Tạp chí Tin học và Điều khiển học, T.23, S.2 (2007), 184–200

PHÁT TRIỂN CÁC GIẢI THUẬT SONG SONG TRONG KHAI PHÁ LUẬT KẾT HỢP

NGUYỄN LONG GIANG

Viện Công nghệ thông tin, Viện Khoa học và Công nghệ Việt Nam

Abstract. In this paper we present two parallel algorithms for mining association rules that are well suited for distributed memory parallel computers. The algorithms are developed based on FP-growth method. The first algorithm is a task parallel formulation using a static load balancing technique. The second algorithm improves upon the first algorithm by dynamically balancing the load when the static task assignment leads to load imbalance. We use the count matrix technique to compute the weight of tasks and to distribute tasks to processors. This technique also helps to reduce the time needed to scan the trees and to reduce communication cost. We also use the hash tree technique to group similar prefix-paths extracted from the tree, and thus can greatly reduce the amount of information exchanged among processors. Our experiments show that the algorithms are capable of achieving very good speedups, and of substantially reducing the amount of time when finding frequent patterns in very large databases.

Tóm tắt. Bài báo này giới thiệu hai giải thuật song song khai thác luật kết hợp sử dụng trên các máy tính song song có bộ nhớ phân tán. Các giải thuật được phát triển trên phương pháp FP-growth. Giải thuật thứ nhất thực hiện tính toán song song sử dụng kỹ thuật cân bằng tải tĩnh. Giải thuật thứ hai được phát triển dựa trên giải thuật thứ nhất sử dụng kỹ thuật cân bằng tải động khi mất cân bằng tải xảy ra. Bài báo sử dụng kỹ thuật ma trận đếm để tính toán trọng số của các tác vụ và phân phối các tác vụ tới những bộ xử lý. Kỹ thuật này giảm thiểu thời gian duyệt cây và giảm bớt chi phí truyền thông giữa các bộ xử lý. Bài báo cũng sử dụng kỹ thuật cây băm để nhóm các tiền tố đường đi giống nhau trích lọc từ cây, và như vậy giảm thiểu lượng thông tin trao đổi giữa các bộ xử lý. Kết quả thử nghiệm cho thấy các giải thuật đạt được độ tăng tốc rất tốt, và giảm thiểu đáng kể thời gian tìm kiếm các mẫu phổ biến trong các CSDL lớn.

1. GIỚI THIỆU

Bài toán khai thác luật kết hợp (Association Rule Mining - ARM) được Agrawal, Imielinski và [1] giới thiệu lần đầu tiên. Bài toán này tập trung tìm kiếm những mối quan hệ có ích tiềm ẩn giữa các mẫu dữ liệu trong một CSDL. Các luật kết hợp đã và đang được sử dụng rất hiệu quả trong một loạt các ứng dụng từ hỗ trợ ra quyết định, dự báo, y tế, tiếp thị và quản trị kinh doanh...

Cho I là một tập các mục trong CSDL giao dịch. Một giao dịch $T \subseteq I$ được định nghĩa là một tập các mục được sinh ra đồng thời trong một giao dịch. Một luật kết hợp giữa tập mục $X \subseteq I$ và tập mục $Y \subseteq I$, biểu diễn $X \to Y$, chỉ ra rằng khả năng hiện diện của các phần tử X trong giao dịch cũng kéo theo khả năng hiện diện của phần tử Y. Độ đo được sử

dụng để đánh giá mức độ kết hợp của luật là độ hỗ trợ (*support*) và độ tin cậy (*confidence*). Độ hỗ trợ của luật $X \to Y$ là tỷ lệ giữa số giao dịch chứa cả X và Y với tổng số giao dịch trong cơ sở dữ liệu. Độ tin cậy của luật $X \to Y$ là tỷ lệ giữa số giao dịch chứa cả X và Y với số giao dịch chứa X. Một tập hợp gồm các mục (item) trong CSDL gọi là một *tập mục* (itemset). Một *tập mục* với k mục gọi là một *tập k mục* (k-itemset). Bài toán khai thác dữ liệu sử dụng luật kết hợp được chia thành hai bước. Bước thứ nhất tìm kiếm tất cả các tập mục phổ biến, là tập mục xuất hiện trong CSDL với độ hỗ trợ lớn hơn độ hỗ trợ tối thiểu. Bước thứ hai dựng các luật tiềm ẩn trong các tập mục phổ biến.

Đã có rất nhiều giải thuật khai thác luật kết hợp được đưa ra, tuy nhiên chúng đều gặp phải vấn đề hiệu năng tính toán khi xử lý những CSDL lớn, và việc tìm kiếm các giải pháp mới vẫn trở nên cấp thiết. Cách tiếp cận song song mang lại nhiều triển vọng cho việc giải quyết vấn đề này. Mục tiêu của bài báo là phát triển các giải thuật song song nhằm tăng hiệu năng thực hiện của các giải thuật và giảm thiểu thời gian tính toán.

Phần còn lại của bài báo có cấu trúc như sau. Phần 2 giới thiệu phương pháp FP-Growth. Phần 3 phát triển hai giải thuật song song khai thác các tập phổ biến từ CSDL giao dịch. Phần 4 trình bày nghiên cứu hiệu năng của giải thuật. Phần 5 trình bày kết luận và hướng nghiên cứu tiếp theo.

2. PHƯƠNG PHÁP FP-GROWTH

Phương pháp FP-Growth sử dụng cấu trúc dữ liệu FP-Tree (Frequent Pattern Tree - cây mẫu phổ biến)[5]. FP-Tree biểu diễn các thông tin về tần suất của các mục trong CSDL. Mỗi nhánh của FP-Tree biểu diễn một tập mục phổ biến, các nút dọc theo nhánh được lưu trữ theo thứ tự giảm dần của tần suất các mục, nút lá biểu diễn các mục có tần suất nhỏ nhất (mục phổ biến ít nhất). FP -Tree gắn với một bảng tiêu đề (header table). Các mục và số đếm của chúng được lưu trữ trong bảng tiêu đề theo thứ tự giảm dần của tần suất. Đầu vào của một mục chứa nút đầu danh sách liên kết với tất cả các nút tương ứng của FP-Tree. Phương pháp FP-Growth bao gồm hai nhiệm vụ chính: 1) xây dựng FP-Tree, và 2) khai thác các mẫu phổ biến sử dụng FP-Tree.

Xây dựng FP-Tree

Để xây dựng FP-Tree cần phải quét CSDL hai lần. Lần quét thứ nhất tìm tất cả *1-itemsets* phổ biến. Sau đó các mục này được chèn vào bảng tiêu đề, theo thứ tự giảm dần của tần suất. Lần quét thứ hai xây dựng FP-Tree. Tất cả các mục không phổ biến trong giao dịch được lược bỏ. Các mục còn lại sắp xếp theo thứ tự giảm dần của tần suất, sau đó được chèn vào FP-Tree thành một nhánh. Nếu một tập mục dùng chung tiền tố với một tập mục đã tồn tại trong cây, tập mục mới sẽ dùng chung tiền tố của nhánh cây biểu diễn tập mục đó. Hơn nữa, một bộ đếm được gắn với mỗi nút trên cây. Bộ đếm lưu trữ số giao dịch chứa tập mục biểu diễn bởi đường dẫn từ gốc tới nút. Bộ đếm này được cập nhật trong suốt lần quét thứ hai, khi một nhánh mới được chèn thêm.

Khai thác các tập phổ biến sử dụng FP-Tree

Cho mục i trong bảng tiêu đề của FP-Tree T_α (α là một tập mục, và T_i biểu thị FP-Tree được xây dựng từ CSDL ban đầu), theo danh sách kiên kết bắt đầu tại mục i trong bảng tiêu đề của T_α , tất cả các nhánh chứa mục i được thăm. Các nhánh đó tạo thành mẫu điều

kiện cơ sở của $\alpha \cup i$, vì vậy hàng ngang thu được tất cả các mục phổ biến trong mẫu điều kiện cơ sở này. Tiếp theo phương pháp FP-Growth xây dựng FP-Tree điều kiện $T_{\alpha \cup i}$, bằng việc khởi tạo lần đầu bảng tiêu đề của nó dựa vào các mục phổ biến được tìm thấy, sau đó tiến hành thăm các nhánh của T_α theo danh sách liên kết của i một hoặc nhiều lần và chèn các tập mục tương ứng vào $T_{\alpha \cup i}$. Chú ý rằng thứ tự của các mục có thể khác nhau trong T_α và $T_{\alpha \cup i}$. Thủ tục ở trên được sử dụng đệ quy, và dừng lại khi FP-Tree kết quả chứa một đường đi đơn duy nhất. Tập đầy đủ các tập mục phổ biến được sinh ra từ tất cả các đường đi đơn của FP-Tree.

3. CÁC GIẢI THUẬT SONG SONG, THIẾT KẾ VÀ THỰC THI

Phân chia CSDL ban đầu và tính đúng đắn của phương pháp

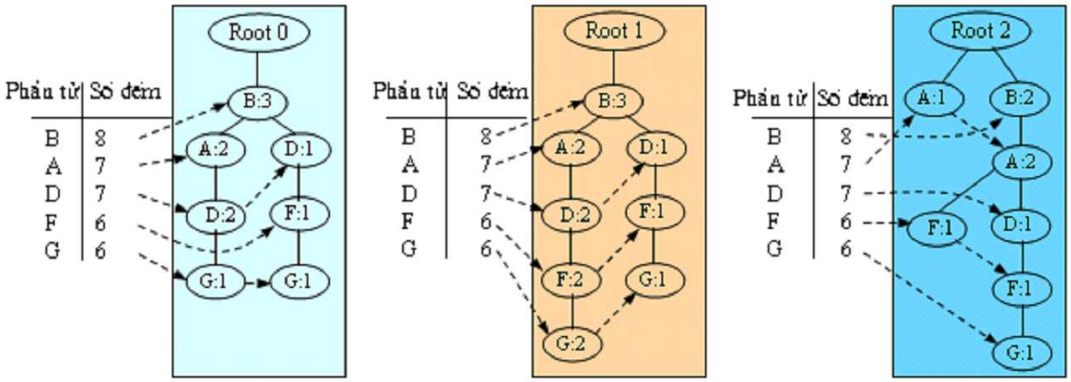
Nếu N là tổng các bộ xử lý, CSDL ban đầu được chia thành N phần bằng nhau, sau đó mỗi phần được gắn vào các bộ xử lý khác nhau, và được lưu trữ trong bộ nhớ cục bộ của nó. Mỗi bộ xử lý sẽ quét CSDL giao dịch cục bộ một lần và liệt kê những sự cố cục bộ. Sau đó tất cả các bộ xử lý sẽ thu được các mục phổ biến bằng việc trao đổi các số đếm cục bộ với tất cả các bộ xử lý khác sử dụng phép rút gọn tổng toàn cục (global sum-reduction). Các mục phổ biến này được sắp xếp theo thứ tự giảm dần theo độ hỗ trợ để xây dựng L, danh sách các mục phổ biến. Chú ý rằng L giống nhau với tất cả các bộ xử lý. Mỗi bộ xử lý quét dữ liệu cục bộ để xây dựng FP-Tree cục bộ, giống như giải thuật xây dựng FP-Tree. Để nắm rõ tiến trình này, xét ví dụ sau.

Ví dụ 1. Cho CSDL giao dịch, DB, là hai cột đầu tiên trong Bảng 1, số lượng bộ xử lý là 3 và ngưỡng hỗ trợ tối thiểu là 5. Ví dụ này minh họa quá trình xây dựng FP-Tree cục bộ.

Bảng 1. Ví dụ về việc phân chia CSDL giao dịch

ID	Các mục trong giao dịch	Các mục được phổ biến	Bộ xử lý
1	A, B, C, D, E	B A D	
2	F, B, D, E, G	B D F G	P_0
3	B, D, A, E, G	B A D G	
4	A, B, F, G, D	B A D F G	
5	B, F, D, G, K	B D F G	P_1
6	A, B, F, G, D	B A D F G	
7	A, R, M, K, O	A	
8	B, F, G, A, D	B A D F G	P_2
9	A, B, F, M, O	B A F	

CSDL được phân tán như nhau tới 3 bộ xử lý. Mỗi bộ xử lý đếm độ hỗ trợ của các mục có mặt trong dữ liệu cục bộ. Sau phép rút gọn tổng toàn cục, tất cả các bộ xử lý lấy số đếm toàn cục đối với tất cả các mục trong CSDL. Bằng việc so sánh số đếm độ hỗ trợ với độ hỗ trợ tối thiểu, mỗi bộ xử lý lấy danh sách L, danh sách các mục phổ biến{(B:8), (A:7), (D:7), (F:6), (G:6)}. Các mục phổ biến trong mỗi giao dịch được sắp xếp theo thứ tự của L (cột thứ ba của Bảng 1). Mỗi bộ xử lý xây dựng FP-Tree cục bộ một cách độc lập. Tất cả FP-Tree cục bộ được minh họa trong Hình 1.



Hình 1. Các FP-Tree cục bộ

Việc xây dựng FP-Tree cục bộ không phải là bước cuối cùng nhưng có ý nghĩa khám phá tất cả các mẫu phổ biến mà không phải quét thêm CSDL lần nào nữa. Theo phương pháp FP-growth, để khai thác tất cả các mẫu phổ biến liên quan đến một mục i (trong bảng tiêu đề), cần xây dựng mẫu điều kiện cơ sở của i và sau đó xây dựng FP-Tree điều kiện của i là T_i. Những nhánh chứa mục i thường hiện diện trong nhiều FP-Tree cục bộ và cần thu thập chúng từ tất cả các bộ xử lý để xây dựng điều kiện cơ sở của i. Điều kiện cơ sở của một mục thay đổi theo cách phân chia CSDL ban đầu. Tuy nhiên, FP-Tree điều kiện mà được xây dựng trên điều kiện cơ sở đó không thay đổi và các tập phổ biến được khai thác từ cây cũng không bị ảnh hưởng. Để chứng minh tính chính xác của cách tiếp cận này, xét hai bổ đề sau.

Bổ đề 1. *Cách phân chia CSDL ban đầu không ảnh hưởng đến FP-Tree điều kiện của một mục.*

Chứng minh. Dựa vào tiến trình xây dựng FP-Tree, mỗi giao dịch trong CSDL ánh xạ tới một đường đi trong FP-Tree. Một đường đi đơn $i_1 \to i_2 \to \cdots \to i_n$ trong đường đi có tiền tố i_1 đăng ký tất cả các giao dịch có tập phổ biến lớn nhất theo mẫu $i_1 \to i_2 \to \cdots \to i_k$ với $1 \leqslant k \leqslant n$. Khi xây dựng FP-Tree điều kiện, một cách đơn giản là thêm các số đếm vào đường đi $i_1 \to i_2 \to \cdots \to i_n$. Vì thế sự phân tán các giao dịch không ảnh hưởng tới đường đi $i_1 \to i_2 \to \cdots \to i_n$ và số đếm của nó. Áp dụng điều này tới tất cả các đường đi của cây điều kiện ta có bổ đề.

Bổ đề 2. *Khai thác FP-Tree cục bộ thu được các mẫu phổ biến đầy đủ.*

Chứng minh. Chú ý rằng các mục trong bảng tiêu đề của tất cả các bộ xử lý là như nhau bất kể số các bộ xử lý N và cách phân tán giao dịch. Cũng chú ý rằng phép xử lý mỗi mục trong bảng tiêu đề là tác vụ độc lập. Nghĩa là mỗi mục (và mẫu điều kiện cơ sở của nó) được xử lý độc lập để sản sinh tất cả các mẫu phổ biến có liên quan mà không cần thêm thông tin nào nữa. Cuối cùng, theo Bổ đề 1, FP-tree điều kiện của một mục là bất biến, nói cách khác, các mẫu phổ biến được sản sinh từ FP-tree điều kiện đó không thay đổi. Vì vậy ta có bổ đề.

Bổ đề 2 cho thấy tính chính xác của cách tiếp cận. Do việc tính toán tại mỗi phần tử

trong bảng tiêu đề là tác vụ độc lập nên ta có thể khai thác song song tất cả các FP-Tree cục bộ bằng cách phân phối các tác vụ đó cho các bộ xử lý. Tuy nhiên, tải trọng công việc của các tác vụ đó thường không bằng nhau, điều đó dẫn tới sự mất cân bằng tải trọng công việc, và vì vậy giảm thiểu hiệu năng tính toán. Mục tiêu bài báo là tìm ra phương pháp hiệu quả để phân phối các tác vụ đó đều nhau cho các bộ xử lý và giảm thiểu sự mất cân bằng tải trọng công việc.

Thuật toán phân phối tác vụ sử dụng kỹ thuật cân bằng tải tĩnh (SLB Static Load Balancing)

Ý tưởng thuật toán là xác định khối lượng tính toán cho tất cả các mục trong bảng tiêu đề của FP-Tree ban đầu (cây được xây dựng từ cơ sở dữ liệu ban đầu) và sau đó chia những mục đó thành các phần có kích thước bằng nhau và gán chúng cho các bộ xử lý.

Cho $T = \{t_1, t_2, \ldots, t_n\}$ là tập tác vụ, với tác vụ t_i đang xử lý phần tử i trong bảng tiêu đề, và điều kiện cơ sở của nó để sản sinh tất cả các mẫu phổ biến liên quan, với n là số tác vụ.

Cho $W = \{w_1, w_2, \ldots, w_n\}$ là tập trọng số với $w_i = weight(t_i)$ là khối lượng công việc để xử lý tác vụ t_i cho đến khi kết thúc. Trọng số này có thể là một độ đo thời gian tính toán thực, hoặc độ đo biểu diễn một thời gian tương đối, liên quan tới thời gian được yêu cầu để xử lý các tác vụ khác. Trong trường hợp này, việc ước lượng thời gian tính toán thực là không thể. Ta sẽ theo cách tiếp cận ước lượng thời gian tương đối.

Để xử lý phần tử i , trước hết xây dựng cây điều kiện của i, bằng cách đệ quy sản sinh ra các mẫu điều kiện cơ sở và FP-Tree kế tiếp cho đến khi tất cả FP-Tree trở thành đường đi đơn. Số bước lặp cỡ hàm số mũ với chiều cao FP-Tree điều kiện của i. Rõ ràng là chiều cao của FP-Tree là chiều dài của đường dài nhất giới hạn bởi số tối đa các mục phổ biến trong bất kỳ giao dịch nào thuộc CSDL, hoặc số mục trong bảng tiêu đề của nó. Do đó, trọng số của tác vụ t_i có thể ước lượng bằng công thức sau

$$w_i = f \text{ (số mục trong bảng tiêu đề).}$$

Để thực hiện, ta sử dụng công thức

$$w_i = \text{số mục trong bảng tiêu đề}$$

Cho $W = \sum_{i=1}^{n} w_i$. Ta phân tán tác vụ tới các bộ xử lý sử dụng giải thuật đóng gói nhị phân (**bin-packing**). Cấp phát một "khối" cho mỗi bộ xử lý, trọng số của mỗi khối là $\frac{W}{N}$, với N là số lượng bộ xử lý. Để tăng hiệu năng song song, phân tán đều các tác vụ cùng trọng số tới càng nhiều khối càng tốt, tránh trường hợp các tác vụ với trọng số lớn được phân tán tới một vài khối. Để thực hiện điều này, các tác vụ được sắp xếp theo thứ tự giảm dần của trọng số, và gán mỗi tác vụ theo thứ tự sắp xếp tới khối bé nhất. Tiến trình này được thực hiện đồng thời trên tất cả các bộ xử lý.

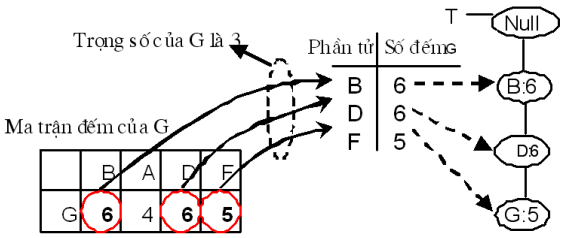
Dưới đây trình bày việc sử dụng kỹ thuật ma trận đếm để đánh giá trọng số tác vụ.

Kỹ thuật Ma trận Đếm

Trong phương pháp FP-growth, với mỗi mục i trong bảng tiêu đề của FP-Tree T_i, hai đường ngang của T_i được sử dụng để xây dựng FP-Tree điều kiện T_i. Đường ngang thứ nhất xây dựng bảng tiêu đề của cây mới bằng việc tìm tất cả các mục phổ biến trong mẫu điều kiện cơ sở của i. Đường ngang thứ hai xây dựng cây mới T_i. Trong môi trường song song, sau khi duyệt cây, tất cả các bộ xử lý cần liên lạc với nhau để hình thành thông tin toàn cục.

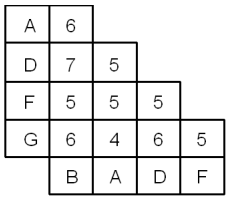
Chi phí cho việc duyệt cây và trao đổi thông tin mất rất nhiều thời gian. Như vậy, câu hỏi đặt ra là, ta có thể giảm thiểu thời gian duyệt cây và thời gian truyền thông để tăng hiệu năng giải thuật? Để ý rằng, số đếm độ hỗ trợ đối với phần tử j trong mẫu điều kiện cơ sở của phần tử i là số giao dịch mà chứa cả j và i. Nếu có một mảng lưu trữ số đếm của tất cả các cặp mục $\{i, j\}$, bảng tiêu đề của FP-Tree T_i có thể được trích lọc trực tiếp từ mảng đó, và bởi vậy, lần duyệt cây đầu tiên và thông tin trao đổi giữa các bộ xử lý đề cập ở trên sẽ được bỏ qua.

Ví dụ 2. Ví dụ này xây dựng bảng tiêu đề của cây điều kiện T_G với mục G sử dụng mảng đếm của G. Mảng đếm của G lưu trữ số đếm của tất cả các cặp mục chứa G và các mục có độ hỗ trợ nhỏ hơn G trong danh sách phổ biến L. Bằng việc so sánh các mục trên mảng đếm của G với độ hỗ trợ tối thiểu, ba phần tử $B : 6, D : 6$ và $F : 5$ được lấy để xây dựng bảng tiêu đề của FP-Tree điều kiện T_G (Hình 2).



Hình 2. Bảng tiêu đề của T_G được xây dựng từ mảng đếm G

Trong ví dụ này, chiều cao của cây T_G là 3 (không xét nút gốc rỗng), và trọng số của mục G cũng là 3. Trọng số của mục G là số mục phổ biến trong bảng tiêu đề của T_G hoặc số mục trong mảng đếm của G thỏa mãn độ hỗ trợ tối thiểu. Với cách tương tự như vậy, ta lấy bảng tiêu đề và trọng số của tất cả các mục phổ biến trong danh sách phổ biến L. Hình 3 cho thấy ma trận đếm được hình thành từ các mảng đếm của tất cả các mục phổ biến.



Hình 3. Ma trận đếm cho tất cả các mục phổ biến

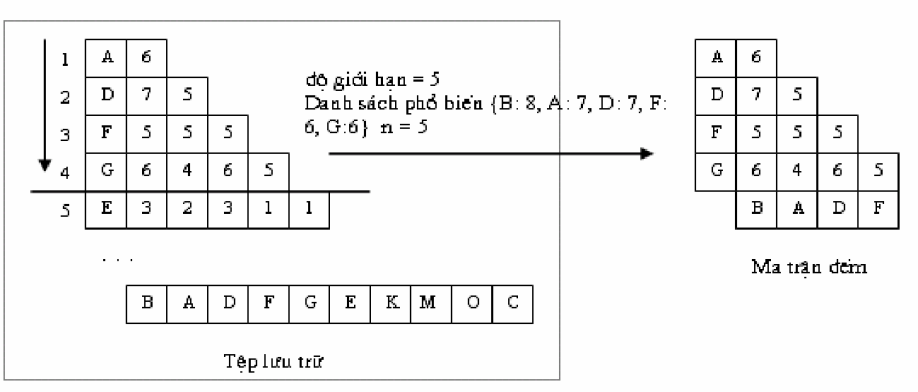
Các giải pháp để xây dựng và duy trì ma trận đếm:

- Để xây dựng ma trận trực tuyến. Số đếm của tất cả các cặp mục phổ biến sẽ được tích lũy trong lần thứ hai quét CSDL khi xây dựng FP-Tree đầu tiên T_i. Cách tiếp cận này yêu cầu một phép rút gọn tổng toàn cục để lấy ma trận toàn cục vì mỗi bộ xử lý chỉ lưu trữ số đếm cục bộ của tất cả các tập hai mục (2-itemsets) đối với tập dữ liệu cục bộ của nó. Nếu số lượng các mục phổ biến là lớn, thao tác này mất một lượng đáng kể thời gian truyền thông.

- Sử dụng bước tiền xử lý để thu thập số đếm của tất cả các tập 2 mục (2-itemsets).

Mỗi bộ xử lý sẽ giữ một bản sao của thông tin này. Khi thông tin này bất biến, nó phải thực hiện một lần trong suốt vòng đời của CSDL. Thông tin này dễ dàng được cập nhật nếu có những thay đổi trên cơ sở dữ liệu.

Ví dụ 3. Ví dụ này giải thích ý tưởng về cách tiếp cận tiền xử lý. Bước tiền xử lý yêu cầu hai lần quét CSDL. Lần quét thứ nhất lấy số đếm của tất cả các mục, sau đó các mục này được lưu trữ theo thứ tự giảm dần của số đếm. Lần quét thứ hai thu thập số đếm của tất cả các tập 2 mục (2-itemsets). Sau đó các số đếm được lưu trữ trong một tệp theo từng hàng, hàng đầu tiên chứa số đếm của mục lớn nhất thứ hai, hàng thứ hai chứa mục lớn nhất thứ ba... Ma trận đếm cho tất cả các mục phổ biến có thể được trích lọc từ tệp đó trong một cách đơn giản, ta đọc $n-1$ hàng đầu tiên vào trong các mảng, với n là số mục phổ biến (xem Hình 4).



Hình 4. Trích lọc ma trận đếm từ tệp lưu trữ

Việc áp dụng kỹ thuật ma trận đếm có thể giảm thiểu thời gian duyệt FP-Tree lần thứ hai và giảm bớt kích thước của các điều kiện cơ sở, và như vậy giảm thiểu thời gian truyền thông cần thiết để phân tán các điều kiện cơ sở cơ đó giữa các bộ xử lý. Lần duyệt FP-Tree thứ hai thu thập tất cả các mẫu phổ biến cơ sở. Đối với mục i, ta thu thập tất cả các chuỗi mà i tham gia bằng việc kiểm tra tất cả các đường đi từ những nút trong danh sách liên kết nút của i lên đến nút gốc, tất cả các mục xuất hiện trong đường đi sẽ được bổ xung vào chuỗi, nếu đã biết các mục phổ biến của cây điều kiện mới, có thể bỏ qua những mục không phổ biến khi kiểm tra những đường đi này.

Bảng 2. Mẫu điều kiện cơ sở của G

Mẫu điều kiện cơ sở của G	
Không sử dụng ma trận	Sử dụng ma trận đếm
(D:1, A:1, B:1)	(D:1, B:1)
(F:1, D:1, B:1)	(F:1, D:1, B:1)
(F:2, D:2, A:2, B:2)	(F:2, D:2, B:2)
(F:1, D:1, B:1)	(F:1, D:1, B:1)
(F:1, D:1, A:1, B:1)	(F:1, D:1, B:1)

Ví dụ 4. Ví dụ này minh họa cách thu thập mẫu điều kiện cơ sở sử dụng ma trận đếm. Đối

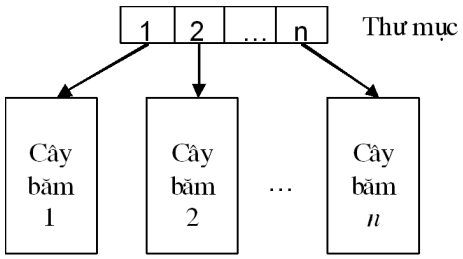
với phần tử G, bằng việc sử dụng mảng đếm của G, ngay lập tức ta có thể tìm được các mục phổ biến $\{B:6, D:6, F:5\}$ của điều kiện cơ sở của G, và mục không phổ biến $A:4$ có thể bỏ qua khi đi ngang theo các đường đi từ những nút trong danh sách liên kết nút của G tới nút gốc. Bảng 2 chứa các tiền tố đường đi liên quan đến mục G được lấy từ FP-Tree T_i.

Bây giờ thực hiện tính toán kích thước của tệp lưu trữ và kích thước của ma trận đếm. Giả sử số mục trong CSDL là n, khi đó kích thước của ma trận đếm là $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$. Chẳng hạn, một cơ sở dữ liệu với 10,000 mục sẽ sinh một ma trận với 49,995,000 ô nhớ. Kích thước ma trận đếm của các mục phổ biến nhỏ hơn nhiều so với tệp lưu trữ vì có thể hy vọng rằng số mục phổ biến nhỏ hơn n nhiều. Bài báo sử dụng cách tiếp cận tiền xử lý vì tính hiệu quả và tính đơn giản của nó.

Kỹ thuật cây băm

Chi phí truyền thông có thể tác động tới hiệu năng của bấy kỳ giải thuật song song nào. Trong giải thuật SLB, truyền thông xuất hiện chủ yếu trong tiến trình thay đổi các điều kiện cơ sở. Phân tích vấn đề này chi tiết hơn, đối với một mục trong bảng tiêu đề, từ cây FP-Tree có thể tìm được hàng triệu tiền tố đường đi, và như vậy dẫn đến một lượng thông tin khổng lồ cần trao đổi giữa các bộ xử lý. Chú ý rằng những đường đi dạng $i_1 \to i_2 \to \cdots \to i_k$ có thể xuất hiện nhiều lần, mặc dầu chúng chỉ khác nhau về số đếm độ hỗ trợ. Những đường đi này có thể được kết hợp vào trong một đường đi duy nhất với số đếm là tổng số đếm của tất cả các đường đi. Kết quả cuối cùng không bị ảnh hưởng bởi việc tính tổng này như Bổ đề 1 đã trình bày. Bằng việc sử dụng kỹ thuật cây băm, ta có thể thực hiện tiến trình này một cách hiệu quả

Tất cả các đường đi có độ dài bằng nhau được lưu trữ trong một cây băm. Ta sử dụng một thư mục là một mảng gồm n khối để trỏ tới tất cả các cây băm. Khối thứ i của thư mục trỏ vào cây băm i, nó lưu trữ tất cả các đường đi có độ dài i (Hình 5).



Hình 5. Cấu trúc dữ liệu sử dụng cây băm để cộng tất cả các đường đi giống nhau

Mỗi cây băm có hai loại nút; nút lá chứa một danh sách các tập mục và số đếm độ hỗ trợ, nút trong chứa một bảng băm gồm những con trỏ trỏ tới các nút khác. Gốc của cây băm có độ sâu là 1. Một nút bên trong với độ sâu d trỏ tới các nút ở độ sâu $d+1$. Khi thêm một đường c, $i_1 \to i_2 \to \cdots \to i_k$, trước hết, đi tới cây băm k, và sau đó bắt đầu từ gốc và đi xuống cây cho tới nút lá. Tại nút bên trong với độ sâu d, quyết định nhánh nào đi tiếp bằng việc sử dụng hàm băm cho phần tử i_d của đường đi. Tại nút lá, nếu đường đi không tồn tại, thêm đường mới vào nút này; đơn giản là tăng số đếm độ hỗ trợ bởi số đếm độ hỗ trợ của c. Khi thêm một đường đi mới vào một nút lá, nếu số đường đi vượt quá một ngưỡng xác định, nút lá được chuyển thành nút trong. Hàm băm sau đây được sử dụng

$$hash(i_1 \to i_2 \to \cdots \to i_k, d) = i_d \ \% \ \text{table\_size}$$

table_size là kích thước bảng băm của nút trong. Phần cuối của mục này sẽ mô tả thuật toán SLB dưới dạng mã giả.

Giải thuật

Giải thuật có 6 bước chính và làm việc như sau:

1. Tất cả các bộ xử lý phối hợp tìm các tập phổ biến 1 phần tử (1-itemsets).
2. Bộ xử lý P_i quét dữ liệu cục bộ của nó để xây dựng FP-Tree cục bộ.
3. P_i nạp ma trận đếm từ tệp lưu trữ và tính trọng số cho tất cả các tác vụ sử dụng ma trận đếm đó, sau đó nó sử dụng giải thuật đóng gói nhị phân để lấy khối lượng công việc của nó. Những bước này được hoàn thành đồng thời trên tất cả các bộ xử lý.
4. Bộ xử lý P_i sản sinh ra tất cả các tiền tố đường đi sử dụng FP-Tree cục bộ của nó. Tất cả các đường giống nhau liên quan tới một mục trong bảng tiêu đề được hợp nhất bằng việc bổ xung vào cây băm.
5. Bộ xử lý P_i truyền thông với các bộ xử lý khác để lấy tất cả các mẫu điều kiện cơ sở cho những mục trong khối của nó.
6. Bây giờ bộ xử lý P_i xây dựng các cây điều kiện cho những mục trong những khối của nó và thực hiện đệ quy phương pháp FP-growth để sản sinh ra tất cả các mẫu phổ biến.

Thuật toán phân phối tác vụ sử dụng kỹ thuật cân bằng tải động(DLB)

Tính hiệu quả của giải thuật SLB liên quan đến khối lượng công việc cần xử lý tại mỗi mục trong bảng tiêu đề của FP-Tree đầu tiên. Với những cây điều kiện có chiều cao tương đối nhỏ, việc ước lượng là khá chính xác. Tuy nhiên, khi chiều cao của những cây điều kiện tăng, độ chính xác có khuynh hướng giảm. Vì lý do này, một giải thuật cân bằng tải động được phát triển nhằm kiểm soát việc mất cân bằng tải trọng và phân phối lại công việc giữa các bộ xử lý.

Giai đoạn đầu của giải thuật này tương tự giải thuật SLB; từ bước 1 đền bước 5 của hai giải thuật là như nhau. Tuy nhiên, trong bước 6, DLB thực hiện theo cách khác. Khi một bộ xử lý kết thúc phần công việc của nó, nó chọn một *bộ xử lý cho* và gửi một yêu cầu công việc cho bộ xử lý này. Nếu *bộ xử lý cho* này không chứa hết công việc, nó sẽ gửi một từ chối; nếu không nó sẽ gửi phần công việc của nó tới bộ xử lý yêu cầu. Trong lúc nhận công việc mới, bộ xử lý bắt đầu xử lý công việc mới cho đến khi dừng. Một tiến trình kiểm soát được dùng để duy trì dãy tiến trình phục vụ cho việc gửi yêu cầu tiếp theo. Tiến trình này tiếp tục cho đến khi mỗi bộ xử lý đã xử lý xong tất cả các mục gán cho nó. Mỗi bộ xử lý quản lý một ngăn xếp cục bộ mà mỗi nút ngăn xếp chứa một mục và mẫu điều kiện cơ sở của nó. Thuật toán tìm điểm cuối Dijkstra - Scholten [3] được sử dụng để phát hiện có phải tất cả các bộ xử lý kết thúc công việc của chúng hay chưa và không có thông báo chuyển tiếp, và như vậy toàn bộ việc tính toán đã kết thúc. Giải thuật được minh họa trong Hình 6.

```
Procedure Dynamic_processing (local_stack)
Input: local_stack, conditional pattern bases
assigned to
       this node
Output: frequent patterns
   1. do {
   2.     Process_part_of_work(Local_stack)
   3.     Service_requests_for_work(Local_stack)
   4.  } While (Work_available(Local_stack))

Procedure Work_available (Local_stack)
   1. If Local_stack isn't empty, return TRUE
   2. If Local_stack is empty {
   3.     Send requests until we get work, return
TRUE, or
   4.     We receive a message terminating program,
          return FALSE
      }
Procedure  Process_part_of_work (Local_stack)
   1. cond_pbase = Pop(Local_stack)
   2. cond_FPtree =
   Construct_cond_FPtree(cond_pbase)
   3. FP_growth(cond_FPtree, cond_pbase.item)

Procedure
Service_requests_for_work(Local_stack)
   1. While (There are requests pending) {
   2.     dest = Get_pending_request()
   3.     if we have work to send
   4.        Send_part_of_work(dest)
   5.     if we don't have work to send
   6.        Send_rejection(dest)
      }
```

Hình 6. Mã giả cho thuật toán DLB

Quản lý tính hạt của các tác vụ

Phương pháp FP-growth, như đã trình bày, là một tiến trình đệ quy trong đó một mẫu điều kiện cơ sở có thể tạo ra một FP-Tree điều kiện mà lần lượt có thể sinh ra các mẫu điều kiện cơ sở kế tiếp nhỏ hơn. Tác vụ trong giải thuật được định nghĩa như một tiến trình của điều kiện cơ sở đối với một mục để sản sinh tất cả các mẫu phổ biến liên quan. Do đó tính hạt của tác vụ được xác định bởi số bước lặp để sản sinh các mẫu điều kiện cơ sở. Để xác định tính hạt của tác vụ có nhiệu quả như thế nào đối với hiệu năng song song, xét tình huống khi một bộ xử lý phục vụ các yêu cầu công việc. Một bộ xử lý đang hoạt động P_i có thể phục vụ các yêu cầu công việc sau khi nó đã hoàn thành việc xử lý một tác vụ trong khối của nó. Nếu trọng số của tác vụ này quá nhỏ, lượng truyền thông có thể tăng thêm bởi vì một số lượng lớn tác vụ có thể được gửi đi. Mặt khác, nếu trọng số của tác vụ này quá lớn, các bộ xử lý yêu cầu phải đợi một lượng thời gian đáng kể cho đến khi P_i có thể phục vụ cho những yêu cầu công việc. Trong cả hai trường hợp, toàn bộ hiệu năng có thể giảm.

Để kiểm soát tính hạt của tác vụ, thuật toán DLB khởi tạo một tham biến gọi là *maxload*. Những tác vụ với tính hạt lớn hơn hơn *maxload* sẽ được chia thành các tác vụ nhỏ hơn (các điều kiện cơ sở kế tiếp nhỏ hơn). Các mẫu điều kiện cơ sở kế tiếp là các tác vụ độc lập (có thể được xử lý độc lập cho đến khi những mẫu phổ biến được sản sinh), và trọng số của chúng có thể được đo trước khi thực hiện. Các mẫu điều kiện cơ sở kế tiếp với trọng số nhỏ hơn *maxload* sẽ được xử lý cho đến khi hoàn thành. Tất cả các mẫu điều kiện cơ sở kế tiếp có trọng số lớn hơn *maxload* sẽ được đặt vào trong ngăn xếp cục bộ để xử lý hoặc gửi tới những bộ xử lý khác trong tương lai.

Giá trị phù hợp của tham biến *maxload* được xác định trong thực nghiệm và sẽ được đề cập trong phần kết quả thực nghiệm.

Quản lý thời gian phục vụ

Việc kiểm soát tính hạt của tác vụ có thể giảm thiểu đáng kể thời gian chờ trong khi nhận công việc. Tuy nhiên, như đã trình bày ở phần đầu, trọng số của một tác vụ không thể đo được chính xác trước khi thực hiện. Xét tình huống khi bộ xử lý P_j gửi một yêu cầu công việc cho bộ xử lý P_i sau khi P_i đã bắt đầu thực hiện nhiệm vụ mới, P_j phải đợi cho đến khi P_i kết thúc công việc của nó. Kết quả là, bộ xử lý P_j có thể vẫn còn bị chặn trong một thời gian dài. Để giải quyết vấn đề này, ta sử dụng kỹ thuật sau. Thay vì chỉ phục vụ các yêu cầu công việc sau khi công việc của nó đã hoàn thành, nếu một yêu cầu gửi đến trong thời gian đầu xử lý phần tử mới, bộ xử lý P_i tạm thời dừng lại và phục vụ các yêu cầu công việc gửi đến, sau đó tiếp tục thực thi công việc của nó, việc này sẽ giảm thiểu thời gian chờ và như vậy cải thiện cân bằng tải.

Tối ưu hơn

Đa ma trận đếm. Như đã thảo luận ở trên, việc sử dụng ma trận đếm có thể giảm thiểu thời gian duyệt FP-Tree khi xây dựng FP-Tree điều kiện kế tiếp. Trong giải thuật cân bằng tải tĩnh, ta chỉ xây dựng ma trận đếm cho FP-Tree đầu tiên T_i. Rõ ràng, ý tưởng sử dụng ma trận đếm có thể được áp dụng cho bất kỳ FP-Tree T_χ nào. Chú ý rằng mẫu điều kiện cơ sở khi ta xây dựng T_χ có thể coi như một tập dữ liệu. Không giống ma trận đếm đối với T_i mà được nạp từ một tệp lưu trữ, ma trận đếm đối với T_χ sẽ được xây dựng trong suốt lần quét thứ hai mẫu điều kiện cơ sở. Khi xây dựng ma trận đếm đối với FP-Tree T_χ ($\chi \neq \emptyset$), không có sự trao đổi thông tin giữa các bộ xử lý vì cây và mẫu điều kiện cơ sở cần thiết để xây dựng cây hiện diện trong cùng bộ xử lý.

Thứ tự xử lý tác vụ. Trong DLB, thứ tự các tác vụ trong danh sách cục bộ được xử lý có thể tác động tới toàn bộ hiệu năng của giải thuật. Xét tình huống khi lần đầu tiên, bộ xử lý P_i xử lý những tác vụ có trọng số nhỏ. Giai đoạn đầu, không có yêu cầu công việc từ các bộ xử lý khác gửi đến P_i. Các yêu cầu công việc chỉ đến khi P_i kết thúc việc xử lý tất cả các tác vụ có trọng số nhỏ và bắt đầu với tác vụ có trọng số lớn hơn trong khối của nó, điều này có thể bắt các bộ xử lý chờ lâu trước khi chúng nhận công việc mới từ P_i. Vì lý do đó, các tác vụ trong mỗi khối được sắp xếp theo thứ tự giảm dần của trọng số để xử lý các tác vụ lớn hơn trước.

Tránh bị chặn khi trao đổi các mẫu điều kiện cơ sở. Sau khi các FP-Tree cục bộ được xây dựng trong tất cả các bộ xử lý, những tiền tố đường đi lấy từ những cây cần phải trao đổi giữa các bộ xử lý để hình thành các mẫu điều kiện cơ sở. Tiến trình trao đổi này đòi hỏi sự phối hợp của tất cả các bộ xử lý và dẫn tới việc bị chặn. Để giảm bớt khả năng bị chặn, mỗi bộ xử lý chọn ngẫu nhiên một bộ xử lý đích để gửi thay vì sử dụng lược đồ hình tròn. Tất cả các tiền tố đường đi liên quan đến một phần tử trong một bộ xử lý sẽ được đóng gói trước khi gửi đi. Kỹ thuật bộ đệm cũng được sử dụng để giảm thiểu khả năng bị chặn.

4. NGHIÊN CỨU HIỆU NĂNG

Các kết quả thực nghiệm được thực hiện trên PC Atlantis Cluster gồm có 16 nút. Mỗi nút có 2 bộ xử lý Intel Xeon 2.8GHz, 2GB bộ nhớ, và ổ cứng 80GB. Các nút được kết nối với nhau bởi Ethernet có dải thông 1Giga-bits/s. Cơ sở dữ liệu được lưu trên đĩa cục bộ của mỗi nút. Chương trình ứng dụng được lập trình bằng ngôn ngữ C. Sử dụng giao thức truyền tin-MPI. để thực hiện truyền thông.

Để đánh giá hiệu năng, ta sử dụng các tập dữ liệu giả định khác nhau được sinh ra bằng việc sử dụng thủ tục được mô tả trong[2]. Các thuộc tính của chúng được mô tả trong Bảng 3.

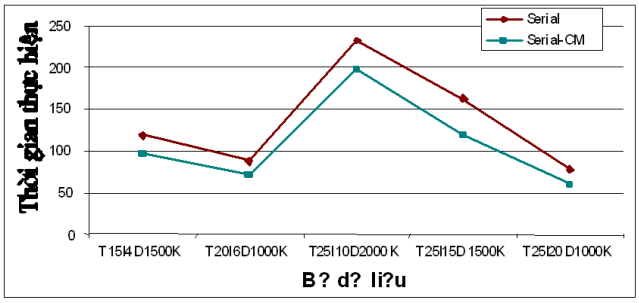
Bảng 3. Thuộc tính tập dữ liệu

Tập dữ liệu	T	I	D	N	Kích thước (MB)
T15I4D2000K	15	4	1,500,000	10,000	223
T20I6D1000K	20	6	1,000,000	10,000	197
T25I10D200K	25	10	2,000,000	10,000	492
T25I15D150K	25	15	1,500,000	10,000	369
T25I20D100K	25	20	1,000,000	10,000	249

T: Độ dài trung bình của giao dịch
I: Độ dài trung bình của tập phần tử phổ biến
D: Số giao dịch
N: Số phần tử

Tính hiệu quả của việc sử dụng ma trận đếm

Trong bài báo này, ta so sánh hiệu năng của giải thuật FP-growth ban đầu với giải thuật sử dụng kỹ thuật ma trận đếm. Hình 7 cho thấy thời gian thực hiện của hai giải thuật trên năm tập dữ liệu của một nút. Trong hình 7, giải thuật đầu tiên có tên "Serial", và giải thuật với ma trận đếm có tên "Serial-CM". Độ hỗ trợ tối thiểu 0.1%. Ta thấy rằng với việc sử dụng ma trận đếm, giải thuật thực hiên tốt hơn giải thuật ban đầu trên tất cả các tập dữ liệu. Trong tất cả các thí nghiệm khác với những giải thuật song song dựa trên cây, khi đề cập đến giải thuật FP-growth, ta sử dụng giải thuật FP-growth với kỹ thuật ma trận đếm.



Hình 7. Hiệu quả của việc sử dụng ma trận đếm

Hiệu quả của việc sử dụng kỹ thuật cây băm

Nghiên cứu hiệu năng kiểm tra tính hiệu quả của việc sử dụng kỹ thuật cây băm khi tất

cả các bộ xử lý trao đổi các điều kiện cơ sở. Ta đếm số tiền tố đường đi mà mỗi bộ xử lý phải gửi tới các bộ xử lý khác và so sánh nó với số đường đi sau khi được hợp nhất bằng việc sử dụng kỹ thuật cây băm. Việc kiểm tra được thực hiện trên tập dữ liệu T25I20D1000K với độ hỗ trợ tối thiểu là 0.1%. Những kết quả được đưa ra ở Bảng 4.

Bảng 4. Hiệu quả của việc sử dụng kỹ thuật cây băm

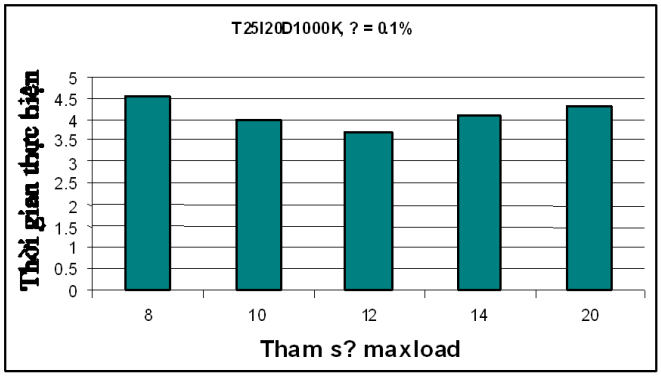
Số nút	**Bộ xử lý**	**Trước khi hợp nhất**	**Sau khi hợp nhất**
2	0	144,639	3,753
	1	148,873	3,951
4	0	112,369	4,637
	1	115,407	4,821
	2	113,148	4,570
	3	112,295	4,732
8	0	68,332	4,277
	1	67,047	4,294
	2	67,938	4,191
	3	68,839	4,405
	4	67,331	4,304
	5	68,160	4,222
	6	67,660	4,323
	7	68,570	4,409
16	0	37,890	3,441
	1	37,779	3,584
	2	37,114	3,467
	3	37,323	3,476
	4	37,834	3,512
	5	38,223	3,474
	6	39,032	3,672
	7	38,041	3,535
	8	37,126	3,518
	9	38,163	3,581
	10	37,693	3,472
	11	37,652	3,435
	12	37,670	3,510
	13	37,119	3,433
	14	37,636	3,514
	15	39,494	3,710

Từ Bảng 4 có thể thấy rằng, bằng việc hợp nhất các tiền tố đường đi giống nhau liên quan đến các mục trong bảng tiêu đề, số đường đi cần thiết để gửi đến các bộ xử lý khác giảm thiểu khá nhiều, và như vậy, chi phí truyền thông cũng giảm thiểu đáng kể.

Giá trị tối ưu của *maxload*

Như đã trình bày trong phần đầu, giá trị *maxload* có thể ảnh hưởng đến hiệu năng của DLB. Khi *maxload* quá lớn, việc mất cân bằng tải sẽ tăng thêm. Ngược lại, nếu *maxload* quá nhỏ, lượng truyền thông có thể tăng lên, ngoài ra có quá nhiều mẫu điều kiện cơ sở nhỏ được lưu trữ trong ngăn xếp cục bộ, do đó làm tăng thời gian tính toán. Tuy nhiên, thật khó để xác định giá trị tối ưu của *maxload* khi nó ảnh hưởng đến một thuộc tính của tập dữ liệu-kích thước mẫu phổ biến trung bình.

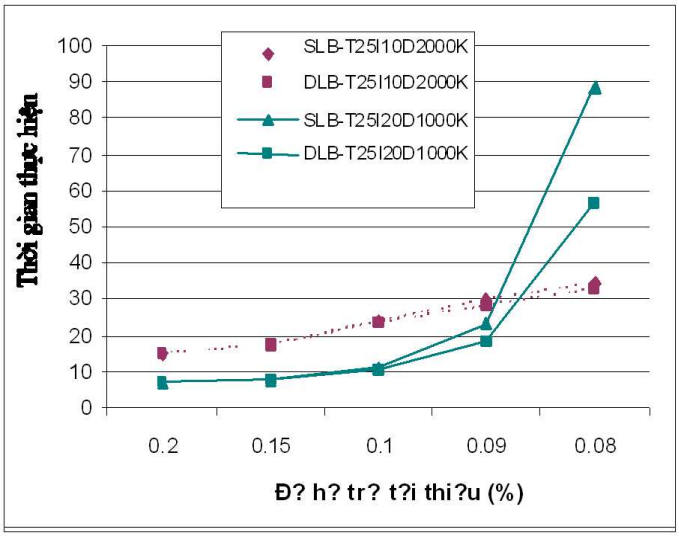
Thực hiện một số thí nghiệm trên những tập dữ liệu khác nhau để đánh giá mức độ ảnh hưởng của *maxload* đến hiệu năng. Kết quả thí nghiệm cho kết luận rằng, giải thuật đạt hiệu năng tốt nhất nếu *maxload* = 2/3 kích thước mẫu phổ biến trung bình. Hình 8 minh họa kết quả thử nghiệm trên tập dữ liệu T25I20D1000K có kích thước mẫu phổ biến trung bình $I = 20$. Tiến hành đo thời gian thực hiện giai đoạn hai của giải thuật (khai thác từ FP-Tree) vì tham biến *maxload* không ảnh hưởng tới giai đoạn đầu của giải thuật (xây dựng cây cục bộ). Trong nghiên cứu này, độ hỗ trợ tối thiểu là 0.1%, và số lượng nút là 4. Giải thuật đạt được hiệu năng tốt nhất khi *maxload*=12.



Hình 8. Ảnh hưởng của việc sử dụng các giá trị *maxload* khác nhau

Khoảng chia độ hỗ trợ tối thiểu của những giải thuật dựa trên cây

Nghiên cứu này so sánh tính ổn định của SLB và DLB khi độ hỗ trợ tối thiểu giảm từ 0.25% đến 0.08%. Các thí nghiệm được thực hiện trên 8 nút với hai tập dữ liệu, T25I10D2000K và T25I20D1000K. Kết quả được minh họa trong Hình 9.



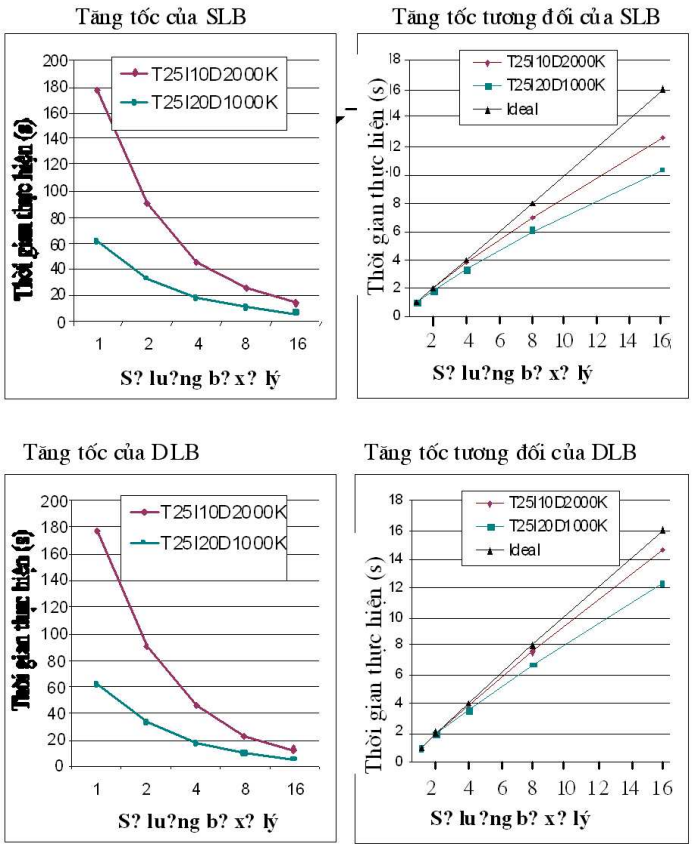
Hình 9. Khoảng chia độ hỗ trợ tối thiểu

Khi độ hỗ trợ tối thiểu giảm, các tập mục phổ biến trong cả hai tập dữ liệu tăng với cấp lũy thừa. Có khá nhiều tập mục phổ biến dài cũng như số lớn các tập mục phổ biến ngắn

trong các tập dữ liệu. Trên tập dữ liệu T25I10D2000K, độ chia của SLB và DLB là bằng nhau Tuy nhiên, độ chia DLB tốt hơn hơn SLB trên tập dữ liệu T25I20D1000K. Điều này là do, với tập dữ liệu T25I20D1000K, khi độ hỗ trợ tối thiểu giảm, số tập phổ biến dài tăng đột ngột so với tập dữ liệu T25I10D2000K. Như đã trình bày trong phần trước, việc đo trọng số của các FP-Tree điều kiện dài hoặc các tập phổ biến dài không chính xác bằng việc đo FP-Tree điều kiện và các tập phổ biến ngắn và do đó dẫn tới việc mất cân bằng tải.

Tăng tốc SLB và DLB

Hình 10 chỉ ra việc tăng tốc của giải thuật SLB và DLB trên hai tập dữ liệu khi độ hỗ trợ tối thiểu là 0.1%. Để nghiên cứu hiệu năng, ta giữ CSDL không thay đổi và thay đổi số lượng bộ xử lý. Việc tăng tốc tương đối được đo bằng việc sử dụng phương trình $S_p = T_1/T_p$ với S_p là sự tăng tốc đạt được với p bộ xử lý, T_1 là thời gian thực hiện tuần tự và T_p là thời gian thực hiện sử dụng p bộ xử lý. Như đồ thị minh họa, cả SLB và DLB đều tăng tốc về hiệu năng rất tốt. Tuy nhiên, trên hai tập dữ liệu, DLB có tỷ lệ tăng tốc tốt hơn. Có thể thấy rằng, việc tăng số lượng bộ xử lý, hệ số góc của đường tốc độ vượt qua giới hạn. Đó là vì, khi mỗi nút xử lý số lượng dữ liệu nhỏ, thời gian truyền thông chiếm phần đáng kể trong tổng thời gian thực hiện. Có nghĩa là, có thể đạt được tỷ lệ tăng tốc tốt hơn với những tập dữ liệu lớn hơn trong số lớn các giao dịch, khi đó, thời gian truyền thông sẽ ít ảnh hưởng.



Hình 10. Tăng tốc hiệu năng của các giải thuật dựa trên cây

5. KẾT LUẬN

Bài báo này đã trình bày quá trình phát triển hai giải thuật song song dựa trên phương pháp FP-growth. Giải thuật SLB dùng kỹ thuật ma trận đếm để ước lượng trọng số của các tác vụ song song được xử lý, và do đó cân bằng tải giữa các bộ xử lý. Kỹ thuật ma trận đếm cũng giúp giảm bớt thời gian duyệt FP-Tree và giảm thiểu thông tin dư thừa trao đổi giữa các bộ xử lý. SLB liên quan đến các bài toán truyền thông mức cao bằng việc sử dụng kỹ thuật cây băm để kết hợp tất cả các tiền tố đường đi giống nhau liên quan đến các phần tử phổ biến lấy từ FP-Tree.

Giải thuật DLB là giải pháp cân bằng tải động được xây dựng dựa trên giải thuật SLB. DLB kế thừa tất cả các ưu điểm của giải thuật SLB. Hơn nữa, DLB có khả năng phân phối lại tải trọng công việc giữa các bộ xử lý khi hiện tượng mất cân bằng tải xẩy ra. Một số giải pháp tối ưu được phát triển để giúp DLB đạt được hiệu năng tính toán cao hơn.

Các giải thuật khai thác dữ liệu song song SLB và DLB phát triển trên phương pháp FP-growth đã được đề xuất trong [1], [4], [8], [11]. Tuy nhiên, các giải thuật này gặp nhiều hạn chế về hiệu năng và thời gian thực hiện. Đóng góp chính của bài báo là cải tiến giải thuật DLB bằng việc sử dụng kỹ thuật đa ma trận đếm và điều chỉnh thứ tự xử lý các tác vụ. Hơn nữa, bài báo tiến hành cài đặt và thử nghiệm các giải thuật trên bộ số liệu thử nghiệm để đánh giá hiệu năng của các giải thuật, từ đó đưa ra các kết luận về tính hiệu quả của việc sử dụng đa ma trận đếm, tìm được giá trị tốt nhất của tham số *maxload* để giải thuật đạt hiệu năng tốt nhất. Bài báo cũng thử nghiệm mối liên quan giữa số lượng các bộ xử lý được sử dụng, độ lớn của bộ số liệu thử nghiệm, độ hỗ trợ tối thiểu với hiệu năng của các giải thuật bằng việc thực hiện tăng tốc giải thuật SLB và DLB. Kết quả thử nghiệm chỉ ra rằng cả SLB lẫn DLB có sự tăng tốc về hiệu năng rất tốt. Giữa hai giải thuật, DLB thực hiện tốt hơn hơn SLB, đặc biệt trên những tập dữ liệu rất lớn với độ hỗ trợ tối thiểu nhỏ.

TÀI LIỆU THAM KHẢO

[1] R. Agrawal and J. Shafer, Parallel mining of association rules, *IEEE Transactions on Knowledge and Data Eng.*, USA, 1996 (962–969).

[2] R. Agrawal, T. Imielinski, and A. Swami, Mining association rules between sets of items in large databases, *Proc. of the ACM SIGMOD Conf.* Washington DC, USA, May, 1993.

[3] E. W. Dijkstra, W. H. Seijen, and A. J. M. Van Gasteren, Derivation of a termination detection algorithm for a distributed computation, *Proc. of the ACM Transactions on Programming Languages and Systems*, ACM Press, New York, USA, 1993 (1–35).

[4] E. H. Han, G. Karypis, and V. Kumar, Scalable parallel data mining for association rules, *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, ACM Press, New York, USA, 1997 (277–288).

[5] J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation, *Proc. of the ACM SIGMOD Conf. on Management of Data*, ACM Press, New York, USA, 2000 (1–12).

[6] A. Mueller, Fast sequential and parallel algorithms for association rule mining: A comparison, *Technical Report CS-TR-3515*, College Park, MD, 1995.

[7] J. Park, M. Chen, and P. Yu., An effective hash-based algorithm for mining association rules, *Proc. ACM SIGMOD Conf. Management of Data*, San Jose, California, United States, 1995 (175–186).

[8] J. Park, M. Chen, and P. Yum, An efficient parallel data mining for association rules, *Proceedings of the 4th International Conference on Information and Knowledge Management*, ACM Press, New York, USA, 1995 (31–36).

[9] A. Savasere, E. Omiecinski, and S. Navathe, An efficient algorithm for mining association rules in large databases, *Proc. of the 21st VLDB Conf.* Zurich, Swizerland, 1995.

[10] R. Srikant and R. Agrawal, Mining sequential patterns: Generalizations and performance improvements, *Proc. of the Fifth Intel Conference on Extending Database Technology*, Springer-Verlag, USA, 1996 (3–17).

[11] M. J. Zaki, S. Parthasarathy, and W. Li, A localized algorithm for parallel association mining, *ACM Symposium Parallel Algorithms and Architectures,* ACM Press, New York, USA, 1997 (321–330).

Nhận bài ngày 3 - 4 - 2007