ISSN 2354-0575

MỘT CÁCH TIẾP CẬN MỚI CHO BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT TRÊN ĐỒ THỊ PHÂN TÁN

Nguyễn Thị Huyền, Phạm Đăng Hải
Trường Đại học Bách khoa Hà Nội

Ngày nhận: 17/2/2016
Ngày xét duyệt: 15/3/2016

Tóm tắt:

Gần đây, hiệu quả của việc truy vấn thông tin trên các đồ thị lớn trở thành một chủ đề quan trọng trong khoa học máy tính. Một câu truy vấn được sử dụng rộng rãi đó là "tìm đường đi ngắn nhất giữa hai đỉnh của đồ thị", một bài toán mà có thể được giải bằng một vài thuật toán nổi tiếng như Dijkstra, Johnson, hay Floyd-Warshall; tuy nhiên, nó là không đơn giản để trả lời câu truy vấn này trên một đồ thị mà dữ liệu phân tán ở nhiều vị trí khác nhau. Trong bài báo này, chúng tôi đề xuất một cách tiếp cận mới dựa trên kỹ thuật ước lượng từng phần để giải quyết bài toán tìm đường đi ngắn nhất giữa hai đỉnh trên một đồ thị phân tán. Chúng tôi chỉ ra rằng thuật toán đề xuất có thể được cài đặt dưới hình thức song song trên nền tảng MapReduce. Bằng việc sử dụng một tập dữ liệu trong thực tế cho thực nghiệm, chúng tôi tiến hành thực nghiệm và chỉ ra được thuật toán của chúng tôi có khả năng mở rộng cho các đồ thị lớn trên các hệ thống phân tán.

Từ khóa: *Truy vấn đồ thị, MapReduce, Đường đi ngắn nhất, Đồ thị phân tán, Ước lượng từng phần.*

1. Đặt vấn đề

Trong các ứng dụng thực tế, bài toán tìm đường đi ngắn nhất giữa hai đỉnh của một đồ thị liên thông có một ý nghĩa to lớn. Ví dụ như, bài toán chọn một hành trình tiết kiệm nhất (theo tiêu chuẩn hoặc khoảng cách hoặc thời gian hoặc chi phí) trên một mạng giao thông đường bộ, đường thủy hoặc đường không; bài toán chọn một phương pháp tiết kiệm nhất để đưa ra một hệ thống động lực từ trạng thái xuất phát đến một trạng thái đích, bài toán lập lịch thi công các công đoạn trong một công trình thi công lớn, bài toán lựa chọn đường truyền tin với chi phí nhỏ nhất trong mạng thông tin, v.v…

Bài toán tìm đường đi ngắn nhất có thể phát biểu dưới dạng hình thức như sau: Cho trước một đồ thị có trọng số G = (V, E), trong đó V là một tập đỉnh, E là một tập cạnh, hãy tìm một đường đi ngắn nhất từ đỉnh xuất $s \in V$ đến đỉnh đích $t \in V$ [1]. Bài toán đường đi ngắn nhất giữa mọi cặp đỉnh là một bài toán tương tự, trong đó ta phải tìm các đường đi ngắn nhất cho mọi cặp đỉnh ***s*** và ***t***.

Trong lý thuyết đồ thị, đã có nhiều thuật toán được đề xuất để giải quyết bài toán tìm đường đi ngắn nhất. Các thuật toán quan trọng nhất giải quyết bài toán này bao gồm:

• **Thuật toán Dijkstra**: giải bài toán nguồn đơn nếu tất cả các trọng số đều không âm. Thuật toán này có thể tính toán tất cả các đường đi ngắn nhất từ một đỉnh xuất phát cho trước s tới mọi đỉnh khác mà không làm tăng thời gian chạy.

• **Thuật toán Bellman-Ford**: giải bài toán nguồn đơn trong trường hợp trọng số có thể có giá trị âm.

• **Giải thuật tìm kiếm A***: giải bài toán nguồn đơn sử dụng heuristics để tăng tốc độ tìm kiếm.

• **Thuật toán Floyd-Warshall**: giải bài toán đường đi ngắn nhất cho mọi cặp đỉnh.

• **Thuật toán Johnson**: giải bài toán đường đi ngắn nhất cho mọi cặp đỉnh, có thể nhanh hơn thuật toán Floyd-Warshall trên các đồ thị thưa.

Các thuật toán trên được xây dựng để giải quyết các bài toán đồ thị tổng quát, ở đó dữ liệu tập trung tại một máy tính đơn nhất. Tuy nhiên, nó là không đơn giản để trả lời câu truy vấn này trên một đồ thị lớn mà dữ liệu phân tán ở nhiều vị trí khác nhau. Trong bài báo này, chúng tôi đã đề xuất một thuật toán dựa trên kỹ thuật ước lượng từng phần và khai thác nền tảng hỗ trợ xử lý dữ liệu song song MapReduce [2] để giải quyết bài toán tìm đường đi ngắn nhất nêu trên.

2. Cơ bản về thuật toán Dijkstra

Dijkstra là một thuật toán giải quyết bài toán đường đi ngắn nhất nguồn đơn trong một đồ thị có hướng, mà trọng số trên các cung không âm [1].

Thuật toán được xây dựng dựa trên cơ sở gán cho các đỉnh các nhãn tạm thời. Nhãn của mỗi đỉnh cho biết cận của độ dài đường đi ngắn nhất từ *s* đến nó. Các nhãn này sẽ được biến đổi theo một thủ tục lặp, mà ở mỗi bước lặp có một nhãn tạm thời trở thành nhãn cố định. Nếu nhãn của một đỉnh nào đó trở thành một nhãn cố định thì nó sẽ cho ta không phải là cận trên mà là độ dài của đường đi ngắn nhất từ đỉnh *s* đến nó. Thuật toán được mô tả cụ thể như sau:

Thuật toán 1: *Thủ tục **Dijkstra** tìm đường đi ngắn nhất từ một đỉnh đến các đỉnh*

Đầu vào: một đồ thị G = (V, E), đỉnh nguồn *s*
Đầu ra: độ dài đường đi ngắn nhất từ đỉnh *s* đến các đỉnh còn lại trong đồ thị

```
1:      for each vertex v in G        // khởi tạo
2:          dist[v] ← ∞;              // khởi tạo giá trị từ s tới đỉnh v = ∞
3:          previous[v] ← null; // đỉnh trước của đỉnh v
4:      dist[source] ← 0;    // khoảng cách từ nguồn tới nguồn
5:      while V is not empty do
6:          u ← đỉnh có thuộc V có khoảng cách tới s là nhỏ nhất;
7:          V ← V/{u};
8:    for each neighbor v of u
9:     alt ← dist[u] + dist_between(u, v)
10:              if alt < dist[v]          then
11:                  dist[v] ← alt;
12:                  previous[v] ← u;
13:     return previous[ ];
```

Thuật toán có độ phức tạp là $O(n^2)$. Do độ phức tạp tính toán cao, việc giải bài toán này với tính chất tuần tự gặp phải bất lợi lớn về thời gian thực hiện chương trình, tốc độ xử lý, khả năng lưu trữ,... Đặc biệt là trên đồ thị có hàng triệu đỉnh và cạnh mà thời gian chạy phải được rút gọn thì thuật toán tuần tự không thực hiện được.

Điều này đặt ra yêu cầu phải chia đồ thị cho một hệ thống phân tán có nhiều máy tính cùng tham gia tính toán đồng thời, song việc chia đồ thị thành các đồ thị nhỏ thì việc lưu trữ các đồ thị nhỏ đó trên hệ thống file phân tán và việc sử dụng thuật toán tìm đường đi ngắn nhất trên hệ thống phân tán đó trở thành một bài toán với nhiều thách thức.

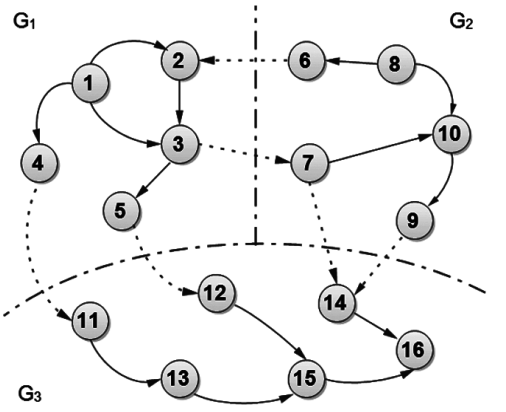
3. Đồ thị phân tán

Cơ sở dữ liệu phân tán là tuyển tập dữ liệu có quan hệ logic với nhau, được phân bố trên các máy tính của một mạng máy tính. Cũng giống như cơ sở dữ liệu phân tán, đồ thị phân tán là một tập các đồ thị con liên thông với nhau bởi các cạnh của đồ thị và các đồ thị này được đặt phân tán trên một hệ thống mạng máy tính.

Trên thực tế chúng ta thấy, dữ liệu của một đồ thị G đặc trưng cho một hệ thống có thể được chia ra làm *k* phần khác nhau nằm ở các máy thuộc các vị trí địa lý khác nhau, ở đó mỗi phần được coi là một đồ thị con (sub-graph). Các đồ thị con này liên thông với nhau bởi tập các cạnh của đồ thị. Một cách hình thức hóa có thể định nghĩa đồ thị phân tán như sau:

Định nghĩa 1: Đồ thị phân tán G = (V, E) là đồ thị bao gồm một tập các *đồ thị con* từ $G_1, G_2, \ldots, G_k$ nằm trên các máy tính khác nhau và một *đồ thị liên kết* giữa chúng G_c. Trong đó, một đồ thị con G_i được định nghĩa bởi (V_i, E_i), với $V_i \in V$ và $E_i \in E$; đồ thị liên kết $G_c = (V_c, E_c)$, ở đó E_c là tập các cạnh kết nối các đồ thị con với nhau (gọi là *cạnh liên kết*) và V_c là tập các đỉnh có các cạnh liên kết.

Ví dụ 1:



Hình 1. *Minh họa đồ thị phân tán*

Hình 1 ở trên là một ví dụ minh họa đồ thị phân tán. Trong ví dụ này, đồ thị phân tán G gồm 3 đồ thị con G_1, G_2 và G_3 và một đồ thị liên kết G_c với các cạnh được vẽ bằng các nét đứt. Trong ví dụ này, dữ liệu của mỗi đồ thị con và đồ thị liên kết bao gồm như sau:

Bảng 1. *Dữ liệu trên các đồ thị con và đồ thị liên kết*

Đồ thị	V_i	E_i
$G_1 = (V_1, E_1)$	{1, 2, 3, 4, 5}	{(1,2); (1,3); (1,4); (2,3); (3,5)}
$G_2 = (V_2, E_2)$	{6, 7, 8, 9, 10}	{(7,10); (8,6); (8,10); (10,9)}
$G_3 = (V_3, E_3)$	{11, 12, 13, 14, 15, 16}	{(11,13); (12,15); (13,15); (14,16); (15,16)}
$G_c = (V_c, E_c)$	{2, 3, 4, 5, 6, 7, 9, 11, 12, 14, 16}	{(3,7); (4,11); (5,12); (6,2); (7,14); (9,14)}

4. Đề xuất thuật toán tìm đường đi ngắn nhất trên đồ thị phân tán

Để tìm kiếm đường đi ngắn nhất trên đồ thị phân tán, chúng tôi đã đề xuất một thuật toán theo kỹ thuật ước lượng từng phần.

Ước lượng từng phần

Kỹ thuật ước lượng từng phần được đưa ra trong [3]. Kỹ thuật này trình bày một vài kiểu tối ưu hóa chương trình theo những cách đặc biệt nhằm mục tiêu tăng tốc độ xử lý. Ở đó, một chương trình

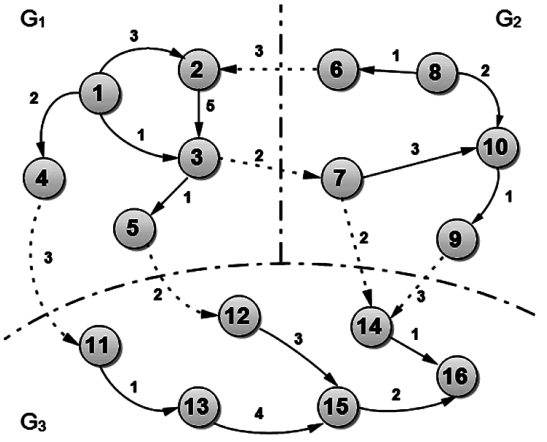
xử lý **p** được chia ra làm *k* phần (p_1, p_2, …,p_k) cái mà có thể thực thi riêng rẽ và đảm bảo thực hiện theo cùng một cách. Một chương trình p_i sẽ thực hiện nhanh hơn thực thi chương trình **p**. Kết quả của p_i là tập con trong kết quả của **p**. Bằng việc áp dụng kỹ thuật ước lượng từng phần này, các công trình nghiên cứu trong [4, 5, 6] đã đề xuất các thuật toán hiệu quả trong việc ước lượng truy vấn trên đồ thị phân tán.

Đề xuất thuật toán

Trong phần này, chúng tôi trình bày đề xuất một thuật toán tìm đường đi ngắn nhất trên đồ thị phân tán. Ở đây, cách tiếp cận là sử dụng thuật toán Dijkstra kết hợp với kỹ thuật ước lượng từng phần. Để làm được điều đó, chúng tôi đã nghiên cứu mối liên hệ giữa đường đi ngắn nhất trên toàn bộ đồ thị và đường đi ngắn nhất trên từng phần đồ thị. Các kỹ thuật này sẽ được trình bày trong các phần dưới đây.

Định nghĩa 2: Đồ thị phân tán có trọng số là một đồ thị phân tán mà trên các cạnh được gán một giá trị số (số nguyên hoặc số thực).

Để áp dụng cho bài toán tìm đường đi ngắn nhất, chúng tôi sử dụng một đồ thị phân tán có trọng số. Bằng cách gán các trọng số vào đồ thị phân tán của Hình 1, chúng ta có một đồ thị phân tán có trọng số như trong Hình 2.



Hình 2. *Ví dụ đồ thị phân tán có trọng số*

Trong Hình 2, các trọng số được đưa vào được xem như độ dài đường đi giữa hai địa điểm (có thể sử dụng đơn vị là kilometer).

Ví dụ 2: Giả sử chúng ta có một đồ thị phân tán có trọng số G như trong Hình 2 là một mạng đường của các tỉnh thành. Ở đó, G_1, G_2, và G_3 tương ứng 3 tỉnh thành khác nhau lần lượt là Hà Nội, Hưng Yên và Hải Dương. Dữ liệu mạng đường đi của các tỉnh thành là rất lớn và được lưu trữ riêng biệt tại các máy tính khác nhau, có kết nối với nhau qua một hệ thống mạng. Trên mỗi mạng đường đi của một tỉnh thành có tồn tại các điểm của các tỉnh thành lân cận. Một tình huống thực tế là: một người đang ở địa điểm (1) tại Hà Nội muốn đến thăm một người bạn ở địa điểm số (16) tại Hải Dương. Trên thực tế, có rất nhiều tuyến đường từ Hà Nội đến Hải Dương, có tuyến trực tiếp qua hai thành phố, nhưng cũng có nhiều tuyến đi liên tỉnh qua Hưng Yên rồi mới đến Hải Dương. Hãy giúp anh ấy tìm ra một tuyến đường ngắn nhất đi từ (1) đến (16) trên hệ thống dữ liệu đã có. Trong các phần tiếp theo, chúng tôi sẽ tập trung giải quyết bài toán này.

Một vài quan sát và suy luận:

Như định nghĩa đồ thị phân tán ở trên, mối liên hệ giữa các đồ thị con được xác định thông qua các đỉnh và cạnh của đồ thị liên kết. Một đỉnh thuộc đồ thị con này có thể là đỉnh đích của một cạnh mà đỉnh thuộc đồ thị con khác. Tất cả các đỉnh cạnh như vậy đều thuộc đồ thị liên kết. Để tổng quát hóa mối quan hệ giữa chúng, chúng tôi đưa ra các định nghĩa sau:

Định nghĩa 3: Đỉnh liên kết trong (*input node*) của một đồ thị con *Gi = (Vi, Ei)*, là một đỉnh thuộc tập đỉnh V_i mà tồn tại ít nhất một cạnh từ một đỉnh thuộc đồ thị con khác đến nó.

Định nghĩa 4: Đỉnh ảo ngoài (*output node* hay *virtual node*) của một đồ thị con $G_i = (V_i, E_i)$, là một đỉnh không thuộc tập đỉnh V_i nhưng tồn tại ít nhất một cạnh từ một đỉnh thuộc V_i đến nó.

Từ hai định nghĩa trên, chúng ta có thể đưa ra mối liên hệ giữa đồ thị G, các đồ thị con G_i và đồ thị liên kết G_c như sau:

• $V_c = U_{i=1}^{k}(V_i.in \cup V_i.out)$, trong đó $V_i.in$ là tập các *input node* của đồ thị con G_i và $V_i.in \subseteq V_i$; $V_i.out$ là tập các *output node* của đồ thị con G_i và $V_i.out \cap V_i = \emptyset$;

• $E_c = U_{i=1}^{k} cE_i$, trong đó cE_i là tập tất cả các cạnh liên kết thuộc G_i, mỗi cạnh thuộc (v, u) $\in cE_i$ là được xác định bởi $v \in V_i.in$ và $u \in V_i.out$.

• $V = U_{i=1}^{k}(V_i)$ và $V_i \cap V_j = \emptyset$ if $i \neq j$; $E = E_c \cup (U_{i=1}^{k} E_i)$ và $E_i \cap E_j = \emptyset$ if $i \neq j$;

Trên thực tế, dữ liệu lưu trữ ở mỗi máy bao gồm một đồ thị con G_i, một tập các *output node Vi.out* và tập các cạnh liên kết cE_i. Toàn bộ dữ liệu lưu trên một máy gọi là một mảnh (fragment) của đồ thị G, kí hiệu bởi $F_i = (V_i \cup V_i.out, E_i \cup cE_i)$). Như vậy, việc xử lý truy vấn trên mỗi máy nghĩa là chúng ta đang xử lý trên một mảnh của đồ thị G.

Bảng 2. *Minh họa input và output node của đồ thị phân tán*

F_i	V_i.in	V_i.out
F_1	{2}	{7, 11, 12}
F_2	{7}	{2, 14}
F_3	{11, 12, 14}	{∅}

Bảng 2 chỉ ra tập các input node và output node trên các mảnh đồ thị. Tuy nhiên, trong bảng trên có hai giá trị đặc biệt được thêm vào: đỉnh (1) là đỉnh xuất nguồn nên nó được thêm vào danh sách *input node* của phần đồ thị con chứa nó (G_1) và đỉnh (16) là đỉnh đích nên nó được thêm vào danh sách *output node* của phần đồ thị con chứa nó (G_3).

Như vậy, chúng ta có câu truy vấn Q(1, 16). Việc trả lời câu truy vấn Q trên đồ thị G là tương đương với việc tìm ra một đường đi P(Q) = {v_1 -> v_2 -> …-> v_n} là đường đi ngắn nhất, trong đó $v_i \in G$ G (*i = 1, 2, ..., n*), $v_1 = 1$ và $v_n = 16$.

Để trả lời câu truy vấn Q trên đồ thị phân tán, ý tưởng cơ bản của chúng tôi gồm các bước như sau:

Bước 1: Khi nhận được câu truy vấn Q(s, t), một máy đóng vai trò là máy chủ điều phối (master) sẽ gửi Q đến mỗi máy cục bộ (slaver)

Bước 2: Sau khi nhận được câu truy vấn Q từ máy *master* (*M*), mỗi máy slaver (S_j) sẽ thực hiện thuật toán Dijkstra tìm đường đi ngắn nhất từ tất các các *input node* đến *output node* trên phần đồ thị con G_i tương ứng một cách song song. Điều này có nghĩa là chúng ta tìm đường đi ngắn nhất từ các đỉnh biên ở đồ thị này đến các đỉnh biên thuộc đồ thị khác, những đỉnh có thể liên kết đồ thị con với nhau. Trên mỗi slaver S_i chúng ta nhận được các phần kết quả P_i tương ứng. Mỗi đỉnh trong các phần kết quả này có thể liên quan đến việc tìm ra kết quả cuối cùng cho câu truy vấn Q trên đồ thị G. Trong P_i, mỗi *input node* ***v*** có thể tồn tại đường đi tới một *output node* ***u***, nhưng ta không biết được ***u*** có thể đến được đỉnh đích ***t*** hay không. Như vậy, ta có thể biểu diễn P_i bằng một tập các vector đường đi với số phần tử là số *output node* của G_i, ở đó giá trị mỗi phần tử của vector được định nghĩa bằng đường đi ngắn nhất từ một input node đến một output node (bao gồm độ dài và đường đi). Bước này được minh họa bởi *Thuật toán 2*.

Bước 3: Tổng hợp các phần kết quả để xây dựng lên một *đồ thị phụ thuộc* trên máy master. Sau đó thực hiện thuật toán Dijkstra một lần nữa trên đồ thị phụ thuộc để tìm ra đường đi ngắn nhất từ đỉnh ***s*** đến đỉnh ***t***. Đây là kết quả cuối cùng cho câu truy vấn Q(s,t) trên đồ thị G.

Thuật toán 2: Thủ tục ***LocalEval***

Đầu vào: mảnh đồ thị F_i và câu truy vấn Q(s, t)
Đầu ra: Tập các vector đường đi P_i

```
1:  P_i ← ∅;
2:  if s ∈ V_i then
3:      V_i.in ← V_i.in ∪ {s};
4:  if t ∈ V_i then
5:      Vi.out ← V_i.out ∪ {t};
6:  for each node v ∈ V_i do
7:      v.visited = false;
8:  for each node v ∈ V_i.in do
9:      v.pvec ← Dijkstra(G_i, v, V_i.out);
10:     if v.pvec ≠ ∅ then
11:         P_i ← v.pvec;
12: return P_i;
```

Thuật toán 2 thực hiện tính toán một phần kết quả cho câu truy vấn Q trên một đồ thị con. Đầu tiên, (1) khởi tạo tạo P_i là một tập vector rỗng, ở đó ***v.pvec*** $\in P_i$ là một vector đường đi, mỗi phần tử ***v.pvec[u]*** là một đường đi từ input node v đến output node u. (2) Sau đó nó thực hiện việc kiểm tra đỉnh nguồn và đích để thêm vào tập V_i.in hay V_i.out tương ứng. Cuối cùng, (3) sử dụng thuật toán Dijkstra tính toán kết quả đường đi ngắn nhất ***v.pvec*** từ input node v đến các đỉnh output node u $\in V_i$.out như sau: nếu tồn tại sẽ ghi giá trị đường đi $X_{(v,u)}$ (khoảng cách ngắn nhất và danh sách các đỉnh đi qua theo thứ tự) vào vector ***v.pvec***; ngược lại, nếu không tồn tại đường đi, giá trị khoảng cách trả về là ∞ và đường đi bằng $\emptyset$ thì không thêm vào ***v.pvec***. Sau đó thêm ***v.pvec*** khác $\emptyset$ vào P_i.

Ví dụ 3:

Trong ví dụ này, chúng tôi thực hiện thuật toán 2, sử dụng câu truy vấn và các mô tả ở trong Ví dụ 2. Theo như Thuật toán 2, chúng ta phải thực hiện trả lời câu truy vấn Q(1, 16) trên 3 máy cục bộ và nhận về 3 phần kết quả lần lượt là P_1, P_2, P_3. Sau đó tổng hợp thành một đồ thị phụ thuộc và trả lời câu truy vấn Q trên đồ thị đó. Tuy nhiên, ở đây, chúng tôi chỉ trình bày áp dụng thuật toán đề xuất trên mảnh đồ thị F_1. Khởi tạo ta có $P_i = \emptyset$, V_i.in = {2} và V_i.out = {7, 11, 12}. Trên V_1, tồn tại đỉnh (1) là đỉnh nguồn của câu truy vấn nên đỉnh (1) sẽ được thêm vào Vi.in, khi đó V_1.in = {1, 2}; đỉnh (16) không tồn tại trên V_1, nên V_i.out giữ nguyên. Sau đó thực hiện Dijkstra tìm đường đi ngắn nhất từ các input node đến output node ta thu được các vector đường đi như trong Bảng 3.

Bảng 3. *Kết quả thực hiện câu truy vấn Q trên F_1*

v.pvec	(7)	(11)	(12)
(1).pvec	$X_{(1,7)}$ = [3, {1, 3, 7}]	$X_{(1,11)}$ = [5, {1, 4, 11}]	$X_{(1,12)}$ = [4, {1, 3, 5, 12}]
(2).pvec	$X_{(2,7)}$ = [8, {2, 3, 7}]	$X_{(2,11)}$ = [∞, {∅}]	$X_{(2,12)}$ = [8, {2, 3, 5, 12}]

Như vậy, P_1 gồm 2 vector với 5 giá trị đường đi từ các đỉnh input node đến các đỉnh output node. Kết quả này sẽ được gửi tới máy chủ điều phối để xây dựng lên đồ thị phụ thuộc. Hoàn toàn tương tự, ta có thể thực hiện tính được các kết quả P_2, P_3 trên các mảnh đồ thị F_1 và F_2 tương ứng như trong Bảng 4 và Bảng 5.

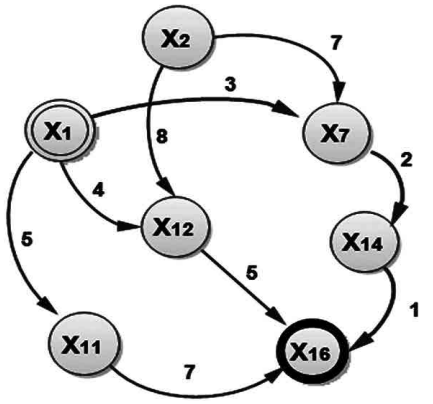
Bảng 4. *Kết quả thực hiện câu truy vấn Q trên F_2*

v.pvec	(2)	(14)
(7).pvec	$X_{(7,2)} = [\infty, \{\emptyset\}]$	$X_{(7,14)} = [2, \{7, 14\}]$

Bảng 5. *Kết quả thực hiện câu truy vấn Q trên F_3*

v.pvec	(16)
(11).pvec	$X_{(11,16)} = [7, \{11, 13, 15, 16\}]$
(12).pvec	$X_{(12,16)} = [5, \{12, 15, 16\}]$
(14).pvec	$X_{(14,16)} = [1, \{14, 16\}]$

Sau khi có các kết quả từ P_1, P_2, và P_3 từ các máy slaver gửi lên, tại máy master tiến hành xây dựng một đồ thị phụ thuộc $G_d = (V_d, E_d)$ như sau: với mỗi $X_{(v,u)}$ của P_i tạo ra hai đỉnh trên đồ thị phụ thuộc tương ứng X_v và X_u và một cạnh nối giữ chúng với trọng số là giá trị khoảng cách của $X_{(v,u)}$. Giá trị đường đi của $X_{(v,u)}$ được dùng cho việc truy vết đường đi khi cạnh nối X_v và X_u tồn tại trong kết quả đường đi ngắn ngất trả lời cho câu truy vấn Q trên đồ thị G_d cũng như đồ thị G.

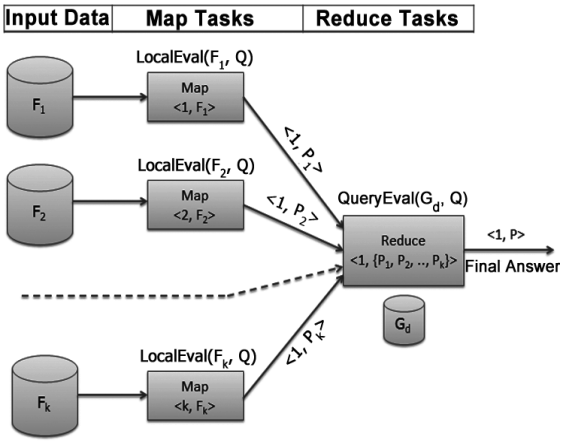


Hình 3. *Đồ thị phụ thuộc G_d được tạo ra từ kết quả của các P_i*

Bằng việc sử dụng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh X_1 đến X_{16} trên đồ thị G_d ta tìm ra được kết quả đường đi P' (màu đỏ): X_1 -> X_7 -> X_{14} -> X_{16} với độ dài = 6. Tiếp theo, nối các giá trị đường đi của $X_{(1,7)}$, $X_{(7,14)}$ và $X_{(14,16)}$ theo thứ tự ta tìm được câu trả lời P(Q): 1 -> 3 -> 7 -> 14 -> 16, có độ dài = 6 là kết quả cuối cùng cho câu truy vấn Q trên đồ thị G.

Cài đặt bài toán sử dụng MapReduce

Trong phần này, chúng tôi trình bày kỹ thuật sử dụng MapReduce để cài đặt các thuật toán đề xuất. Mô hình sử dụng MapReduce để trả lời câu truy vấn Q trên đồ thị phân tán G được thể hiện trong Hình 4.



Hình 4. *Mô hình MapReduce sử dụng để trả lời câu truy vấn trên đồ thị phân tán*

Trong mô hình này, chúng tôi sử dụng một Job để cài đặt thuật toán. Ở đây, số Map Tasks phụ thuộc và số phần đồ thị con của bài toán, và chỉ duy nhất một Reduce Task được sử dụng vào mục đích tổng hợp các phần kết quả và tìm ra câu trả lời cuối cùng cho câu truy vấn Q trên đồ thị G.

Dữ liệu các mảnh đồ thị được lưu trên hệ thống HDFS của Hadoop gồm *k* mảnh. Cách thức hoạt động của mô hình như sau: (1) khi một câu truy vấn Q(s,t) được đưa ra, máy chủ điều phối (master) sẽ chuyển nó xuống các slavers (giả sử có *k* máy slavers); (2) tại mỗi máy thực thi hàm Map bằng cách gọi thủ tục ***LocalEvalMapper***, thủ tục này có nhiệm vụ tìm ra các đường đi ngắn nhất từ các input node đến các output node bằng cách gọi hàm ***LocalEval(F_i, Q)*** đã trình bày và trả về kết quả là tập các vector đường đi chứa trong P_i. Kết quả của mỗi Map Task được ghi xuống hệ thống HDFS và gửi đến Reduce Task với cùng một khóa *K1*. Ở đây, chúng tôi chọn một giá trị khóa duy nhất nhằm mục đích tất cả các phần kết quả được tập hợp về một máy để tạo lên một đồ thị phụ thuộc G_d. (3) Sau đó, tại máy chủ điều phối sẽ thực hiện hàm Reduce bằng cách gọi thủ tục ***EvalReducer***, thủ tục này có nhiệm vụ tạo ra một đồ thị phụ thuộc G_d từ các phần kết quả gửi đến bởi các hàm Map và thực hiện hàm Dijkstra để tìm đường đi ngắn nhất từ đỉnh nguồn đến đỉnh đích và trả về kết quả cuối cùng cho câu truy vấn Q trên đồ thị G.

5. Thực nghiệm và đánh giá kết quả

Thiết lập môi trường thực nghiệm

Các cài đặt sử dụng cho thực nghiệm các thuật toán trong bài báo này như sau:

a) Môi trường thực nghiệm

Hệ thống Hadoop dùng để thực nghiệm trong bài báo bao gồm 17 máy tính để bàn: một máy được

dùng làm master node, 16 máy slaver node dùng cho việc tính toán tìm ra kết quả của bài toán. Mỗi máy tính có 2 CPUs và bộ nhớ RAM là 2 GB. Tất cả các thuật toán trong bài báo được cài đặt bằng Java.

b) Dữ liệu thực nghiệm

Bài báo sử dụng một dữ liệu mạng đường đi của thành phố New York [7] với kích thước 264.346 đỉnh và 733.846 cạnh, trong đó mỗi đỉnh là định danh một địa điểm của thành phố và mỗi cạnh chỉ đường đi từ một địa điểm đến một địa điểm khác. Mỗi cạnh có trọng số là khoảng cánh giữa hai địa điểm.

Để tạo ra các phần đồ thị con khác nhau mà vẫn đảm bảo tính đúng đắn của dữ liệu, bài báo đã sử dụng công cụ GraphLab [8] cho việc phân mảnh đồ thị thành nhiều đồ thị con. Chúng tôi đã chuẩn bị 4 bộ dữ liệu với số phần đồ thị con lần lượt là 4, 8, 16 và 32.

c) Câu truy vấn

Chúng tôi đã chọn ra ngẫu nhiên một câu truy vấn: Tìm đường đi ngắn nhất từ đỉnh có mã là "1" đến đỉnh có mã là "14" trên đồ thị. Câu truy vấn này tồn tại kết quả một đường đi ngắn nhất với tổng độ dài = 111751, nó được đưa ra từ kết quả chạy thuật toán Dijkstra trên đồ thị ban đầu (chưa phân mảnh). Chúng tôi dùng kết quả này để đối chiếu với kết quả sinh ra từ thuật toán đề xuất.

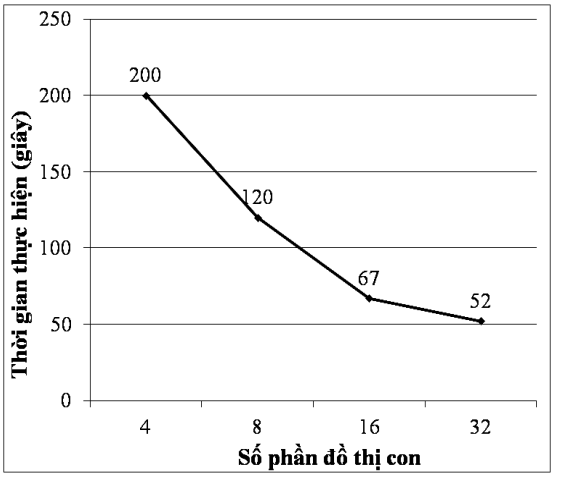
d) Kết quả thực nghiệm

Trong phần này, chúng tôi trình bày kết quả của việc thực hiện câu truy vấn nêu trên và đo lượng thời gian trả lời câu truy vấn trên môi trường phân tán với các bộ dữ liệu có kích thước khác nhau.

Hình 5 cho thấy kết quả khi chia đồ thị ra thành càng nhiều phần đồ thị con và thực hiện thuật toán song song trên từng đồ thị con đó thì thời gian thực hiện thuật toán sẽ giảm dần. Cụ thể khi dữ liệu được chia thành 4 phần đồ thị con thì thời gian thực hiện của thuật toán là 200s, khi chia thành 8 phần đồ thị con thì thời gian thực hiện giảm còn 120s và khi số đồ thị con là 32 thì thời gian giảm chỉ còn 52s.

e) Phân tích

Hình 5 chỉ ra rằng, thời gian trả lời câu truy vấn sẽ tỉ lệ nghịch với số phần đồ thị con. Thời gian giảm nhanh khi tăng số lượng hàm Map dùng cho việc tìm kiếm các kết quả từng phần. Ở đây, mỗi phần đồ thị con được thực hiện bởi một hàm Map. Thời gian thực hiện ở đây bao gồm cả thời gian khởi tạo 1 Job trong Hadoop, ghi/đọc dữ liệu từ HDFS và thời gian chạy thuật toán trả lời câu truy vấn. Chính vì vậy, ở đây chúng tôi không so sánh về mặt thời gian thực hiện giữa thuật toán song song phân tán với thời gian thuật toán chạy tuần tự trên một máy. Đề xuất này của chúng tôi có ý nghĩa trong việc xử lý phân tán với dữ liệu lớn, cái mà một máy tính đơn nhất có thể không giải quyết được cả về mặt lưu trữ và việc xử lý tính toán.



Hình 5. *Kết quả thời gian thực thi câu truy vấn với số lượng đồ thị con khác nhau*

6. Kết luận

Bài báo đã trình bày một cách tiếp cận mới để giải quyết bài toán tìm đường đi ngắn nhất từ một đỉnh đến một đỉnh khác trên đồ thị phân tán. Chúng tôi đã đưa ra một thuật toán dựa trên kỹ thuật ước lượng từng phần và khai thác nền tảng hỗ trợ xử lý dữ liệu song song phân tán MapReduce. Kết quả thực nghiệm trên một dữ liệu đồ thị thực tế đã chỉ ra tính hiệu quả trong cách tiếp cận của chúng tôi cho việc xử lý đồ thị lớn trên môi trường phân tán. Do đó, đề xuất này có thể áp dụng cho các ứng dụng khác trong thực tế như phân tích mạng xã hội, mạng giao thông,..vv. Tiếp theo hướng nghiên cứu này, chúng tôi sẽ tập trung vào các loại bài toán khác trên môi trường phân tán ví dụ như bài toán truy vấn theo biểu thức chính quy có điều kiện.

Tài liệu tham khảo

[1]. Lê Minh Hoàng, "*Chuyên đề Lý thuyết đồ thị*", Đại Học Sư Phạm Hà Nội, 1999-2002

[2]. Dean, Jeffrey, and Sanjay Ghemawat, "*MapReduce: Simplified Data Processing on Large Clusters*", Communications of the ACM 51.1 (2008): 107-113.

[3] N. D. Jones, *An Introduction to Partial Evaluation. ACM Computing Surveys (CSUR)*, 28(3):480–503, 1996.

[4]. P. Buneman, G. Cong, W. Fan, and A. Kementsietsidis, *Using Partial Evaluation in Distributed Query Evaluation*, In Proceedings of the 32nd International Conference on Very Large Databases,

pages 211–222. VLDB Endowment, 2006.
[5]. W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu, *Adding Regular Expressions to Graph Reachability and Pattern Queries*, In Data Engineering (ICDE), 2011 IEEE 27th International Conference on, pages 39–50. IEEE, 2011.
[6]. W. Fan, X. Wang, and Y. Wu, *Performance Guarantees for Distributed Reachability Queries*, Proceedings of the VLDB Endowment, 5(11):1304–1316, 2012.
[7]. Camil Demetrescu. 2014, "*9th DIMACS Implementation Challenge: Shortest Paths*", Accessed January 08, 2016. http://www.dis.uniroma1.it/challenge9/download.shtml
[8]. Low, Yucheng, et al, "*Graphlab: A new Framework for Parallel Machine Learning*", arXiv preprint arXiv:1408.2041 (2014).

A NOVEL APPROACH TO SHORTEST PATH PROBLEM ON DISTRIBUTED GRAPH

Abstract:

Recently, the efficient of query evaluation on big graphs is an important research topic in computer science. A widely-used query is the shortest path query between two nodes in a graph, which can be evaluated on a general graph by using a few well-known algorithms such as Dijkstra, Johnson or Floyd-Warshall; however, it is nontrivial to answer this kind of query in a distributed graph. In this paper, we propose a novel approach based on the partial evaluation to solve shortest path problem between two nodes on distributed graph. We show that our algorithm can be readily implemented in the MapReduce framework, in parallel. Using a real-life data we perform experiments and show that our algorithm is scalable on large graphs on the distributed systems.

Keywords: *Graph Querying, MapReduce, Shorstest Path, Distributed Graph, Partial Evaluation.*