Trường hợp không tính đến chi phí truyền thông đã được Hasan giải quyết năm 1995. Tuy nhiên, trong thực tế với những mạng máy tính nhỏ hoặc siêu máy tính ta có thể bỏ qua chi phí truyền thông, nhưng với những mạng máy tính lớn thì chi phí truyền thông ảnh hưởng khá lớn đến thời gian truy vấn thông tin. Qua một số thử nghiệm khai thác CSDL từ các trang WEB ta thấy khâu chậm nhất là khâu chuyển tải thông tin từ CSDL lên trang WEB và ngược lại. Có thể giải thích lý do này là do các chương trình trợ giúp việc chuyển tải dữ liệu, ví dụ các thủ tục truy nhập được viết bằng ngôn ngữ JAVA, thì do bản chất thông dịch của ngôn ngữ nên các thao tác truy nhập thông tin chậm một cách đáng kể so với ở CSDL tập trung.

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] Hong, *Parallel Query Processing Using Shared Memory Multiprocessors and Disk Arrays*, University of California, Berkeley, August 1992.

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

[4] M.R. Garey and D.S. Johnson, *Computer and Intractability*, W.H. Freeman and Company, 1989.

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

Nhận bài ngày 10 tháng 4 năm 2001
Nhận bài sau khi sửa ngày 8 tháng 7 năm 2001

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