Kỷ yếu Hội nghị Quốc gia lần thứ X về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR), Đà Nẵng, ngày 17-18/08/2017
DOI: 10.15625/vap.2017.000116

XÂY DỰNG MODULE XỬ LÝ TRUY VẤN PHÂN TÁN TRÊN HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU ĐỒ THỊ NEO4J

Nguyễn Duy Tân, Ngô Thanh Hùng

Trường Đại học Công nghệ Thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh

duytan.khmt@gmail.com, hungnt@uit.edu.vn

TÓM TẮT: *Bài báo trình bày kiến trúc phân tán và module xử lý truy vấn phân tán trên hệ quản trị cơ sở dữ liệu đồ thị Neo4j. Kiến trúc được áp dụng là phân mảnh dọc cơ sở dữ liệu đồ thị Neo4j, các phân mảnh được đặt trên các máy tính kết nối với nhau thông qua giao thức HTTP. Module được thiết kế nhằm xử lý một số mẫu truy vấn theo ngôn ngữ Cypher dựa trên cơ sở đại số quan hệ. Module có thể được ứng dụng cho một số bài toán thực tế và có thể được phát triển thêm để có thể tối ưu hơn và xử lý được nhiều mẫu câu truy vấn hơn.*

Từ khóa: *Neo4j, Cypher, phân mảnh dọc cơ sở dữ liệu đồ thị, xử lý truy vấn phân tán cơ sở dữ liệu đồ thị.*

I. GIỚI THIỆU

Neo4j là một hệ thống quản lý cơ sở dữ liệu đồ thị được phát triển bởi Neo Technology Inc và là cơ sở dữ liệu đồ thị phổ biến nhất hiện nay theo DB-Engines (*db-engines.com)*. Nó là một cơ sở dữ liệu giao dịch đầy đủ (ACID) chứa các dữ liệu được cấu trúc dưới dạng đồ thị bao gồm các nút và các mối quan hệ giữa chúng. Trong Neo4j, mọi thứ được lưu trữ ở dạng của một nút, một cạnh (quan hệ), hoặc một thuộc tính. Mỗi nút và cạnh có thể có một số lượng không giới hạn các thuộc tính. Cả nút và cạnh đều có thể được gắn nhãn. Nhãn có thể được sử dụng để thu hẹp việc tìm kiếm.

Cypher là một trong những ngôn ngữ truy vấn cơ sở dữ liệu đồ thị phổ biến của Neo4j cho phép truy vấn và cập nhật đồ thị một cách hiệu quả. Cypher là một ngôn ngữ tương đối đơn giản nhưng rất mạnh mẽ. Cú pháp của Cypher cung cấp một cách trực quan để so khớp các mẫu cần tìm dựa vào thông tin về các nút và các mối quan hệ. Các truy vấn cơ sở dữ liệu phức tạp có thể dễ dàng được thể hiện thông qua Cypher.

Đối với cơ sở dữ liệu đồ thị chưa phân tán thì tất cả các quan hệ nằm trên cùng một máy chủ cùng với thông tin về các nút làm ảnh hưởng đến việc quản lý tính riêng tư và bảo mật dữ liệu. Việc phân mảnh cơ sở dữ liệu đồ thị, cũng giống như việc phân mảnh cơ sở dữ liệu quan hệ, mang lại nhiều lợi ích như: - giúp phân quyền và bảo mật dễ hơn nhờ có thể áp dụng chính sách bảo mật, phân quyền đối với từng mảnh một cách riêng biệt; - giúp nâng cao tính sẵn sàng của hệ thống nhờ tránh được trường hợp "đơn điểm gây lỗi" (single point of failure); - và trong một số trường hợp còn có thể giảm thời gian truy vấn nhờ giảm dung lượng của từng phân mảnh hay nói cách khác là nhờ tăng tỉ lệ truy xuất thành công từ bộ nhớ; ... Do các phiên bản miễn phí hiện tại của Neo4j chưa hỗ trợ phân tán nên việc nghiên cứu phát triển hệ thống phân tán cơ sở dữ liệu đồ thị Neo4j là cần thiết. Hiện nay có nhiều công trình liên quan đến việc phân tán cơ sở dữ liệu đồ thị Neo4j. Trong bài báo [2] nhóm tác giả đã xây dựng một thành phần mở rộng của cơ sở dữ liệu đồ thị Neo4j, cho phép di chuyển các dữ liệu từ xa phục vụ cho các câu truy vấn nhằm tối ưu hóa thời gian truy vấn. Bên cạnh đó, bài báo còn giới thiệu thuật toán phân vùng động sử dụng ít tài nguyên và đồng thời tăng hiệu suất lên so với cách phân vùng ngẫu nhiên. Bài báo [3] xem xét các phương pháp phân vùng cơ sở dữ liệu đồ thị theo cạnh, đỉnh và thuộc tính. Phương pháp phân vùng theo đỉnh đã được áp dụng nhưng việc xử lý truy vấn dữ liệu giữa các máy chủ chưa được trình bày. Trong [4, 5, 6] đề cập đến một số thuật toán phân tán đồ thị và áp dụng cụ thể để phân chia dữ liệu mạng xã hội thực tế như: Facebook, Twitter,…. Như đã biết thì đối với cơ sở dữ liệu quan hệ có ba phương pháp phân mảnh cơ bản là phương pháp phân mảnh ngang, phương pháp phân mảnh dọc và phương pháp lai. Trong khi đó các công trình mà chúng tôi đã tổng quan ở trên đều tập trung vào việc phân tán đồ thị theo các đỉnh - đây chính là phương pháp phân mảnh ngang cơ sở dữ liệu đồ thị. Nhận thấy phương pháp phân mảnh dọc cơ sở dữ liệu đồ thị cũng sẽ mang lại những lợi ích tương tự như những lợi ích mà nó đã mang lại cho cơ sở dữ liệu quan hệ nên trong nghiên cứu này chúng tôi trình bày một hình thức phân mảnh dọc cơ sở dữ liệu đồ thị - phân mảnh các quan hệ. Chúng tôi sẽ trình bày kiến trúc phân mảnh dọc và module xử lý truy vấn phân tán trên hệ quản trị cơ sở dữ liệu đồ thị Neo4j và ngôn ngữ Cypher.

Cấu trúc bài báo gồm 6 phần, phần (I) - giới thiệu, phần (II) sẽ trình bày kiến trúc phân mảnh dọc cơ sở dữ liệu đồ thị, phần (III) sẽ phân tích module xử lý truy vấn phân tán, phần (IV) là phần cài đặt và thực nghiệm, phần (V) là phần kết luận và phần cuối (VI) là tài liệu tham khảo.

II. KIẾN TRÚC PHÂN TÁN

A. Đề xuất kiến trúc

Trong bài báo này, kiến trúc phân mảnh dọc được thực hiện trên cơ sở phân tán các quan hệ của cơ sở dữ liệu đồ thị. Mỗi phân mảnh sẽ bao gồm một số quan hệ của cơ sở dữ liệu đồ thị ban đầu cùng với toàn bộ thông tin về các nút có liên quan đến một hoặc nhiều trong số các quan hệ này. Các quan hệ không được trùng lắp ở các phân mảnh. Mỗi phân mảnh có thể được đặt ở các thư mục khác nhau hoặc trên các máy tính khác nhau (các nút mạng) được liên kết với nhau qua giao thức HTTP, giống như giao thức được sử dụng bởi API của Neo4j.

Trên tất cả các nút mạng sẽ chứa module xử lý truy vấn phân tán. Metadata mô tả kiến trúc phân mảnh sẽ được lưu trữ trên tất cả các nút mạng bao gồm các thông tin sau:

- Thông tin các phân mảnh gồm vị trí các phân mảnh và các quan hệ đi kèm.

  PARTITION = Vị trí phân mảnh thứ nhất # [Quan hệ]{Thuộc tính quan hệ};
  Vị trí phân mảnh thứ hai # [Quan hệ]{Thuộc tính quan hệ};…;

Thông tin các phân mảnh được tách biệt bởi dấu chấm phẩy (;). Vị trí mỗi phân mảnh, được mô tả bởi IP của nút mạng và đường dẫn đến thư mục chứa cơ sở dữ liệu Neo4j, và các quan hệ cách nhau bởi dấu thăng (#). Mỗi quan hệ được đặt trong ngoặc vuông ([]), thuộc tính của quan hệ được đặt trong ngoặc nhọn ({}). Trường hợp phân mảnh có nhiều quan hệ thì các quan hệ này được tách biệt bởi dấu gạch nối (-). Trong số đó, thông tin về phân mảnh trên chính nút mạng này (phân mảnh trên local) được mô tả bằng đường dẫn đến thư mục chứa cơ sở dữ liệu Neo4j.

- Thông tin các loại nút cùng các thuộc tính

  NODE = (Loại nút 1){Thuộc tính của loại nút 1}; (Loại nút 2){Thuộc tính của loại nút 2};

  (Loại nút 3){Thuộc tính của loại nút 3}; …;

Các loại nút được tách biệt bởi dấu chấm phẩy (;). Tên loại nút được đặt trong dấu ngoặc đơn (), thuộc tính của nút được đặt trong dấu ngoặc nhọn ({}).

- Thông tin các quan hệ dùng mô tả quan hệ và các loại nút đi kèm

  RELATIONSHIP = (Loại nút 1)-[:Quan hệ 1]->(Loại nút 2);

  (Loại nút 3)-[:Quan hệ 2]->(Loại nút 3); …;

Mỗi quan hệ được gắn với một hoặc hai loại nút cố định. Các quan hệ được tách biệt bởi dấu chấm phẩy (;).

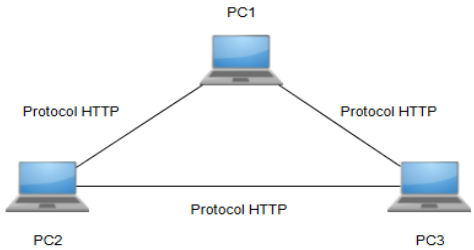
Kiến trúc phân mảnh dọc được đề xuất ở đây có thể được mở rộng bằng việc cho phép tập các quan hệ trên các phân mảnh khác nhau có thể có phần giao để tăng tính sẵn sàng của hệ thống cũng như tăng khả năng tính toán song song (giúp giảm thời gian xử lý truy vấn). Tuy nhiên trong bài báo này chúng tôi chưa tập trung vào vấn đề tối ưu truy vấn phân tán này. Mô hình và module xử lý được đề xuất ở dạng cơ bản nhất. Bên cạnh đó vấn đề bảo mật cũng chưa được nghiên cứu để đưa vào thiết kế này.

B. Ví dụ minh họa

Trong bài báo này, các câu truy vấn trong các ví dụ cũng như trong phần kiểm chứng được mô tả cho trường hợp phân mảnh cơ sở dữ liệu đồ thị trên Neo4j sau đây. Cơ sở dữ liệu đồ thị Movie ban đầu gồm:

- Thông tin nút: gồm hai loại nút
  - Person: gồm các thuộc tính như: tên, ngày sinh, nơi sinh,…
  - Movie: gồm các thuộc tính như: tựa đề, nội dung, năm phát hành, ngôn ngữ,….
- Thông tin quan hệ: gồm ba quan hệ
  - ACTS_IN: liên kết nút Person và Movie. Dùng để chỉ định diễn viên trong một bộ phim nào đó. Đây là quan hệ có hướng với đỉnh đầu là nút Person và đỉnh cuối là nút Movie.
  - DIRECTED: liên kết nút Person và Movie. Dùng để chỉ định đạo diễn của một bộ phim nào đó. Đây là quan hệ có hướng với đỉnh đầu là nút Person và đỉnh cuối là nút Movie.
  - FRIEND: liên kết nút Person và Person. Dùng để chỉ định mối quan hệ bạn bè giữa hai người. Đây là quan hệ có hướng với đỉnh đầu là nút Person và đỉnh cuối là nút Person.

Cơ sở dữ liệu này sẽ được phân thành 3 phân mảnh dựa trên 3 quan hệ ACTS_IN, DIRECTED và FRIEND. Phân mảnh 1 chứa toàn bộ quan hệ ACTS_IN và toàn bộ nút Person, nút Movie. Phân mảnh 2 chứa toàn bộ quan hệ DIRECTED và toàn bộ nút Person, nút Movie. Phân mảnh 3 chứa toàn bộ quan hệ FRIEND và toàn bộ nút Person (phân mảnh này không chứa nút dữ liệu của Movie vì quan hệ FRIEND không liên quan gì đến nút Movie).



Hình 1. Ví dụ các nút mạng chứa các phân mảnh dọc cơ sở dữ liệu đồ thị Movie

Mỗi phân mảnh sẽ được đặt lên một nút mạng như trong hình 1, trong đó:

Các thông tin metadata giống nhau giữa các nút mạng bao gồm các thông tin:

NODE = (Movie){id,studio,releaseDate, description, language, title, trailer, imageUrl, genre};
(Person){id, birthday, birthplace, name, biography};

RELATIONSHIP = (Person)-[:ACTS_IN]->(Movie); (Person)-[:DIRECTED]->(Movie);
(Person)-[:FRIEND]->(Person);

Và các metadata khác nhau về thông tin các phân mảnh như sau:

- Nút mạng PC1 chứa phân mảnh 1 với thông tin metadata như sau:

  PARTITION = D:\Neo4j\\Movie1\graph.db#[ACTS_IN]{};
  \\192.168.1.11\Neo4j\Movie2\graph.db#[DIRECTED]{};
  \\192.168.1.12\Neo4j\Movie3\graph.db#[FRIEND]{};

- Nút mạng PC2 chứa phân mảnh 2 với thông tin metadata như sau:

  PARTITION =
  \\192.168.1.10\Neo4j\\Movie1\graph.db#[ACTS_IN]{};D:\Neo4j\Movie2\graph.db#[DIRECTED]{};
  \\192.168.1.12\Neo4j\Movie3\graph.db#[FRIEND]{};

- Và nút mạng PC3 chứa phân mảnh 3 với thông tin metadata như sau:

  PARTITION = \\192.168.1.10\Neo4j\\Movie1\graph.db#[ACTS_IN]{};
  \\192.168.1.11\Neo4j\Movie2\graph.db#[DIRECTED]{};
  D:\Neo4j\Movie3\graph.db#[FRIEND]{};

III. MODULE XỬ LÝ TRUY VẤN PHÂN TÁN

A. Ý tưởng thuật toán

Module xử lý truy vấn phân tán có nhiệm vụ tiếp nhận yêu cầu truy vấn, được viết dưới ngôn ngữ truy vấn Cypher, từ nút mạng nội bộ, thực hiện phân tích truy vấn và thực hiện xử lý để trả lời truy vấn đó giống như khi chưa có sự phân mảnh. Để giải quyết việc này chúng tôi đề xuất một thuật toán gồm 3 bước cơ bản:

B1: Phân tích câu truy vấn gốc thành một tập các truy vấn con (cũng được viết dưới ngôn ngữ Cypher), trong đó mỗi truy vấn con chỉ liên quan đến đúng một quan hệ và mỗi truy vấn con bao gồm tất cả các điều kiện liên quan đến quan hệ đó.

Gọi T(r1, r2, …, rm) là truy vấn gốc liên quan đến m quan hệ khác nhau từng đôi một r1, r2, …, rm, với các ri thuộc tập tất cả các quan hệ của cơ sở dữ liệu đồ thị đang xét. Truy vấn T sẽ được tách thành n truy vấn con: T1(r1), T2(r2), …, Tm(rm) mỗi truy vấn con lần lượt liên quan đến đúng 1 quan hệ r1, r2, …, rm tương ứng.

B2: Gửi các truy vấn con đến từng các nút mạng chứa các quan hệ tương ứng, gọi thực thi thông qua GraphDatabaseService API của Neo4j trên từng nút mạng đó và lấy kết quả về. Mỗi truy vấn con Ti(ri) sẽ trả về một kết quả O(Ti).

B3: Tổng hợp các kết quả O(Ti) và tạo ra kết quả O(T) cho truy vấn gốc T.

Bước 1 và bước 3 là các giai đoạn xử lý phức tạp, phụ thuộc rất lớn vào dạng câu truy vấn gốc. Trong khuôn khổ bài báo này chúng tôi đã nghiên cứu và giải quyết được 3 dạng mẫu câu truy vấn khá phổ biến của Cypher dựa trên cơ

sở của phép các toán đại số quan hệ như hợp, giao, kết, … Cách giải quyết các dạng câu truy vấn này chúng tôi sẽ trình bày chi tiết trong phần 3.3 tiếp theo.

Việc hiện thực thuật toán trong khuôn khổ bài báo này còn một số hạn chế sau:

- Ở bước 1, chúng tôi chưa xem xét đến việc tối ưu xử lý truy vấn bằng cách cân nhắc việc gộp một số quan hệ, mà tất cả các quan hệ này được chứa trong cùng một phân mảnh nào đó, vào một truy vấn con. Việc này giúp giảm số truy vấn con cần xử lý trong bước 2, đồng thời cũng giúp giảm khối lượng xử lý để tạo kết quả cuối cùng O trong bước 3. Nhờ đó có thể giảm đáng kể thời gian xử lý truy vấn.
- Ở bước 2, chúng tôi chỉ chọn máy đầu tiên chứa quan hệ cần tìm chứ chưa tính đến việc lựa chọn máy nào trong số các máy chứa cùng một quan hệ (nếu có sự trùng lắp) để có thể thực hiện song song nhằm giảm thời gian xử lý truy vấn.

Những hạn chế này chúng tôi sẽ hướng đến trong các nghiên cứu tiếp theo.

B. Mã giả chi tiết của thuật toán

Đầu vào: truy vấn từ người dùng.

Đầu ra: kết quả thực thi câu truy vấn.

Bước 1: If Truy vấn hợp lệ

Đi tiếp Bước 2

Else

Thông báo lỗi

Bước 2: Danh sách mệnh đề = Phân rã truy vấn (Truy vấn)

Danh sách quan hệ = Lấy quan hệ (Truy vấn)

Bước 3: If Danh sách quan hệ > 1

Danh sách truy vấn mới = Viết lại truy vấn (Truy vấn,Danh sách mệnh đề, Danh sách quan hệ)

Else

Thêm Truy vấn vào Danh sách truy vấn mới

Bước 4: If Danh sách truy vấn mới > 1

Danh sách kết quả = Thực thi song song (Danh sách truy vấn mới)

Else

Kết quả = Thực thi truy vấn (Danh sách truy vấn mới)

Bước 5: If Danh sách truy vấn mới > 1

Kết quả = Tổng hợp dữ liệu (Danh sách kết quả)

Bước 6: Xuất kết quả

C. Thuật toán viết lại truy vấn và tổng hợp kết quả trả về

Trong ngôn ngữ truy vấn Cypher, dữ liệu các nút được đặt trong cặp dấu ngoặc đơn (n). Dữ liệu quan hệ được đặt trong cặp dấu ngoặc vuông, trước tên quan hệ có dấu hai chấm [:r]. Cặp dấu gạch ngang và lớn hơn/ nhỏ hơn để liên kết nút và quan hệ (n1) - [:r] -> (n2) hoặc (n1) <- [:r] - (n2). Mỗi quan hệ luôn được gắn với một hoặc hai loại nút cố định. Đỉnh đầu và đỉnh cuối của quan hệ được xác định rõ ràng. Do đó các quan hệ trong cơ sở dữ liệu đồ thị Neo4j luôn có hướng. Việc viết lại truy vấn và tổng hợp kết quả được thực hiện dựa vào các mẫu câu truy vấn. Dưới đây là 3 mẫu câu truy vấn được nghiên cứu xử lý trong khuôn khổ của nghiên cứu này.

1. **Mẫu truy vấn đơn:**
   - Dạng mẫu **M1**:

     [MATCH]

     [WITH]

     [WHERE]

     [RETURN] [ORDER BY] [SKIP] [LIMIT]

- Mô tả: các mệnh đề bắt buộc [MATCH], và [RETURN]. Các mệnh đề tùy biến (có thể có hoặc không) là [WITH], [WHERE], [ORDER BY], [SKIP] và [LIMIT].

Dạng câu truy vấn này chỉ chứa 1 mệnh đề [MATCH], và là sự tổ hợp của một trong 3 dạng con **M1.1, M1.2, M1.3** dưới đây.

- Dạng con thứ nhất **M1.1**: được áp dụng cho câu truy vấn có quan hệ dạng

$T(r_1, r_2, \ldots, r_m)$: MATCH (n_1) - [:r_1|:r_2|:r_3|…|:r_m] -> (n_2) RETURN n_1, n_2

Ý nghĩa của truy vấn này là tìm tất cả các cặp nút (n_1, n_2) có một trong các quan hệ r_1, r_2, …, r_m, trong đó các quan hệ này khác nhau từng đôi một.

Truy vấn dạng này được viết lại dưới đúng m truy vấn con như sau:

$T_1(r_1)$: MATCH (n_1) - [:r_1] -> (n_2) RETURN n_1, n_2

$T_2(r_2)$: MATCH (n_1) - [:r_2] -> (n_2) RETURN n_1, n_2

……

$T_m(r_m)$: MATCH (n_1) - [:r_m] -> (n_2) RETURN n_1, n_2

Các truy vấn này được thực thi và kết quả cuối cùng là hợp các kết quả của các truy vấn trên.

$O(T) = O(T_1) \cup O(T_2) \cup \ldots \cup O(T_m)$

Ví dụ:

```
MATCH (n) - [r:ACTS_IN | :DIRECTED] -> (m)
RETURN n.name, type(r) as relationship, m.title, m.language;
```

Câu truy vấn tìm tất cả những người đã tham gia diễn xuất hoặc làm đạo diễn cho một bộ phim nào đó.

Truy vấn được viết lại thành hai truy vấn. Một truy vấn tìm tất cả những người đã tham gia diễn xuất trong một bộ phim nào đó

```
T1(ACTS_IN): MATCH (n)-[r:ACTS_IN]->(m)
                RETURN n.name, type(r) as relationship, m.title, m.language
```

Và truy vấn thứ hai tìm tất cả những người đã tham gia đạo diễn cho một bộ phim nào đó

```
T2(DIRECTED): MATCH (n)-[r:DIRECTED]->(m)
                RETURN n.name, type(r) as relationship, m.title, m.language
```

Kết quả cuối cùng là hợp kết quả của hai câu truy vấn trên.

- Dạng con thứ hai **M1.2**: được áp dụng cho các câu truy vấn trong đó một nút hoặc nhiều nút có quan hệ cùng lúc với nhiều nút khác nhau. Đây là dạng rất phổ biến.

$T(r_1, r_2, \ldots, r_k)$: MATCH (n_1) - [:r_1] - >(n_2) ,(n_2) – [:r_2] -> (n_3)
, … , (n_i) – [:r_i] -> (n_{i+1})
, … , (n_m) – [:r_m] -> (n_{m+1})
RETURN n_1, n_2, …, n_{m+1}

Ý nghĩa của câu truy vấn là tìm tất cả các bộ m+1 nút (n_1, n_2, …, n_{m+1}) sao cho giữa từng cặp nút n_i và n_{i+1} có quan hệ r_i với nhau, trong đó các quan hệ r_1, r_2, …, r_m khác nhau từng đôi một.

Truy vấn dạng M1.2 được viết lại thành đúng m câu truy vấn đơn, mỗi câu truy vấn chỉ chứa một quan hệ như sau.

$T_1(r_1)$: MATCH (n_1) – [:r_1] -> (n_2) RETURN n_1, n_2, ID(n_2) AS n2ID

$T_2(r_2)$: MATCH (n_2) – [:r_2] -> (n_3) RETURN n_2, n_3, ID(n_2) AS n2ID

……

$T_m(r_m)$: MATCH (n_m) – [:r_m] -> (n_{m+1}) RETURN n_m, n_{m+1}, ID(n_i) AS niID

Kết quả cuối cùng của câu truy vấn là lần lượt kết các kết quả của các câu truy vấn lại với nhau dựa trên khóa chung của chúng:

$$O(T) = O(T_1) *^{n2} O(T_2) *^{n3} O(T_3) * \ldots *^{nm} O(T_m)$$

Ví dụ: Câu truy vấn tìm tất cả diễn viên và đạo diễn trong cùng một bộ phim

```
MATCH (a)-[:ACTS_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name AS actor, m.title, d.name AS director
```

Được viết lại thành hai truy vấn, truy vấn thứ nhất tìm tất cả những người là diễn viên:

```
MATCH (a)-[:ACTS_IN]->(m)
RETURN distinct a.name as actor, m.title, ID(m) AS mID
```

và truy vấn thứ hai tìm tất cả những người là đạo diễn

```
MATCH (m)<-[:DIRECTED]-(d)
RETURN m.title, d.name as director, ID(m) AS mID
```

Kết quả của truy vấn ban đầu là kết quả của hai câu truy vấn trên được kết lại dựa trên trường mID.

- Dạng con thứ ba **M1.3**: được dùng để xác định một cặp nút thuộc quan hệ này nhưng không thuộc quan hệ khác và có thể thỏa một số điều kiện cụ thể.

$T(r_1, r_2)$: MATCH <mệnh đề điều kiện MATCH>
WHERE NOT <mệnh đề điều kiện WHERE NOT>
AND <các điều kiện WHERE>
RETURN <tập các nút RETURN>

Trong đó <mệnh đề điều kiện MATCH> có thể có dạng của mệnh đề [MATCH] trong dạng mẫu truy vấn M1.1 hoặc M1.2, <mệnh đề điều kiện WHERE NOT> có dạng của mệnh đề [MATCH] trong dạng mẫu truy vấn M1.1, <các điều kiện WHERE> dùng để so sánh các nút hoặc các thuộc tính với các giá trị cụ thể, và tập hợp tất cả các loại nút trong <mệnh đề điều kiện WHERE NOT>, <các điều kiện WHERE> và <tập các nút RETURN> phải là tập con thật sự hoặc bằng với tập tất cả các loại nút trong <mệnh đề điều kiện MATCH>.

Ý nghĩa của dạng truy vấn này là tìm tất cả các bộ nút (tương ứng với tập các nút trong mệnh đề [RETURN]) thỏa mệnh đề điều kiện [MATCH] và các điều kiện WHERE nhưng không thỏa mệnh đề điều kiện WHERE NOT.

Truy vấn trên được viết lại dưới 2 truy vấn con:

$T_1(r1)$: MATCH <mệnh đề điều kiện MATCH>
WHERE <các điểu kiện WHERE>
RETURN <tập các nút RETURN>

$T_2(r2)$: MATCH <mệnh đề điều kiện WHERE NOT>
RETURN <tập các nút RETURN>

Kết quả cuối cùng là hiệu giữa hai tập kết quả của hai câu truy vấn trên:

$$O(T) = O(T_1) \backslash O(T_2)$$

Ví dụ:

Câu truy vấn tìm tất cả những người chỉ đóng vai trò là diễn viên trong một bộ phim và sinh ra ở New York

```
MATCH (n) - [r:ACTS_IN] -> (m)
WHERE NOT((n) - [:DIRECTED] -> (m))  AND n.birthplace = 'New York'
RETURN n.name, n.birthday, n.birthplace, m.title, m.studio;
```

sẽ được viết lại thành hai truy vấn, truy vấn thứ nhất tìm tất cả những người đóng vai trò diễn viên và được sinh ra ở New York

```
MATCH (n)-[r:ACTS_IN]->(m)
WHERE n.brirthplace = 'New York'
RETURN n.name, n.birthday, n.birthplace, m.title, m.studio, ID(n) AS nID, ID(m) AS mID
```

và truy vấn thứ hai tìm tất cả những người đóng vai trò đạo diễn

```
MATCH (n)-[:DIRECTED]->(m)
RETURN ID(n) AS nID, ID(m) AS mID
```

Kết quả sau cùng là hiệu của truy vấn thứ nhất với truy vấn thứ hai.

2. **Mẫu truy vấn lồng không có mệnh đề OPTIONAL:**

- Dạng mẫu **M2**:

  [MATCH] … [WITH] …

  [MATCH] … [WITH] …

  …

  [WHERE]

  [RETURN] [ORDER BY] [SKIP] [LIMIT]

- Mô tả: mẫu truy vấn có nhiều cặp mệnh đề [MATCH] [WITH] và một mệnh đề [RETURN]. Các mệnh đề còn lại như [WHERE], [ORDER BY], [SKIP] và [LIMIT] thì có thể tùy biến (sử dụng hoặc không sử dụng).
- Ý nghĩa của mẫu truy vấn: đây là dạng mẫu truy vấn lồng, trong đó các kết quả tìm được ở các mệnh đề [MATCH] ở tầng trên sẽ được xem xét thêm các điều kiện truy vấn bổ sung bởi mệnh đề [MATCH] ở tầng liền dưới nó và cứ tiếp tục như vậy cho đến mệnh đề [MATCH] ở tầng cuối cùng.
- Truy vấn được viết lại bằng cách biến đổi từng cặp mệnh đề [MATCH] [WITH] với các tùy biến khác thành một truy vấn con dưới dạng một cặp mệnh đề [MATCH] và [RETURN] với các tùy biến tương ứng. Mỗi câu truy vấn con này sẽ thuộc một trong 3 dạng **M1.1, M1.2** hoặc **M1.3** ở trên.
- Kết quả của truy vấn ban đầu là kết quả trả về của các truy vấn con lần lượt được kết với nhau dựa trên các trường dữ liệu chung.
- Ví dụ: Truy vấn ban đầu, tìm tất cả những người đã tham gia diễn xuất ít nhất 10 bộ phim và đã làm đạo diễn ít nhất 2 bộ phim

```
MATCH (a:Actor)-[:ACTS_IN]->(m:Movie) WITH a, count(m) AS ACTS
WHERE ACTS >= 10
WITH a, ACTS
MATCH (a:Director)-[:DIRECTED]->(m:Movie) WITH a, ACTS, collect(m.title) AS directed
WHERE length(directed) >= 2
RETURN a.name, ACTS, directed ORDER BY length(directed) DESC, ACTS DESC;
```

sẽ được viết lại thành hai truy vấn, truy vấn thứ nhất tìm tất cả những người đã làm diễn viên ít nhất 10 bộ phim

```
MATCH (a:Actor)-[:ACTS_IN]->(m:Movie)  WITH a, count(m) AS ACTS
WHERE ACTS >= 10
RETURN a, ACTS, ID(a) AS aID
```

Và truy vấn thứ hai tìm tất cả những người đã làm đạo diễn ít nhất 2 bộ phim

```
MATCH (a:Director)-[:DIRECTED]->(m:Movie) WITH a, collect(m.title) AS directed
WHERE length(directed) >= 2
RETURN a.name, directed, ID(a) AS aID
```

Kết quả của hai câu truy vấn trên được kết lại dựa trên trường aID.

3. **Mẫu truy vấn lồng có mệnh đề OPTIONAL:**

- Dạng mẫu **M3**:

  [MATCH]

  [OPTIONAL MATCH] … [WITH]…

  [OPTIONAL MATCH] … [WITH]…

  …

[WHERE]

[RETURN] [ORDER BY] [SKIP] [LIMIT]

- Mô tả: mẫu truy vấn có một mệnh đề [MATCH], nhiều cặp mệnh đề ([OPTIONAL MATCH], [WITH]) và một mệnh đề [RETURN]. Các mệnh đề còn lại như [WHERE], [ORDER BY], [SKIP] và [LIMIT] thì không bắt buộc.
- Ý nghĩa của câu truy vấn: đây là câu truy vấn lồng, trong đó một hoặc một số loại nút (tạm gọi là loại nút A) trong kết quả của mệnh đề [MATCH] sẽ được kết trái (kết về phía các quan hệ - chính là các loại nút này) với các loại nút khác (tạm gọi là loại nút B) mô tả trong mệnh đề [OPTIONAL MATCH] đầu tiên. Ý nghĩa của kết trái trong trường hợp này là ngay cả trong trường hợp không tìm được nút B cụ thể nào để kết được (có quan hệ theo mô tả trong [OPTIONAL MATCH]) với tập các nút A thì các tập các nút A này vẫn được giữ nguyên trong kết quả trả về (nếu được yêu cầu trong [WITH]). Có một hoặc một số loại nút trong [WITH] sẽ được tiếp tục kết trái với các loại nút khác trong cặp mệnh đề ([OPTIONAL MATCH], [WITH]) tiếp theo và cứ như vậy cho đến cặp [OPTIONAL MATCH], [WITH] cuối cùng.
- Viết lại truy vấn được thực hiện bằng cách tách mệnh đề [MATCH] đầu tiên, từng cặp mệnh đề [OPTIONAL MATCH] [WITH]. Đối với mỗi mệnh đề [OPTIONAL MATCH] [WITH] thì từ khóa [OPTIONAL MATCH] sẽ được thay thế thành từ khóa [MATCH], từ khóa [WITH] sẽ được thay thế thành [RETURN]. Mệnh đề [RETURN] của mỗi câu truy vấn con sẽ được thêm khóa (ID) ứng với các nút chung trong tất cả các mệnh đề [RETURN] của tất cả các câu truy vấn con nhằm thực hiện kết dữ liệu lại với nhau. Tiếp theo mỗi câu truy vấn đơn này sẽ được viết lại dựa trên các dạng con của mẫu truy vấn đơn M1.1, M1.2, M1.3 đã mô tả ở trên. Sau khi viết lại hoàn tất thì sẽ được danh sách các câu truy vấn đơn. Trường hợp mệnh đề [MATCH] đầu tiên không chứa bất kỳ loại quan hệ nào thì mệnh đề này được ghép với từng truy vấn trong danh sách truy vấn đơn. Giữa mệnh đề [MATCH] đầu tiên và mỗi câu truy vấn có thêm từ khóa [OPTIONAL]. Trường hợp mệnh đề [MATCH] đầu tiên có chứa quan hệ thì mệnh đề này cũng sẽ được viết lại dựa trên các dạng con của mẫu truy vấn đơn M1.1, M1.2, M1.3 đã mô tả ở trên. Kết quả viết lại sẽ được thêm vào danh sách truy vấn đơn ở trên để thực thi.
- Kết quả của truy vấn ban đầu:
  - Trường hợp mệnh đề [MATCH] đầu tiên không chứa quan hệ: kết quả trả về của truy vấn con ứng với cặp ([OPTIONAL MATCH]) đầu tiên kết trái với kết quả trả về của truy vấn con ứng với cặp ([OPTIONAL MATCH]) thứ hai; kết quả này được kết trái với kết quả trả về của truy vấn con ứng với cặp ([OPTIONAL MATCH]) thứ ba; và cứ lần lượt như thế cho đến câu truy vấn ứng với cặp ([OPTIONAL MATCH]) cuối cùng. Kết quả cuối cùng này cũng chính là kết quả của truy vấn ban đầu.
  - Trường hợp mệnh đề [MATCH] đầu tiên có chứa quan hệ: kết quả trả của các câu truy vấn trong mệnh đề [MATCH] này sẽ được kết lại với nhau dựa trên các trường dữ liệu chung. Tiếp theo kết quả này sẽ được kết trái với các tập dữ liệu của các câu truy vấn còn lại.
- Ví dụ: Truy vấn ban đầu tìm thông tin của bộ phim có tên là "The Matrix" và đạo diễn, diễn viên liên quan đến bộ phim này nếu có:

```
MATCH (movie:Movie {title: "The Matrix"})
OPTIONAL MATCH (director)-[:DIRECTED]->(movie)
WITH movie, COLLECT(director.name) AS directors
OPTIONAL MATCH (actor)-[:ACTS_IN]->(movie)
WITH  movie, COLLECT(actor.name) AS actors, directors
RETURN movie.title, actors, directors
```

Sẽ được tách thành ba truy vấn con, truy vấn thứ nhất tìm đạo diễn của bộ phim The Matrix

```
MATCH (movie:Movie {title: "The Matrix"})
OPTIONAL MATCH (director)-[:DIRECTED]->(movie)
RETURN movie,COLLECT(director.name) AS directors,ID(movie) AS movieID
```

truy vấn thứ hai tìm các diễn viên của bộ phim The Matrix

```
MATCH (movie:Movie {title: "The Matrix"})
OPTIONAL MATCH (actor)-[:ACTS_IN]->(movie)
```

```
WITH movie, COLLECT(actor.name) AS actors
RETURN movie.title, actors, ID(movie) AS movieID
```

Kết quả của ba câu truy vấn sẽ được kết ngoài trái lại dựa trên trường movieID.

IV. CÀI ĐẶT VÀ THỰC NGHIỆM

A. Cài đặt

Chương trình xử lý truy vấn được xây dựng dưới dạng một ứng dụng web. Nền tảng lập trình Java được sử dụng để xây dựng. Giao diện chương trình gồm một hộp thoại cho phép người dùng nhập truy vấn. Sau khi nhập liệu người dùng nhấn nút "Execute" để thực thi chương trình. Thành phần bên dưới có 3 tab. Tab "Result of distributed Neo4j" để hiển thị kết quả truy vấn trên cở sở dữ liệu phân tán. Tab "Execution Plan" thể hiện các bước thực thi truy vấn trên các phân mảnh. Tab "Result of undistributed Neo4j" để so sánh kết quả truy vấn trên cở sở dữ liệu khi không phân mảnh.

Để thuận tiện cho việc tham khảo, thử nghiệm, đánh giá của cộng đồng, chúng tôi đã triển khai hệ thống lên website: http://45.119.83.92/WebNeo4jCQL/index.jsp với giao diện dễ sử dụng và tập câu truy vấn viết sẵn bao gồm toàn bộ các truy vấn đã được thử nghiệm trong bài báo này.

B. Thực nghiệm

Dữ liệu và các truy vấn thực nghiệm được lấy từ các trang trong tên miền của nhà sản xuất Neo4j [1]. Cơ sở dữ liệu đồ thị thực nghiệm Movie gồm các thông tin sau:

- Nút: Person (50179 nút), Movie (12862 nút).
- Quan hệ: ACTS_IN, DIRECTED, FRIEND, RATED.

Cơ sở dữ liệu đồ thị Movie này được chia làm ba phân mảnh như trong ví dụ ở phần II.B ở trên:

Trong đó cấu hình các nút mạng lần lượt là:

- Máy PC1: Intel Core i5 3320M 2.60Ghz, RAM 4GB.
- Máy PC2: Intel Core 2 Duo P8600 2.40Ghz, RAM 4GB.
- Máy PC3: Intel Dual Core T4200 2.00Ghz, RAM 3GB.

Bảng 1. Số lượng thực nghiệm các mẫu truy vấn

Mẫu truy vấn	Số lượng truy vấn thử nghiệm
Mẫu truy vấn đơn	30
Mẫu truy vấn lồng không có mệnh đề OPTIONAL	5
Mẫu truy vấn lồng có mệnh đề OPTIONAL	15

Việc thực thi truy vấn trên hệ thống phân mảnh dọc, được hiện thực trong nghiên cứu này, cho kết quả trùng khớp với kết quả thực thi truy vấn trên cơ sở dữ liệu đồ thị lúc chưa phân mảnh trong tất cả 50 truy vấn thử nghiệm trong bảng 1. Điều này cho thấy thuật toán và việc hiện thực hóa thuật toán là hoàn toàn chính xác cho các dạng mẫu câu đã trình bày ở trên.

Trong khuôn khổ nghiên cứu này, chúng tôi chưa tìm thấy bất kỳ công trình nào liên quan đến việc phân mảnh dọc cơ sở dữ liệu đồ thị do đó chúng tôi chưa thể thực hiện các đánh giá so sánh về mô hình cũng như hiệu quả với các phương pháp, các nghiên cứu khác.

V. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Bài báo đã đề xuất một kiến trúc phân mảnh dọc cho cơ sở dữ liệu đồ thị Neo4j và xây dựng một module xử lý truy vấn trên kiến trúc phân mảnh dọc này. Kiến trúc phân mảnh đề xuất đơn giản, các metadata hiện được mô tả thủ công. Các nút mạng trong kiến trúc liên kết với nhau thông qua giao thức HTTP và chưa có cơ chế kiểm soát xem một nút mạng nào đó có gặp sự cố và bị ngắt khỏi mạng hay không. Module xử lý truy vấn phân tán đã giải quyết được năm dạng mẫu câu truy vấn Cypher dựa trên cơ sở các phép toán đại số quan hệ như hội, giao, hiệu, kết, … Các mẫu câu này cho phép giải quyết được phần lớn các câu truy vấn các tập hợp dữ liệu trong cơ sở dữ liệu đồ thị. Nhờ đó, các kết quả trong nghiên cứu đã có thể được sử dụng như một framework hoặc như một cơ sở dữ liệu đơn giản cho các ứng dụng thực tế có sử dụng cơ sở dữ liệu đồ thị phân tán Neo4j.

Tuy nhiên vẫn còn khá nhiều mẫu câu khác, như các mẫu câu liên quan đến việc lựa chọn và sau đó thực hiện xử lý với từng bộ dữ liệu kết quả như foreach…, case…when…, unwind…, collect..., vẫn chưa được nghiên cứu giải quyết thành công trong nghiên cứu này. Trong thuật toán xử lý phân tán đề xuất, các yếu tố để tăng hiệu năng, giảm thời gian trả lời truy vấn cũng chưa được nghiên cứu thấu đáo như: chưa sử dụng bản sao các phân mảnh để tối ưu truy

vấn; chưa tính đến việc gộp một số quan hệ (mà chứa trên một phân mảnh nào đó) trong xử lý mẫu M1.1 và M1.2, hay gộp các cặp [MATCH] [WITH] khả gộp (chứa các quan hệ trên cùng một phân mảnh và có nút chung làm cơ sở để kết) trong mẫu M2, … để giảm thời gian xử lý và trao đổi dữ liệu. Một khía cạnh khác là việc chứng minh cho nhu cầu thực tế của mô hình phân tán này, ít nhất là thông qua việc phân tích một hoặc một số bài toán thực tế, cũng chưa được trình bày trong bài báo này. Đây cũng chính là những hướng nghiên cứu tiếp theo nhằm xây dựng một hệ thống lưu trữ và truy vấn cơ sở dữ liệu đồ thị phân tán hoàn chỉnh, tối ưu và thiết thực.

TÀI LIỆU THAM KHẢO

[1] Domain chứa các tài liệu liên quan đến Neo4j do chính nhà sản xuất cung cấp, https://neo4j.com/.

[2] Nicoara, D., Kamali, S., Daudjee, K., & Chen, L. (2015, March). Hermes: Dynamic Partitioning for Distributed Social Network Graph Databases. In EDBT (pp. 25-36).

[3] Chairunnanda, P., Forsyth, S., & Daudjee, K. (2012, June). Graph data partition models for online social networks. In Proceedings of the 23rd ACM conference on Hypertext and social media (pp. 175-180). ACM.

[4] Averbuch, A., & Neumann, M. (2013). Partitioning Graph Databases-A Quantitative Evaluation. arXiv preprint arXiv:1301.5121.

[5] Nicoara, D. (2014). Distneo4j: Scaling graph databases through dynamic distributed partitioning, A thesis presented to the University of Waterloo in fulfillment of thethesis requirement for the degree of Master of Mathematics in Computer Science.

[6] Ho, L. Y., Wu, J. J., & Liu, P. (2012, June). Distributed graph database for large-scale social computing. In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on (pp. 455-462). IEEE.

BUILDING A MODULE TO HANDLE DISTRIBUTED QUERY PROCESSING ON NEO4J GRAPH DATABASE

Nguyen Duy Tan, Ngo Thanh Hung

ABSTRACT: *This paper presents the distributed architecture and distributed query processing module on Neo4j graph database. The proposed architecture is vertical fragmentation. Fragments are placed on computers, which are connected to each other through the protocol HTTP. Distributed query processing module was built based on relational algebra for handling some samples of queries, written in Cypher language. It can be used in some practical case study and be developed with more sample of queries and higher degree of performance.*