ĐẠI HỌC QUỐC GIA TP. HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

TIỂU LUẬN TỔNG QUAN LUẬN ÁN TIẾN SĨ

NCS: Phan Hồng Trung
Khóa: 12 – Đợt 2 – 2018
Mã số: N180201
Chuyên Ngành: Công Nghệ Thông Tin (62-48-02-01)
Khoa: Khoa Học & Kỹ Thuật Thông Tin

Tên Đề Tài:

PHÁT TRIỂN MÔ HÌNH QUẢN LÝ MẠNG THÔNG TIN LỚN

(Developing a Management Model of Large Information Networks)

Cán bộ hướng dẫn: GS.TS ĐỖ PHÚC

TP.Hồ Chí Minh, tháng 12/2023

ĐẠI HỌC QUỐC GIA TP. HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

TIỂU LUẬN TỔNG QUAN LUẬN ÁN TIẾN SĨ

NCS: Phan Hồng Trung
Khóa: 12 – Đợt 2 – 2018
Mã số: N180201
Chuyên Ngành: Công Nghệ Thông Tin (62-48-02-01)
Khoa: Khoa Học & Kỹ Thuật Thông Tin

Tên Đề Tài:

PHÁT TRIỂN MÔ HÌNH QUẢN LÝ MẠNG THÔNG TIN LỚN

(Developing a Management Model of Large Information Networks)

Cán bộ hướng dẫn: GS.TS ĐỖ PHÚC

XÁC NHẬN CỦA CÁN BỘ HƯỚNG DẪN — NGHIÊN CỨU SINH

GS.TS.ĐỖ PHÚC — PHAN HỒNG TRUNG

TP.Hồ Chí Minh, tháng 12/2023

MỤC LỤC

DANH MỤC BẢNG

DANH MỤC HÌNH

DANH MỤC THUẬT NGỮ

BERT	Bidirectional Encoder Representations from Transformers - BERT là một mô hình ngôn ngữ được tạo ra bởi Google AI dùng trong xử lý ngôn ngữ tự nhiên để biến đổi từ thành vector
C-HIN	Content-Enriched Heterogeneous Information Network - Mạng thông tin không đồng nhất giàu nội dung
DKD-Tree	Distributed KD-Tree là phương pháp lập chỉ mục không gian metric trong môi trường phân tán, giúp giải quyết hiệu quả các bài toán Range Query và kNN Query trong không gian metric lớn
HIN	Heterogeneous Information Network - Mạng thông tin không đồng nhất
KD-Tree	KD-Tree là phương pháp lập chỉ mục không gian metric, giúp giải quyết hiệu quả các bài toán Range Query và kNN Query
đồ thị tri thức	Knowledge graph - Đồ thị tri thức là một đồ thị biểu diễn tri thức dưới dạng mối quan hệ giữa các thực thể

DANH MỤC TỪ VIẾT TẮT

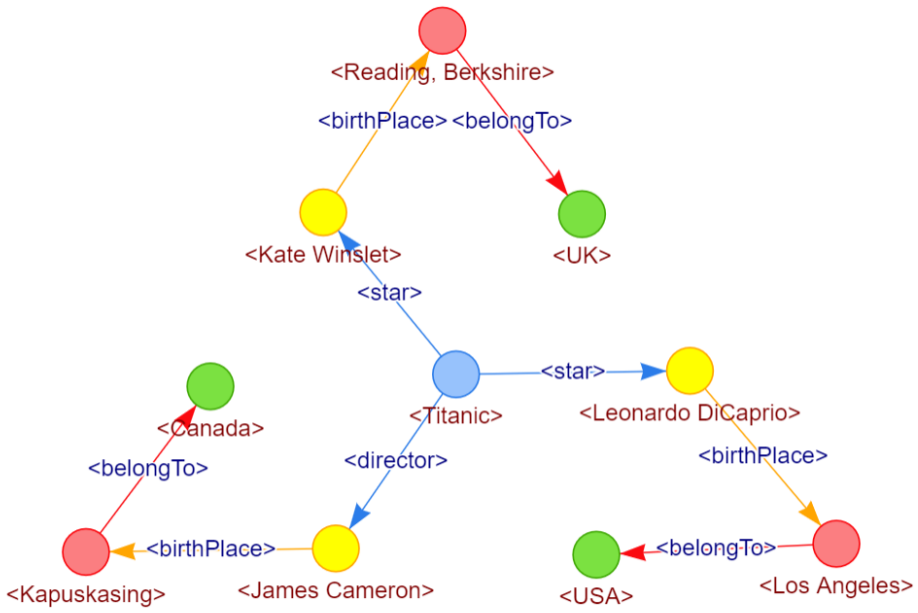
KG	Knowledge Graph - Đồ thị tri thức
QAS	Query Answering System - Hệ thống trả lời câu hỏi

Chương 1

TỔNG QUAN VỀ LUẬN ÁN

1.1 Giới thiệu đề tài

Mạng thông tin là một đồ thị có hướng biểu diễn mối quan hệ đa dạng giữa các kiểu thực thể [1]. Mạng thông tin lớn là mạng thông tin không thể lưu trữ và xử lý bằng một máy tính. Hình 1.1 là một ví dụ về mạng thông tin đơn giản, biểu diễn mối quan hệ giữa các kiểu thực thể <Movie>, <Human>, <City> và <Country>. Ngày nay, mạng thông tin lớn đã trở thành một nội dung quan trọng và không thể phủ nhận trong thế giới kỹ thuật số. Sự gia tăng nhanh chóng của dữ liệu, từ các nguồn như mạng xã hội, thiết bị di động và cảm biến, đã đặt ra những thách thức đồng thời mang lại cơ hội cho việc cải tiến và phát triển. Điều này đặt ra một yêu cầu cấp thiết là phải có những mô hình quản lý mạng thông tin lớn hiệu quả để lưu trữ và truy xuất thông tin nhanh chóng và chính xác.



Hình 1.1. Ví dụ về mạng thông tin

Đề tài "Phát Triển Mô Hình Quản Lý Mạng Thông Tin Lớn"không chỉ là một nhiệm vụ kỹ thuật mà còn là một thách thức với sự đa dạng và phức tạp của dữ liệu ngày càng tăng. Trong bối cảnh này, nghiên cứu và phát triển mô hình quản lý mạng thông tin lớn đang trở thành một lĩnh vực nghiên cứu quan trọng, với mục tiêu tối ưu hóa việc xử lý dữ liệu, đảm bảo tính nhất quán và đồng bộ, cũng như tối ưu hóa hiệu suất toàn diện của hệ thống.

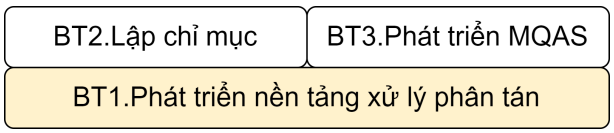
1.2 Mục tiêu và phạm vi nghiên cứu

Toàn bộ luận án sẽ được chia thành ba bài toán như trong Hình 1.2:

BT1. Phát triển nền tảng xử lý phân tán (báo cáo trong chuyên đề 3)
Chúng tôi tập trung nghiên cứu giải quyết các vấn đề sau:
1) Nghiên cứu phương pháp lưu trữ dữ liệu phân tán để lưu trữ dữ liệu lớn.
2) Phát triển hệ thống xử lý phân tán để triển khai các ứng dụng phân tán cũng như huấn luyện phân tán mạng nơ-ron sâu (Deep Neural Network - DNN).

BT2. Lập chỉ mục phân tán (báo cáo trong chuyên đề 1)
Chúng tôi tập trung nghiên cứu giải quyết vấn đề lập chỉ mục phân tán tập vector nhiều chiều qui mô lớn để tìm kiếm nhanh vector tương đồng với vector cho trước dựa trên nền tảng xử lý phân tán đã phát triển trong BT1. Kết quả bài toán này sẽ được áp dụng vào BT3 để tìm kiếm nhanh câu trả lời cho một câu hỏi.

BT3. Phát triển hệ thống trả lời câu hỏi nhiều bước (Multi-hop Query Answering System - MQAS) theo tiếp cận phân tán (báo cáo trong chuyên đề 2)
Chúng tôi tập trung nghiên cứu phát triển MQAS dựa trên mạng thông tin lớn theo tiếp cận phân tán. Nội dung cụ thể là:
1) Xây dựng một mạng thông tin cung cấp tri thức cho MQAS.
2) Phát triển MQAS với tri thức được rút trích từ mạng thông tin đã xây dựng dựa trên nền tảng xử lý phân tán đã phát triển trong BT1.



Hình 1.2. Cấu trúc luận án

Như vậy, nền tảng xử lý phân tán trong BT1 được áp dụng để phát triển BT2 và BT3.

1.3 Phương pháp nghiên cứu

Các phương pháp nghiên cứu chính được áp dụng trong luận án để giải quyết các bài toán đặt ra bao gồm:

1. **Phương pháp nghiên cứu tổng luận:** NCS dành phần lớn thời gian cho việc tham khảo các tài liệu cũng như các công trình, mô hình đề xuất đã công bố liên

quan đến bài toán chính của luận án. Từ đó kế thừa các thành quả đã đạt được, rút ra các hạn chế còn tồn tại cần phải giải quyết trong các nghiên cứu đề xuất để làm động lực nghiên cứu chính cho các vấn đề đặt ra của luận án.

2. **Phương pháp phân tích:** thông qua việc triển khai giải pháp trên các nền tảng phổ biến, NCS đã phân tích ưu và nhược điểm của các nền tảng trong việc hỗ trợ triển khai giải pháp, từ đó đề xuất giải pháp để khắc phục các nhược điểm đó. Ví dụ, khi triển khai ứng dụng phân tán trên Spark, qua phân tích, NCS và GVHD nhận thấy một số hạn chế của Spark gây trở ngại cho việc triển ứng dụng như: tư duy lập trình bị phụ thuộc vào cấu trúc Map/Reduce, việc chia sẻ dữ liệu giữa master node và worker nodes rất hạn chế chỉ thông qua broadcast và accumulated variables.

3. **Phương pháp nghiên cứu thực nghiệm và so sánh:** nhằm để chứng minh được tính hiệu quả và khả thi của các mô hình cải tiến đã đề xuất, NCS tiến hành hàng loạt các thực nghiệm trên các tập dữ liệu chuẩn, là các tập dữ liệu được chấp nhận và sử dụng rộng rãi trong lĩnh vực nghiên cứu về mạng thông tin, điển hình như: Yago, FreeBase, DBLP... Các kết quả thực nghiệm sau đó được so sánh với các công trình khác hoặc được đánh giá bằng các chỉ số chuẩn để chứng minh tính hiệu quả của mô hình được đề xuất.

1.4 Kết quả nghiên cứu

Chúng tôi đã đạt được những kết quả như sau:

1. Chúng tôi đã trích xuất dữ liệu từ bộ dữ liệu YAGO 4 và xây dựng một mạng thông tin lớn phục vụ cho hệ thống trả lời câu hỏi.

2. Chúng tôi đã phát triển nền tảng xử lý phân tán DDLF (Distributed Deep Learning Framework) giúp triển khai các ứng dụng phân tán và huấn luyện phân tán DNN dễ dàng và hiệu quả. DDLF được dùng để triển khai cấu trúc DKD-Tree và MQAS.

3. Chúng tôi đã phát triển cấu trúc DKD-Tree để lập chỉ mục phân tán tập vector nhiều chiều qui mô lớn để tìm kiếm nhanh vector tương đồng với vector cho trước.

4. Chúng tôi đã thiết kế và triển khai MQAS dựa trên mạng thông tin lớn. Hệ thống này có thể đưa ra các câu trả lời chính xác và đầy đủ cho các câu hỏi nhiều bước.

1.5 Đóng góp của nghiên cứu

Nghiên cứu của chúng tôi có những đóng góp như sau:

1. Xây dựng mạng thông tin cung cấp tri thức cho MQAS với dữ liệu được rút trích từ bộ dữ liệu YAGO 4.

2. Phát triển nền tảng xử lý phân tán giúp triển khai các ứng dụng phân tán và huấn luyện phân tán DNN dễ dàng và hiệu quả, tạo điều kiện thuận lợi trong nghiên cứu và triển khai ứng dụng phân tán.

3. Lập chỉ mục phân tán tập vector nhiều chiều qui mô lớn để tìm kiếm nhanh vector tương đồng với vector cho trước.

4. Phát triển MQAS dựa trên mạng thông tin lớn, cung cấp các câu trả lời chính xác và nhanh chóng cho người dùng.

1.6 Cấu trúc của chuyên đề

Cấu trúc của chuyên đề được chia thành 5 chương. Nội dung các chương bao gồm:

1. **CHƯƠNG 1: TỔNG QUAN VỀ LUẬN ÁN** Trình bày khái quát về đề tài; nêu các mục tiêu, phương pháp và kết quả nghiên cứu.

2. **CHƯƠNG 2: CƠ SỞ LÝ THUYẾT & CÁC MÔ HÌNH LIÊN QUAN** Trình bày cơ sở lý thuyết và các mô hình liên quan. Phân tích ưu và nhược điểm của các mô hình. Các ưu điểm sẽ được kế thừa, còn các nhược điểm được rút ra để làm động lực nghiên cứu.

3. **CHƯƠNG 3: BÀI TOÁN 1** Phát triển nền tảng xử lý phân tán.

4. **CHƯƠNG 4: BÀI TOÁN 2** Lập chỉ mục phân tán.

5. **CHƯƠNG 5: BÀI TOÁN 3** Phát triển MQAS theo tiếp cận phân tán.

6. **CHƯƠNG 6: KẾT LUẬN & HƯỚNG PHÁT TRIỂN** trình bày kết luận chung và hướng phát triển trong tương lai.

Ngoài ra, trong phần "KẾT QUẢ ĐẠT ĐƯỢC", chúng tôi liệt kê các bài báo liên quan đến việc nghiên cứu đã được đăng hoặc chấp thuận đăng.

Chương 2

CƠ SỞ LÝ THUYẾT & CÁC MÔ HÌNH LIÊN QUAN

2.1 Cơ sở lý thuyết

2.1.1 Mạng thông tin (Information Network - IN)

Định nghĩa 2.1. Mạng thông tin

Mạng thông tin là một đồ thị có hướng $G = (V, E)$ kết hợp với hàm ánh xạ loại đối tượng $\varphi : V \longrightarrow A$ và hàm ánh xạ loại liên kết $\psi : E \longrightarrow R$. Mỗi đối tượng $v \in V$ thuộc một loại đối tượng cụ thể trong tập hợp $A : \varphi(v) \in A$ và mỗi liên kết $e \in E$ thuộc một loại quan hệ cụ thể trong tập hợp $R : \psi(e) \in R$ [1].

2.1.2 Mạng thông tin không đồng nhất (Heterogeneous Information Network - HIN)

Định nghĩa 2.2. Mạng thông tin không đồng nhất

Một mạng thông tin được gọi là mạng thông tin không đồng nhất nếu số loại đối tượng $|A| > 1$ hoặc số loại quan hệ $|R| > 1$; ngược lại, nó là một mạng thông tin đồng nhất [1].

2.1.3 Mạng thông tin không đồng nhất giàu nội dung (Content-Enriched Heterogeneous Information Network - C-HIN)

Định nghĩa 2.3. Mạng thông tin không đồng nhất giàu nội dung

Nếu tất cả các đỉnh và cạnh của HIN có văn bản mô tả thì HIN được gọi là C-HIN.

Ví dụ về *C-HIN* được trình bày trong Table 5.1 và Table 5.2.

Bảng 2.1. Các thực thể và mô tả của chúng

Thực Thể	**Mô Tả**
Hà Nội	Hà Nội là thủ đô của Việt Nam. Hà Nội là trung tâm thương mại, văn hóa và giáo dục của miền Bắc Việt Nam.
Huế	Huế là thành phố cổ của Việt Nam. Có rất nhiều cảnh đẹp ở Huế.
Đà Nẵng	Đà Nẵng là một thành phố ven biển ở miền Trung Việt Nam. Đà Nẵng là trung tâm thương mại và giáo dục của miền Trung Việt Nam.
Quảng Ngãi	Quảng Ngãi là một thành phố ở miền Trung Việt Nam. Sông Trà là một thắng cảnh thiên nhiên của Quảng Ngãi.
Hồ Tây	Hồ Tây là hồ nước ngọt lớn nhất của Hà Nội, Việt Nam; nằm về phía Tây Bắc của trung tâm thành phố.
Núi Ngự Bình	Núi Ngự Bình là một ngọn núi ở Huế. Núi Ngự Bình nằm cách trung tâm Huế 30 km.
Núi Ngũ Hành	Núi Ngũ Hành, hay còn gọi là Ngũ Hành Sơn, là một cụm năm ngọn núi đá cẩm thạch và đá vôi ở Đà Nẵng. Tất cả các ngọn núi đều có lối vào hang động và nhiều đường hầm.

Bảng 2.2. Các quan hệ/vị từ và mô tả của chúng

Quan Hệ/Vị Từ	**Mô Tả**
Specialty_Dish_Of	là đặc sản của
Born_In	được sinh ở
Located_At	được đặt ở
Is_In	ở trong
Folk_Song_Of	là dân ca của
Organized_At	được tổ chức ở
Is_Beautiful_Sight_Of	là thắng cảnh của

2.1.4 Mạng thông tin lớn (Large Information Network - LIN)

Mạng thông tin lớn thường được sử dụng để mô tả mạng thông tin đa dạng có quy mô lớn. Tuy nhiên, không có một định nghĩa cụ thể thế nào là lớn, vì điều này phụ thuộc vào ngữ cảnh cụ thể và yêu cầu của ứng dụng hoặc nghiên cứu.

Dưới đây là một số yếu tố có thể xem xét khi đánh giá liệu một IN nên được coi là LIN:

1. **Quy mô đối tượng và mối quan hệ**: Nếu mạng chứa một lượng lớn đối tượng

và mối quan hệ giữa chúng, đặc biệt là so với quy mô thông thường trong ứng dụng hay nghiên cứu cụ thể, thì có thể coi đó là một LIN.

2. **Số lượng đỉnh và cạnh**: Nếu số lượng đỉnh và cạnh trong mạng là lớn, vượt qua một ngưỡng nào đó, nó có thể được xem xét là một LIN. Ngưỡng này phụ thuộc vào bối cảnh và yêu cầu cụ thể.

3. **Độ phức tạp của mối quan hệ**: Nếu mối quan hệ giữa các đối tượng là phức tạp và đa dạng, có thể coi mạng đó là một LIN.

4. **Khối lượng dữ liệu**: Nếu mạng chứa một lượng lớn dữ liệu đính kèm, bao gồm các thuộc tính của đỉnh và cạnh, thì đó cũng là một yếu tố quan trọng.

5. **Yêu cầu về tài nguyên tính toán**: Nếu việc xử lý và phân tích mạng đòi hỏi tài nguyên tính toán lớn, có thể nói rằng mạng đó là một LIN.

Một LIN thường xuyên xuất hiện trong các lĩnh vực như mạng xã hội, hệ thống thương mại điện tử với nhiều loại đối tượng và mối quan hệ, hay trong lĩnh vực nghiên cứu khoa học với dữ liệu đa dạng và phức tạp. Quyết định liệu một IN có nên được coi là lớn hay không phụ thuộc vào ngữ cảnh và mục tiêu cụ thể của ứng dụng hoặc nghiên cứu.

Như vậy, LIN là một loại mạng phức tạp mà trong đó dữ liệu và thông tin được tổ chức và liên kết thông qua mối quan hệ phức tạp giữa các đối tượng khác nhau. Đặc điểm chính của LIN là sự đa dạng về loại đối tượng, mối quan hệ, và thông tin. Dưới đây là một số đặc điểm quan trọng của LIN:

1. **Đa dạng về loại đối tượng**: LIN thường bao gồm nhiều loại đối tượng khác nhau như người, đồ vật, sự kiện, địa điểm... Mỗi loại đối tượng có thể được biểu diễn bằng các đỉnh trong đồ thị.

2. **Mối quan hệ đa dạng**: Các đối tượng trong LIN không chỉ tồn tại độc lập mà còn có mối quan hệ phức tạp với nhau. Mối quan hệ này có thể là bạn bè, tương tác, mua bán, nơi làm việc...

3. **Thuộc tính và thông tin phong phú**: Mỗi đối tượng và mối quan hệ có thể được mô tả bởi nhiều thuộc tính và thông tin chi tiết. Dữ liệu có thể bao gồm văn bản, hình ảnh, video, và các định dạng khác.

4. **Cấu trúc đồ thị phức tạp**: LIN được mô hình hóa dưới dạng đồ thị, trong đó các đỉnh đại diện cho đối tượng và các cạnh đại diện cho mối quan hệ giữa chúng. Do có rất nhiều loại đối tượng và loại quan hệ nên cấu trúc đồ thị rất phức tạp.

5. **Khối lượng dữ liệu lớn**: LIN thường đi kèm với lượng dữ liệu lớn, cần được xử lý và phân tích bằng các phương tiện hiệu quả.

6. **Đa nhiệm và phức tạp**: Các nhiệm vụ như phân loại đối tượng, dự đoán mối quan hệ, và khai phá tri thức có thể trở nên phức tạp do sự đa dạng và phức tạp của LIN.

LIN thường xuất hiện trong nhiều lĩnh vực như mạng xã hội, e-commerce, y học, và nhiều ngành công nghiệp khác. Việc hiểu và quản lý hiệu quả mạng thông tin lớn đòi hỏi sự sáng tạo trong việc phát triển mô hình, thuật toán và kỹ thuật xử lý dữ liệu lớn.

2.1.5 Đồ thị tri thức

Đồ thị tri thức là một đồ thị biểu diễn những mối quan hệ giữa các thực thể. Một thực thể có thể là một con người, một địa danh, một sự kiện, một tổ chức... Một mối quan hệ là sự kết hợp giữa hai thực thể thông qua một predicate và được biểu diễn bằng một bộ ba *<thực thể đầu, vị từ, thực thể đuôi>*. Đồ thị tri thức dùng để lưu trữ và biểu diễn tri thức. Nó cho phép con người và máy tính khai thác tốt hơn các mối quan hệ giữa các thực thể và suy luận ra những mối qua hệ mới. Về bản chất, đồ thị tri thức là một HIN hoặc một C-HIN. Đồ thị tri thức được xem là một trong số các nền tảng của AI. Ngày nay, nhiều công ty như Google, Microsoft, Facebook... đã phát triển đồ thị tri thức để phục vụ công việc của mình. Đồ thị tri thức của Google và Microsoft phục vụ cho công cụ tìm kiếm, đồ thị tri thức của Facebook phục vụ cho mạng xã hội. Về mặt hình thức, chúng ta có thể định nghĩa đồ thị tri thức như sau:

Định nghĩa 2.4. Đồ thị tri thức

Đồ thị tri thức $G = (V, E)$ là một đồ thị có hướng với các đỉnh là thực thể và các cạnh là bộ ba có dạng <thực thể đầu, vị từ, thực thể đuôi>. Mỗi bộ ba được ký hiệu là <h, p, t> để biểu diễn mối quan hệ p từ thực thể h đến thực thể t.

2.1.6 Lược đồ mạng

Định nghĩa 2.5. Lược đồ mạng (Network Schema)

Cho trước HIN $G = (V, E)$ cùng với ánh xạ kiểu đối tượng $\varphi : V \longrightarrow A$ và ánh xạ kiểu liên kết $\psi : E \longrightarrow R$. Lược đồ mạng của HIN là một đồ thị có hướng được định nghĩa trên các loại đối tượng A và các loại quan hệ R, ký hiệu $TG = (A, R)$ [1].

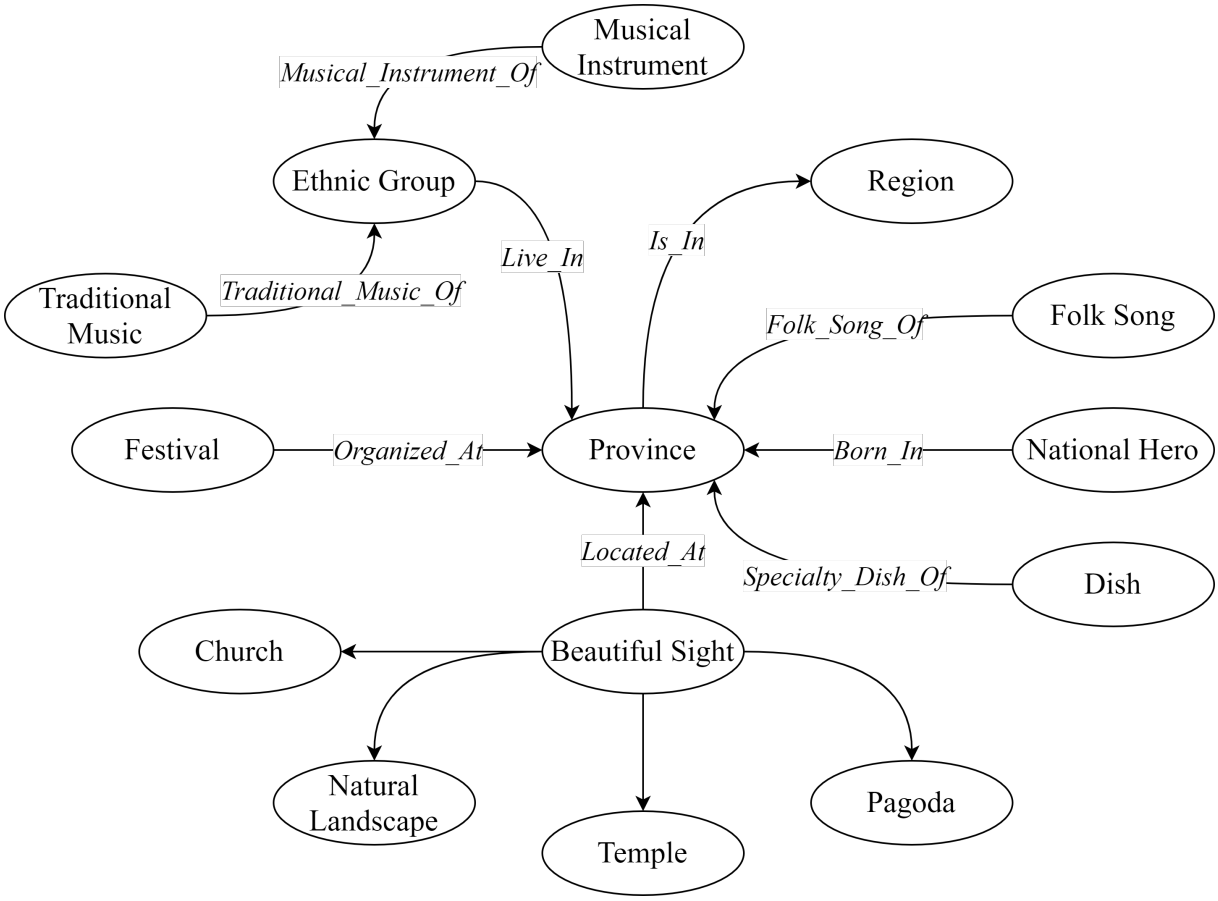
Chúng tôi đã xây dựng *C-HIN* về Du lịch Việt Nam. Lược đồ mạng của *C-HIN* này được hiển thị trong Figure 5.1. Trong *C-HIN* này, chúng tôi có một tập hợp các loại đối tượng A = *{Beautiful_Sight, Ethnic_Group, Province, Dish, National_Hero, Folk_Song, Traditional_Music, Festival, Region...}* và một tập hợp các loại quan hệ R = *{Organized_At, Located_At, Folk_Song_Of, Born_In, Specialty_Dish...}*.

2.1.7 Meta-path

Định nghĩa 2.6. Meta-path

Một meta-path P là một con đường được định nghĩa trên lược đồ mạng $TG = (A, R)$ của một HIN cho trước, được ký hiệu bằng $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} ... \xrightarrow{R_l} A_{l+1}$, định nghĩa một quan hệ kết hợp $R = R_1 \circ R_2 \circ ... \circ R_l$ giữa kiểu A_1 và A_{l+1}, trong đó $\circ$ là phép toán kết hợp trên quan hệ, l là chiều dài của meta-path [2].

Như vậy, nếu $l = 1$ thì R là quan hệ trực tiếp, nếu $l > 1$ thì R là quan hệ gián tiếp. Table 5.3 trình bày một số meta-path được rút trích ra từ lược đồ mạng của *C-HIN* về Du lịch Việt Nam.



Hình 2.1. Lược đồ mạng của *C-HIN* về Du lịch Việt Nam

Bảng 2.3. Một số meta-path của *C-HIN* về Du lịch Việt Nam

#	**Meta-Path**	**Relation Label**	**Relation Type**
1	Beautiful_Sight → Province		
		Located_At	direct
2	National_Hero → Province		
		Born_In	direct
3	Dish → Province		
		Specialty_Dish	direct
4	Folk_Song → Province		
		Folk_Song_of_Province	direct
5	Festival → Province		
		Organized_at_Province	direct
6	Beautiful_Sight → Province → Region		
		Beautiful_Sight_of_Region	indirect

#	Meta-Path	Relation Label	Relation Type
7	Dish → Province → Region		
		Specialty_Dish_of_Region	indirect
8	Folk_Song → Province → Region		
		Folk_Song_of_Region	indirect

2.1.8 Path-instance

Định nghĩa 2.7. Path-instance

Một path-instance của một meta-path $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} ... \xrightarrow{R_l} A_{l+1}$ *là một đường đi p trong HIN, trong đó* $p : a_1 \xrightarrow{R_1} a_2 \xrightarrow{R_2} ... \xrightarrow{R_l} a_{l+1}$, a_i *là một đỉnh của HIN,* $a_j \xrightarrow{R_j} a_{j+1}$ *là một cạnh với đỉnh nguồn là* a_j *và đỉnh đích là* a_{j+1} *trong HIN.*

Một vài path-instance của meta-path $Beautiful\_Sight \xrightarrow{Located\_At} Province$ được trình bày trong Table 5.4.

Bảng 2.4. Một vài path-instance của meta-path $Beautiful\_Sight \xrightarrow{Located\_At} Province$

#	Path-instance
1	$(Ngu\ Binh\ Mountain) \xrightarrow{Located\_At} (Hue)$
2	$(Ngu\ Hanh\ Mountain) \xrightarrow{Located\_At} (Da\ Nang)$
3	$(Hoan\ Kiem\ Lake) \xrightarrow{Located\_At} (Ha\ Noi)$
4	$(Ha\ Long\ Bay) \xrightarrow{Located\_At} (Quang\ Ninh)$

2.2 Tổng quan về phát triển mô hình quản lý mạng thông tin lớn

Phát triển mô hình quản lý mạng thông tin lớn là một quá trình phức tạp và đòi hỏi sự kết hợp của nhiều kỹ thuật và công việc khác nhau. Dưới đây là một số công việc quan trọng trong quá trình phát triển mô hình quản lý mạng thông tin lớn:

1. **Thu thập dữ liệu:** Thu thập dữ liệu mạng thông tin lớn từ các nguồn đa dạng như cơ sở dữ liệu, tài liệu, trang web, và các nguồn dữ liệu khác. Xử lý và làm sạch dữ liệu để loại bỏ nhiễu và chuyển đổi dữ liệu về định dạng thích hợp cho việc phân tích.

2. **Biểu diễn đồ thị và cấu trúc dữ liệu:** Xây dựng đồ thị đa dạng để mô hình hóa mối quan hệ giữa các đối tượng trong mạng thông tin lớn. Sử dụng cấu trúc dữ liệu hiệu quả để lưu trữ thông tin về đỉnh, cạnh và thuộc tính của mạng.

3. **Nhúng đồ thị:** Biến đổi các đỉnh và cạnh trong đồ thị vào không gian số có chiều thấp để tạo ra biểu diễn số của đồ thị. Sử dụng kỹ thuật nhúng đồ thị như DeepWalk, node2vec, GraphSAGE để giảm chiều của đồ thị và bảo toàn thông tin quan trọng.

4. **Phát triển mô hình học máy:** Xây dựng các mô hình học máy để thực hiện các nhiệm vụ như dự đoán mối quan hệ giữa các đối tượng, phân loại đối tượng, hoặc thậm chí dự đoán cấu trúc mới của đồ thị. Sử dụng kỹ thuật học máy như mạng nơ-ron, học máy đa nhiệm, và học tăng cường để cải thiện hiệu suất.

5. **Điều chỉnh và đánh giá mô hình:** Tối ưu hóa các tham số của mô hình để đảm bảo hiệu suất tốt nhất trên dữ liệu. Sử dụng các phương pháp đánh giá như đánh giá chéo (cross-validation) để đảm bảo tính tổng quát của mô hình.

6. **Khai phá tri thức:** Sử dụng mô hình để khai thác tri thức từ dữ liệu, như việc rút trích mối quan hệ ẩn giữa các đối tượng hoặc dự đoán thông tin còn thiếu.

7. **Triển khai và quản lý mô hình:** Triển khai mô hình vào môi trường thực tế để sử dụng trong các ứng dụng cụ thể. Quản lý và theo dõi hiệu suất của mô hình, cũng như cập nhật nó khi có dữ liệu mới hoặc khi cần thiết.

8. **Bảo mật và quyền riêng tư:** Xem xét và triển khai các biện pháp bảo mật để đảm bảo an toàn cho dữ liệu và mô hình. Tuân thủ các chuẩn quyền riêng tư và chính sách để bảo vệ thông tin cá nhân.

Phát triển mô hình quản lý LIN đòi hỏi sự kết hợp của kiến thức của nhiều lĩnh vực như đồ thị lớn, học máy, và khoa học dữ liệu. Các công việc trên đưa ra một khung làm việc tổng quan cho quá trình này.

2.3 Các mô hình liên quan

2.3.1 Các giải pháp lưu trữ LIN

Bảng 2.5 trình bày các giải pháp lưu trữ LIN phổ biến hiện nay.

Bảng 2.5. Các giải pháp lưu trữ LIN phổ biến

#	Giải Pháp	Ưu Điểm	Nhược Điểm
1	**Lưu trữ đám mây**		
	Lưu trữ đám mây là một giải pháp lưu trữ LIN hiệu quả và linh hoạt. Dữ liệu của mạng được lưu trữ trên các máy chủ đám mây, cho phép truy cập từ bất kỳ đâu.	- Hiệu suất tốt - Khả năng mở rộng tốt - Truy cập từ bất kỳ đâu	- Chi phí cao - Yêu cầu kết nối internet ổn định
	Ví dụ: Amazon S3, Azure Blob Storage, Google Cloud Storage		

Bảng 2.5. Các giải pháp lưu trữ LIN phổ biến

#	Giải Pháp	Ưu Điểm	Nhược Điểm
2	**Cơ sở dữ liệu đồ thị**		
	Cơ sở dữ liệu đồ thị là một giải pháp lưu trữ LIN hiệu quả, cho phép truy vấn dữ liệu một cách hiệu quả.	- Hiệu suất tốt - Khả năng mở rộng tốt	- Chi phí cao - Yêu cầu kiến thức chuyên môn
	Ví dụ: Neo4j, OrientDB, TigerGraph		
3	**Hệ thống lưu trữ phân tán**		
	Hệ thống lưu trữ phân tán là một giải pháp lưu trữ LIN hiệu quả cho các mạng có kích thước lớn. Dữ liệu của mạng được lưu trữ trên nhiều máy chủ, cho phép cải thiện hiệu suất và khả năng mở rộng.	- Hiệu suất tốt - Khả năng mở rộng tốt - Chi phí thấp	- Phức tạp để triển khai và quản lý - Yêu cầu kết nối mạng ổn định
	Ví dụ: Hadoop Distributed File System (HDFS), Ceph, GlusterFS		

Việc chọn giải pháp lưu trữ LIN phụ thuộc vào các yếu tố sau:

- Kích thước của mạng: Nếu mạng có kích thước lớn, cần sử dụng giải pháp lưu trữ phân tán hoặc cơ sở dữ liệu đồ thị.
- Các yêu cầu truy cập: Nếu cần truy cập dữ liệu từ bất kỳ đâu, cần sử dụng giải pháp lưu trữ đám mây.
- Khả năng chi trả: Chi phí của các giải pháp lưu trữ LIN có thể khác nhau.
- Kiến thức và kinh nghiệm: Một số giải pháp lưu trữ LIN yêu cầu kiến thức và kinh nghiệm chuyên môn.

2.3.2 Các framework xử lý LIN

Sau đây là các nền tảng xử lý LIN phổ biến hiện nay.

2.3.2.1 Apache Giraph

Apache Giraph là một dự án mã nguồn mở được Apache Software Foundation phát triển, chủ yếu dành cho xử lý đồ thị phân tán. Dự án này được xây dựng trên cơ sở của Apache Hadoop và được thiết kế để giải quyết các vấn đề liên quan đến tính toán và xử lý đồ thị lớn [3, 4].

Ưu điểm của Giraph:

1) Xử lý đồ thị lớn: Giraph được thiết kế đặc biệt để xử lý đồ thị lớn có cấu trúc phức tạp.

2) Model tính toán BSP (Bulk Synchronous Parallel): Sử dụng mô hình tính toán BSP giúp Giraph đảm bảo tính nhất quán và hiệu suất trong quá trình xử lý.

3) Sự Tích hợp với Hadoop: Giraph tích hợp chặt chẽ với Hadoop Distributed File System (HDFS) và có thể sử dụng cơ sở hạ tầng của Hadoop để quản lý lưu trữ và tính toán đồ thị.

4) Hỗ trợ nhiều loại đồ thị: Giraph có thể xử lý nhiều loại đồ thị khác nhau, bao gồm cả đồ thị hướng và vô hướng.

5) Thư viện đa dạng: Cung cấp một số lượng lớn các thuật toán đồ thị sẵn có, giúp người phát triển dễ dàng triển khai ứng dụng của mình.

Nhược điểm của Giraph:

1) Khả năng mở rộng: Mặc dù Giraph có khả năng xử lý đồ thị lớn, nhưng khả năng mở rộng của nó có thể bị hạn chế đối với các đồ thị rất lớn và phức tạp.

2) Hiệu suất đối với một số ứng dụng: Các mô hình tính toán BSP có thể không phải là lựa chọn tốt nhất cho mọi loại ứng dụng, đặc biệt là những ứng dụng yêu cầu tính linh hoạt và đáp ứng cao.

3) Thời gian triển khai và cấu hình: Việc triển khai và cấu hình Giraph có thể đòi hỏi kiến thức sâu rộng về Hadoop và Giraph, làm tăng độ phức tạp của quá trình triển khai.

4) Cộng đồng và hỗ trợ: Giraph có cộng đồng và hỗ trợ tương đối nhỏ so với một số hệ thống xử lý đồ thị khác, điều này có thể tạo ra khó khăn khi cần hỗ trợ hoặc gặp vấn đề.

2.3.2.2 Apache Spark

Apache Spark là một nền tảng mã nguồn mở được thiết kế để xử lý và phân tích dữ liệu lớn một cách nhanh chóng và hiệu quả. Nó cung cấp một mô hình lập trình linh hoạt và một hệ sinh thái công nghệ mạnh mẽ để thực hiện các công việc xử lý dữ liệu phân tán.

Ưu điểm của Apache Spark:

1) Hiệu suất cao: Spark nhanh hơn so với Hadoop MapReduce vì nó xử lý trong bộ nhớ (in-memory processing). Điều này giảm thời gian đọc và ghi trên đĩa, cung cấp hiệu suất tốt hơn.

2) Hỗ trợ nhiều ngôn ngữ: Spark hỗ trợ nhiều ngôn ngữ lập trình như Scala, Java, Python và R, làm cho nó trở nên linh hoạt và dễ tích hợp với nhiều ứng dụng khác nhau.

3) Hệ sinh thái đa dạng: Spark cung cấp một hệ sinh thái đa dạng với nhiều thư viện như Spark SQL, Spark Streaming, MLlib (Machine Learning Library) và GraphX (xử lý đồ thị). Điều này làm cho nó thích hợp cho nhiều loại công việc khác nhau, từ xử lý batch đến xử lý dữ liệu trực tuyến và machine learning.

4) Xử lý dữ liệu có cấu trúc và không có cấu trúc: Spark có thể xử lý cả dữ liệu có

cấu trúc và không có cấu trúc, làm cho nó phù hợp cho nhiều loại dữ liệu khác nhau.

Nhược điểm của Apache Spark:

1) Yêu cầu tài nguyên hệ thống cao: Spark yêu cầu nhiều tài nguyên hệ thống, đặc biệt là bộ nhớ, để có thể tận dụng được ưu điểm của xử lý in-memory. Điều này có thể tạo ra áp lực đối với cụm máy tính.

2) Khó làm chủ công nghệ: Việc học sử dụng Spark có thể đòi hỏi một khối lượng kiến thức lớn, đặc biệt là khi sử dụng các tính năng phức tạp như Spark Streaming hoặc machine learning trên Spark.

3) Chưa thích hợp cho tất cả các tình huống: Mặc dù Spark phù hợp cho nhiều loại công việc xử lý dữ liệu, nhưng không phải lúc nào nó cũng là lựa chọn tốt nhất.

Việc chọn giải pháp xử lý LIN phụ thuộc vào các yếu tố sau:

- Kích thước của mạng: Nếu mạng có kích thước lớn, cần sử dụng giải pháp có khả năng xử lý dữ liệu lớn.
- Các yêu cầu về tính năng: Nếu cần sử dụng các tính năng cụ thể, cần chọn giải pháp có hỗ trợ các tính năng đó.
- Kiến thức và kinh nghiệm: Nếu có kiến thức và kinh nghiệm về một nền tảng cụ thể, nên chọn nền tảng đó.

2.4 Động lực nghiên cứu

Nhìn chung, cả Apache Giraph và Apache Spark đều là những công cụ mạnh mẽ trong việc xử lý LIN. Tuy nhiên, cả hai đều gặp phải một số nhược điểm không tránh khỏi, tạo ra cơ hội cho sự nghiên cứu và phát triển một mô hình quản lý LIN mới, với mong muốn vượt qua những hạn chế hiện tại.

Apache Giraph, mặc dù được thiết kế đặc biệt cho xử lý đồ thị lớn, có thể đối mặt với khó khăn trong khả năng mở rộng và hiệu suất đối với các đồ thị cực kỳ phức tạp. Thời gian triển khai và cấu hình cũng có thể làm giảm hiệu suất và tăng khả năng phức tạp của quá trình phát triển ứng dụng. Sự hạn chế này tạo ra một động lực mạnh mẽ để nghiên cứu một mô hình mới, linh hoạt hơn và dễ triển khai hơn.

Apache Spark, mặc dù mạnh mẽ với khả năng xử lý dữ liệu lớn, nhưng cũng có nhược điểm về hiệu suất trong một số trường hợp và đòi hỏi sự tinh chỉnh kỹ thuật đặc biệt để tối ưu hóa. Sự phức tạp trong việc triển khai và tích hợp với các hệ thống khác cũng là một thách thức. Do đó, tạo ra động lực mạnh mẽ để tìm kiếm một mô hình mới, có khả năng tối ưu hóa hiệu suất và tích hợp một cách linh hoạt với các hệ thống khác.

Cấu trúc *map/reduce* của Apache Spark được áp dụng rất hiệu quả trong xử lý dữ liêu lớn nhưng không phải phù hợp với mọi loại ứng dụng vì việc lập trình bị gò bó trong cấu trúc *map/reduce*, ràng buộc tư duy lập trình, dẫn đến việc triển khai ứng dụng khác kém linh hoạt và không hiệu quả. Mặt khác, do phụ thuộc vào cấu trúc *map/reduce*, hàm *worker_train(data_iterator)* chỉ nhận một tham số *data_iterator* (là 1 data partition). Người lập trình không thể truyền thông tin khác cho hàm *worker_train()* khi cần. Dùng biện pháp thay thế thì quá phiền phức. Điều này làm khó triển khai ứng dụng.

Trong Apache Spark, cơ chế chia sẻ dữ liệu giữa master node và worker nodes rất hạn chế thông qua *broadcast và accumulated variables*. Broadcast variables là biến chỉ đọc, accumulated variables là biến chỉ cộng. Hơn nữa, các biến này có nội dung giống nhau trên tất cả các worker nodes và chúng sẽ bị mất khi kết thúc một Spark session.

Với các thách thức này, nghiên cứu và phát triển một mô hình quản lý mạng thông tin lớn mới trở thành một hướng đi hứa hẹn. Một mô hình có khả năng mở rộng tốt, hiệu suất cao, dễ triển khai và tích hợp linh hoạt có thể giải quyết những hạn chế của Giraph và Spark, mang lại lợi ích đặc biệt cho các ứng dụng yêu cầu xử lý mạng thông tin lớn một cách hiệu quả và linh hoạt. Điều này không chỉ giúp nâng cao khả năng xử lý dữ liệu đồ thị, mà còn thúc đẩy sự tiến bộ trong lĩnh vực quản lý thông tin lớn và phân tán.

Chương 3

PHÁT TRIỂN NỀN TẢNG XỬ LÝ PHÂN TÁN

3.1 Giới thiệu bài toán

Trong những năm gần đây, sự phát triển của dữ liệu và học máy (Machine Learning - ML) đã tạo ra nhu cầu ngày càng tăng đối với các ứng dụng và mô hình phân tán. Các ứng dụng phân tán có thể tận dụng sức mạnh tính toán của nhiều máy tính để cải thiện hiệu suất, độ chính xác và khả năng mở rộng. Các mô hình phân tán có thể được huấn luyện trên các cụm máy tính lớn để tạo ra các mô hình học máy phức tạp hơn.

Có một số nền tảng xử lý phân tán và huấn luyện DNN sẵn có. Tuy nhiên, các nền tảng này có thể có những hạn chế, chẳng hạn như:

- Không linh hoạt: Các nền tảng này có thể không đáp ứng được các nhu cầu cụ thể của ứng dụng hoặc mô hình.
- Khó sử dụng: Các nền tảng này có thể phức tạp và khó học hỏi.
- Không hiệu quả: Các nền tảng này có thể không tận dụng tối đa khả năng tính toán của các máy tính trong cụm.

Vì vậy, việc phát triển nền tảng xử lý phân tán mới có thể là một giải pháp hiệu quả để giải quyết các vấn đề này. Một nền tảng xử lý phân tán mới có thể được thiết kế để đáp ứng các nhu cầu cụ thể của ứng dụng hoặc mô hình. Nền tảng này cũng có thể được thiết kế để dễ sử dụng và hiệu quả. Dưới đây là một số lý do cụ thể tại sao nên phát triển nền tảng xử lý phân tán mới:

- Để đáp ứng các yêu cầu cụ thể của ứng dụng hoặc mô hình: Các nền tảng xử lý phân tán sẵn có có thể cung cấp các tính năng chung, nhưng chúng có thể không đáp ứng được các yêu cầu cụ thể của ứng dụng hoặc mô hình. Ví dụ, một ứng dụng có thể cần hỗ trợ cho các loại dữ liệu cụ thể hoặc các loại tính toán cụ thể. Một mô hình có thể cần hỗ trợ cho các loại kiến trúc mạng nơ-ron cụ thể hoặc các kỹ thuật học tập tăng cường cụ thể.

- Để cải thiện khả năng sử dụng: Các nền tảng xử lý phân tán sẵn có có thể phức tạp và khó học hỏi. Điều này có thể làm giảm hiệu suất của các nhà phát triển ứng dụng. Một nền tảng xử lý phân tán mới có thể được thiết kế để dễ sử dụng hơn, giúp các nhà phát triển tập trung vào việc xây dựng ứng dụng.
- Để cải thiện hiệu suất: Các nền tảng xử lý phân tán sẵn có có thể không tận dụng tối đa khả năng tính toán của các máy tính trong cụm. Điều này có thể dẫn đến hiệu suất chậm. Một nền tảng xử lý phân tán mới có thể được thiết kế để tận dụng tối đa khả năng tính toán của các máy tính trong cụm, giúp cải thiện hiệu suất của ứng dụng.

Apache Spark là một nền tảng xử lý dữ liệu lớn nổi bật và phổ biến nhất hiện nay. Vì vậy, có nhiều nền tảng huấn luyện DNN đã kết hợp với *Apache Spark* để tận dụng đặc điểm tính toán phân tán của nó. Ban đầu, chúng tôi triển khai các nội dung nghiên cứu trên *Apache Spark*. Tuy nhiên, trong quá trình phát triển các ứng dụng dựa trên *Apache Spark* để huấn luyện DNN chúng tôi gặp một số trở ngại. Ví dụ, cấu trúc *map/reduce* của *Apache Spark* rất hữu ích khi xử lý dữ liệu lớn nhưng tỏ ra không phù hợp trong việc huấn luyện DNN. Việc kết hợp gượng ép *Apache Spark* để huấn luyện DNN làm giảm hiệu suất của toàn hệ thống. Vì vậy, chúng tôi đã phát triển nền tảng xử lý phân tán DDLF, hoàn toàn độc lập với *Apache Spark*, khắc phục được những trở ngại mà chúng tôi gặp phải khi dùng *Apache Spark* để huấn luyện mạng nơ-ron và triển khai ứng dụng phân tán.

Việc phát triển nền tảng xử lý phân tán mới có thể là một giải pháp hiệu quả để giải quyết các vấn đề của các nền tảng xử lý phân tán sẵn có. Một nền tảng xử lý phân tán mới có thể được thiết kế để đáp ứng các yêu cầu cụ thể của ứng dụng hoặc mô hình, cải thiện khả năng sử dụng và cải thiện hiệu suất. Tuy nhiên, việc này cũng dẫn đến nhiều thách thức như đòi hỏi người phát triển phải có kiến thức chuyên sâu về hệ thống xử lý phân tán.

3.2 Mục tiêu nghiên cứu

Trong bài toán này, chúng tôi tập trung giải quyết vấn đề: *"Xây dựng nền tảng xử lý phân tán để có thể triển khai ứng dụng phân tán và huấn luyện phân tán DNN"*. Bài toán này được chia thành 5 phần:

- Nghiên cứu phương pháp lưu trữ dữ liệu phân tán để lưu trữ dữ liệu lớn.
- Nghiên cứu triển khai hệ thống huấn luyện phân tán DNN trên *Apache Spark*.
- Phân tích những trở ngại khi triển khai hệ thống huấn luyện phân tán DNN trên *Apache Spark* và trình bày các giải pháp khắc phục chúng.
- Phát triển nền tảng mới tên DDLF để triển khai ứng dụng phân tán và huấn luyện phân tán DNN.
- So sánh hiệu suất của DDLF và *Apache Spark* trong huấn luyện phân tán DNN.

3.3 Những công trình liên quan

Nhiều hoạt động nghiên cứu và phát triển các nền tảng huấn luyện phân tán DNN đã được thực hiện nhờ sự tiến bộ vượt bậc của học sâu DL. Li và cộng sự đã đề xuất một nền tảng máy chủ có tham số cho việc học phân tán và một số phương pháp đã được đề xuất để giảm chi phí liên lạc giữa các nút, chẳng hạn như chỉ trao đổi các giá trị tham số khác 0, lưu trữ cục bộ danh sách chỉ mục và bỏ qua ngẫu nhiên các thông báo đã truyền [5, 6].

Phương pháp này có những ưu điểm sau: 1) Tăng khả năng mở rộng: Phương pháp này giúp tăng khả năng mở rộng quá trình huấn luyện mô hình học máy trên nền tảng phân tán, cho phép sử dụng hiệu quả nguồn lực máy tính lớn. 2) Hiệu suất cao: Các cập nhật được thực hiện thông qua Parameter Server giúp giảm độ trễ và tăng hiệu suất trong quá trình huấn luyện. 3) Linh hoạt trong cách cập nhật: Khả năng hỗ trợ cả cập nhật bất đồng bộ và đồng bộ giúp tối ưu hóa quá trình huấn luyện dựa trên yêu cầu cụ thể của ứng dụng.

Tuy nhiên, phương pháp này cũng có những nhược điểm như: 1) Chi phí của Parameter Server: Việc duy trì một Parameter Server có thể tạo ra thêm chi phí và yêu cầu nguồn lực, đặc biệt là khi kích thước của mô hình và dữ liệu huấn luyện lớn. 2) Khả năng đồng bộ hóa: Trong trường hợp sử dụng đồng bộ nếu không được quản lý cẩn thận có thể dẫn đến giảm hiệu suất.

Abadi và cộng sự đã trình bày *TensorFlow*, một nền tảng tập trung cho huấn luyện DNN tích hợp cả hai phương pháp song song dữ liệu và mô hình [7]. Hệ thống TensorFlow bao gồm hai thành phần chính: 1) TensorFlow Core: Đây là thành phần cơ bản của hệ thống, cung cấp các API để xây dựng và huấn luyện các mô hình học máy. TensorFlow Core sử dụng một mô hình lập trình dựa trên tensor, cho phép người dùng biểu diễn dữ liệu và phép tính dưới dạng các tensor. 2) TensorFlow Extended (TFX): Đây là một bộ công cụ bổ sung cho TensorFlow Core, cung cấp các tính năng để chuẩn bị dữ liệu, xây dựng và triển khai các mô hình học máy. TFX bao gồm các mô-đun để thu thập dữ liệu, tiền xử lý dữ liệu, xây dựng mô hình, đánh giá mô hình và triển khai mô hình.

Ưu điểm của phương pháp này là: 1) Mở rộng và phân tán: TensorFlow được thiết kế để có thể mở rộng cho việc huấn luyện mô hình trên dữ liệu lớn và phân tán trên nhiều máy tính. 2) Kiến trúc Linh hoạt: Điều này giúp người sử dụng có thể xây dựng và triển khai mô hình máy học trên nhiều loại thiết bị. 3) Hiệu suất cao: TensorFlow được tối ưu hóa để đạt hiệu suất cao, đặc biệt là trên GPU (Graphics Processing Unit) và TPU (Tensor Processing Unit).

Tuy nhiên, phương pháp này cũng có các nhược điểm như: 1) Yêu cầu kỹ thuật cao: Cho đến một số phiên bản gần đây, TensorFlow có thể được coi là khá khó sử dụng đối với người mới bắt đầu và yêu cầu kiến thức lập trình và toán học đáng kể. 2) Cú pháp phức tạp: Một số người sử dụng cho rằng cú pháp của TensorFlow có thể phức tạp so với một số thư viện khác. 3) Kích thước lớn của thư viện: TensorFlow có kích thước lớn, điều này có thể là một vấn đề nếu bạn cần tính nhỏ gọn trong các ứng dụng di động hoặc nhúng.

Harlap và cộng sự đã đề xuất phương pháp PipeDream - huấn luyện song song đường ống (pipeline parallel training) để tăng tốc độ huấn luyện mạng nơ-ron sâu (DNN) [8].

Phương pháp này chia quá trình huấn luyện DNN thành các giai đoạn nhỏ hơn, sau đó thực hiện các giai đoạn này song song trên các thiết bị khác nhau.

Ưu điểm của phương pháp này là: 1) Hiệu suất cao: Phương pháp PipeDream có thể tăng tốc độ huấn luyện DNN lên tới 10 lần so với các phương pháp huấn luyện song song đường ống truyền thống. 2) Tương thích với các mô hình DNN hiện đại: Phương pháp PipeDream có thể được áp dụng cho các mô hình DNN hiện đại, bao gồm cả các mô hình có cấu trúc phức tạp.

Tuy nhiên, phương pháp này cũng có các nhược điểm như: 1) Yêu cầu bộ nhớ lớn: Phương pháp PipeDream yêu cầu bộ nhớ lớn để lưu trữ các bộ đệm và kết quả tính toán. 2) Khá phức tạp: Phương pháp PipeDream khá phức tạp để triển khai và tối ưu hóa.

Thế hệ tiếp theo của các ứng dụng học máy sẽ liên tục tương tác với môi trường và học từ những tương tác này. Các ứng dụng này đặt ra các yêu cầu hệ thống mới và khắt khe, cả về hiệu suất và tính linh hoạt. Moritz và cộng sự đã đề xuất *Ray* - một nền tảng phân tán cho các ứng dụng AI mới nổi. [9]. Ray cung cấp một số tính năng chính, bao gồm: 1) Quản lý tài nguyên: Ray cung cấp các tính năng để quản lý tài nguyên trên các máy tính trong cụm. Các tính năng này bao gồm khả năng tự động phân bổ tài nguyên cho các công việc, khả năng theo dõi việc sử dụng tài nguyên và khả năng giải phóng tài nguyên khi không cần thiết. 2) Luồng công việc (workflow): Ray cung cấp các tính năng để tạo và quản lý luồng công việc. Luồng công việc là một chuỗi các công việc được thực hiện theo trình tự. Ray cung cấp các tính năng để xác định các phụ thuộc giữa các công việc, để thực hiện các công việc song song và để theo dõi tiến độ của luồng công việc. 3) Đồng bộ hóa: Ray cung cấp các tính năng để đồng bộ hóa dữ liệu và trạng thái giữa các máy tính trong cụm. Các tính năng này bao gồm khả năng đồng bộ hóa dữ liệu cục bộ, khả năng đồng bộ hóa dữ liệu trên mạng và khả năng đồng bộ hóa trạng thái của các đối tượng.

Ray có thể được áp dụng cho nhiều loại ứng dụng AI mới nổi, bao gồm: 1) Đào tạo DNN: Ray có thể được sử dụng để huấn luyện DNN trên các cụm máy tính lớn. Ray cung cấp các tính năng để quản lý tài nguyên, để tạo và quản lý luồng công việc và để đồng bộ hóa dữ liệu. 2) Xử lý dữ liệu lớn: Ray có thể được sử dụng để xử lý dữ liệu lớn trên các cụm máy tính lớn. Ray cung cấp các tính năng để quản lý tài nguyên, để tạo và quản lý luồng công việc và để đồng bộ hóa dữ liệu. 3) Máy học tự động (MLOps): Ray có thể được sử dụng để triển khai và quản lý các ứng dụng ML. Ray cung cấp các tính năng để quản lý tài nguyên, để tạo và quản lý luồng công việc và để đồng bộ hóa dữ liệu.

Ray có những ưu điểm sau: 1) Hiệu quả: Ray cung cấp các tính năng để tăng hiệu quả của các ứng dụng AI mới nổi. Các tính năng này bao gồm khả năng quản lý tài nguyên, khả năng tạo và quản lý luồng công việc và khả năng đồng bộ hóa dữ liệu. 2) Tính linh hoạt: Ray là một khung công tác linh hoạt có thể được áp dụng cho nhiều loại ứng dụng AI mới nổi. 3) Tính mở: Ray là một khung công tác mở có thể được sử dụng với nhiều loại phần mềm AI.

Tuy nhiên, Ray cũng có những nhược điểm như: 1) Yêu cầu kỹ thuật: Ray yêu cầu người dùng có kiến thức về lập trình phân tán. 2) Khá phức tạp: Ray khá phức tạp để triển khai và sử dụng.

Với sự phát triển về kích thước dữ liệu và kích thước mô hình, phương pháp song song

dữ liệu không thể hoạt động trên các mô hình có kích thước tham số không thể vừa với bộ nhớ thiết bị có một GPU. Để cải thiện hơn nữa việc huấn luyện mô hình khổng lồ cấp công nghiệp, Wang và cộng sự đã giới thiệu *Whale*, một nền tảng huấn luyện phân tán thống nhất [10]. Nó cung cấp các chiến lược song song toàn diện bao gồm song song dữ liệu, song song mô hình, đường ống, chiến lược kết hợp và chiến lược song song tự động. *Whale* tương thích với *TensorFlow* và các tác vụ huấn luyện có thể dễ dàng được phân tán bằng cách thêm một vài dòng mã mà không cần thay đổi mã mô hình người dùng. Theo các tác giả, *Whale* là công trình đầu tiên có thể hỗ trợ các chiến lược phân tán kết hợp khác nhau trong một nền tảng. Trong thử nghiệm của họ trên mô hình lớn BERT, chiến lược đường ống của *Whale* nhanh hơn 2,32 lần so với song song dữ liệu của *Horovod* trên 64 GPU. Trong các tác vụ phân loại hình ảnh quy mô lớn (100.000 lớp), chiến lược kết hợp của *Whale*, bao gồm chia nhỏ toán tử và song song dữ liệu, nhanh hơn 14,8 lần so với song song dữ liệu của *Horovod* trên 64 GPU.

Whale có thể áp dụng cho nhiều loại ứng dụng ML, bao gồm: 1) Xử lý ảnh: Whale có thể được sử dụng để xử lý ảnh, chẳng hạn như nhận dạng vật thể và phân loại hình ảnh. 2) Xử lý ngôn ngữ tự nhiên: Whale có thể được sử dụng để xử lý ngôn ngữ tự nhiên, chẳng hạn như dịch ngôn ngữ và sinh văn bản. 3) Xử lý dữ liệu lớn: Whale có thể được sử dụng để xử lý dữ liệu lớn, chẳng hạn như huấn luyện mô hình học máy cho các nhiệm vụ như phân loại và dự đoán.

Các ưu điểm của Whale bao gồm: 1) Hỗ trợ cho nhiều loại mô hình ML: Whale hỗ trợ cho nhiều loại mô hình ML, điều này giúp cho Whale trở nên linh hoạt và có thể được áp dụng cho nhiều loại ứng dụng ML. 2) Hỗ trợ cho nhiều loại cụm máy tính: Whale hỗ trợ cho nhiều loại cụm máy tính, điều này giúp cho Whale có thể được triển khai trên nhiều loại môi trường. 3) Tăng hiệu quả: Whale cung cấp một số kỹ thuật để tăng hiệu quả của huấn luyện ML phân tán, điều này giúp cho Whale có thể giảm thời gian huấn luyện các mô hình ML lớn.

Các nhược điểm của Whale bao gồm: 1) Yêu cầu kỹ thuật: Whale yêu cầu người dùng có kiến thức về lập trình phân tán. 2) Khá phức tạp: Whale khá phức tạp để triển khai và sử dụng.

Tiếp theo là một số nền tảng phân tán hỗ trợ phân tích dữ liệu lớn. Hadoop là một nền tảng mã nguồn mở được phát triển bởi Apache Software Foundation để xử lý và lưu trữ lượng lớn dữ liệu trên các cụm máy tính phân tán [11][12]. Hai thành phần chính của Hadoop bao gồm: 1) Hadoop Distributed File System (HDFS): Là hệ thống lưu trữ phân tán, chia nhỏ dữ liệu thành các khối và phân tán chúng trên nhiều nút máy tính trong một cụm. Điều này tăng khả năng mở rộng và đồng thời đảm bảo tính an toàn và độ tin cậy của dữ liệu. 2) MapReduce: Là mô hình lập trình và xử lý dữ liệu phân tán. Nó chia công việc thành hai pha chính: pha Map để xử lý và ánh xạ dữ liệu, và pha Reduce để tổng hợp kết quả từ các pha Map.

Hadoop có nhiều ưu điểm như: 1) Xử lý dữ liệu lớn: Hadoop cho phép xử lý và lưu trữ lượng lớn dữ liệu một cách hiệu quả, phù hợp với môi trường big data. 2) Phân tán và khả năng mở rộng: Hadoop có khả năng phân tán dữ liệu trên nhiều máy tính, giúp tăng cường khả năng mở rộng khi cần thiết. 3) Đa dạng dữ liệu: Có thể xử lý cả dữ liệu có cấu trúc và không có cấu trúc, làm cho Hadoop linh hoạt khi làm việc với nhiều loại dữ liệu khác nhau. 4) Giá thành thấp: Hadoop có thể chạy trên cụm máy tính thông thường,

giúp giảm chi phí so với nhiều giải pháp khác.

Tuy nhiên, Hadoop có những nhược điểm sau: 1) Hiệu suất không đồng đều: Trong một số trường hợp, hiệu suất của Hadoop có thể không đồng đều do một số công việc không phù hợp với mô hình MapReduce. 2) Độ phức tạp của việc lập trình: Việc phát triển ứng dụng trên Hadoop có thể phức tạp do cần phải xử lý nhiều khía cạnh của việc phân tán dữ liệu. 3) Độ trễ cao cho các tác vụ nhỏ: Trong một số trường hợp, việc thực hiện các tác vụ nhỏ trên Hadoop có thể gặp độ trễ cao, không phù hợp cho các ứng dụng yêu cầu xử lý thời gian thực. 4) Khả năng quản lý khó khăn: Quản lý một cụm Hadoop và tối ưu hóa hiệu suất có thể đòi hỏi kiến thức chuyên sâu và kỹ năng quản trị cao.

Apache Spark là một nền tảng mã nguồn mở được thiết kế để xử lý và phân tích lượng lớn dữ liệu một cách nhanh chóng và hiệu quả. Nó cung cấp một mô hình lập trình linh hoạt và một hệ sinh thái công nghệ mạnh mẽ để thực hiện các công việc xử lý dữ liệu phân tán [13][14][15][16][17].

Ưu điểm của Apache Spark là: 1) Hiệu suất cao: Spark nhanh hơn so với Hadoop MapReduce vì nó xử lý trong bộ nhớ (in-memory processing). Điều này giảm thời gian đọc và ghi trên đĩa, cung cấp hiệu suất tốt hơn. 2) Hỗ trợ nhiều ngôn ngữ: Spark hỗ trợ nhiều ngôn ngữ lập trình như Scala, Java, Python và R, làm cho nó trở nên linh hoạt và dễ tích hợp với nhiều ứng dụng khác nhau. 3) Hệ sinh thái đa dạng: Spark cung cấp một hệ sinh thái đa dạng với nhiều thư viện như Spark SQL, Spark Streaming, MLlib (Machine Learning Library) và GraphX (xử lý đồ thị). Điều này làm cho nó thích hợp cho nhiều loại công việc khác nhau, từ xử lý batch đến xử lý dữ liệu trực tuyến và machine learning. 4) Xử lý dữ liệu có cấu trúc và không có cấu trúc: Spark có thể xử lý cả dữ liệu có cấu trúc và không có cấu trúc, làm cho nó phù hợp cho nhiều loại dữ liệu khác nhau.

Tuy nhiên, Apache Spark cũng có những nhược điểm như: 1) Yêu cầu tài nguyên hệ thống cao: Spark yêu cầu nhiều tài nguyên hệ thống, đặc biệt là bộ nhớ, để có thể tận dụng được ưu điểm của xử lý in-memory. Điều này có thể tạo ra áp lực đối với cụm máy tính. 2) Khó làm chủ công nghệ: Việc học sử dụng Spark có thể đòi hỏi một khối lượng kiến thức lớn, đặc biệt là khi sử dụng các tính năng phức tạp như Spark Streaming hoặc machine learning trên Spark. 3) Chưa thích hợp cho tất cả các tình huống: Mặc dù Spark phù hợp cho nhiều loại công việc xử lý dữ liệu, nhưng không phải lúc nào nó cũng là lựa chọn tốt nhất.

Các nền tảng trên cung cấp nhiều giải pháp đầy triển vọng. Tuy nhiên, chúng có những nhược điểm sau đây: 1) Chưa đáp ứng được nhu cầu vừa triển khai ứng dụng phân tán nói chung vừa huấn luyện *DNN* phân tán. 2) Các nền tảng này khá phức tạp và khó làm chủ. 3) Không cho phép tổ chức dữ liệu và xử lý ở các worker nodes một cách linh hoạt.

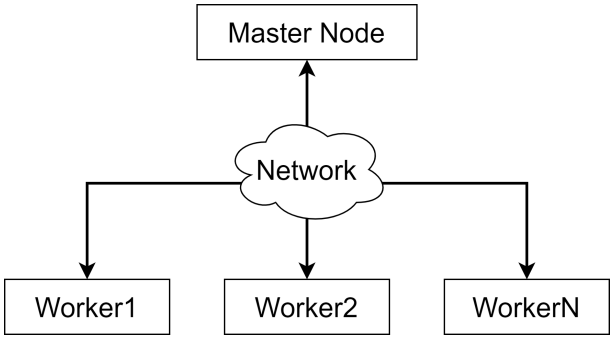
Chính vì vậy, để tạo điều kiện thuận lợi cho việc nghiên cứu và triển khai ứng dụng, chúng tôi phát triển nền tảng xử lý phân tán mới vừa có thể triển khai ứng dụng phân tán vừa có thể huấn luyện phân tán DNN, đơn giản, dễ sử dụng, cho phép tổ chức dữ liệu và xử lý ở các worker nodes một cách linh hoạt theo nhu cầu cụ thể. Bên cạnh đó, chúng tôi cũng 1) Phân tích những trở ngại khi triển khai hệ thống huấn luyện phân tán DNN trên *Apache Spark* và đề xuất các giải pháp khắc phục chúng; 2) Trình bày chi tiết quá trình phát triển nền tảng huấn luyện phân tán DNN.

3.4 Cơ sở lý thuyết

3.4.1 Cụm máy tính (Computer Cluster)

Cụm máy tính là nhiều máy tính được kết nối mạng và chúng hoạt động giống như một thực thể duy nhất [18, 19]. Có nhiều kiến trúc cụm máy tính, Hình 3.1 là ví dụ về một cụm tính toán tiêu biểu. Mỗi máy tính tham gia vào cụm được gọi là một node. Trong đó:

- Master node: là máy tính chạy chương trình chính, gửi yêu cầu đến các worker node để thực thi song song và thu thập kết quả.
- Worker node: là máy tính tham gia xử lý các yêu cầu của master node.



Hình 3.1. Một cụm máy tính

Cụm máy tính cung cấp giải pháp xử lý phân tán để giải quyết các vấn đề phức tạp bằng cách chia vấn đề lớn thành nhiều phần nhỏ, mỗi phần nhỏ sẽ được xử lý bởi một worker node. Cụm máy tính có rất nhiều ưu điểm. Sau đây là những ưu điểm tiêu biểu:

- Hiệu quả về chi phí: Cùng một mục đích, thay vì triển khai một máy tính lớn đắt tiền, triển khai một cụm máy tính sẽ kinh tế hơn nhiều.
- Tốc độ xử lý: Các cụm máy tính cung cấp tốc độ xử lý tương tự như tốc độ xử lý của máy tính lớn hay siêu máy tính.
- Tính sẵn sàng cao: Khi một worker node bị lỗi thì cụm máy tính vẫn hoạt động bình thường, dù hiệu suất có giảm đi.
- Khả năng mở rộng: Dễ dàng nâng cấp phần cứng (mở rộng dọc) hoặc thêm/bớt worker node (mở rộng ngang) trong cụm máy tính tùy theo nhu cầu thực tế.
- Tính linh hoạt: Dễ dàng thay đổi số lượng worker node tham gia xử lý một vấn đề cụ thể thông qua việc thay đổi cấu hình.

3.4.2 Cụm Spark (Spark Cluster)

Có nhiều nền tảng xử lý phân tán như: Hadoop, Apache Spark, Apache Storm, Samza, Flink [15, 20]. Trong đó, Apache Spark là nền tảng xử lý phân tán dữ liêu lớn mạnh mẽ và được dùng phổ biến nhất nhờ các tính năng sau: *1) Tốc độ xử lý được ví nhanh như chớp; 2) Dễ sử dụng và linh hoạt; 3) Cung cấp hỗ trợ cho các phân tích phức tạp; 4) Xử lý luồng thời gian thực.*

Để triển khai hệ thống xử lý phân tán trên nền tảng Apache Spark, chúng ta cần phải xây dựng Spark Cluster. Spark Cluster là cụm máy tính mà mỗi máy tính được cài đặt phần mềm Apache Spark để điều khiển hoạt động trong cụm. Do Apache Spark không có hệ thống quản lý file riêng, nên Spark Cluster thường được kết hợp với một hệ thống file phân tán như HDFS của Hadoop (Hadoop Distributed File System - HDFS) hay S3 của Amazon.

Apache Spark là nền tảng tính toán trong bộ nhớ nên các máy trong cụm cần có nhiều bộ nhớ. Hơn nữa, để tạo sự cân bằng và hiệu quả trong xử lý phân tán, cấu hình phần cứng của các worker nodes nên giống nhau. Master node là nơi thu gom tất cả kết quả xử lý từ các worker nodes, thực hiện các xử lý tổng hợp để tạo ra kết quả cuối cùng. Các kết quả gửi về từ worker nodes đôi khi rất lớn, đòi hỏi cấu hình phần cứng của master node phải mạnh hơn các worker nodes.

3.4.3 Học sâu (Deep Learning - DL)

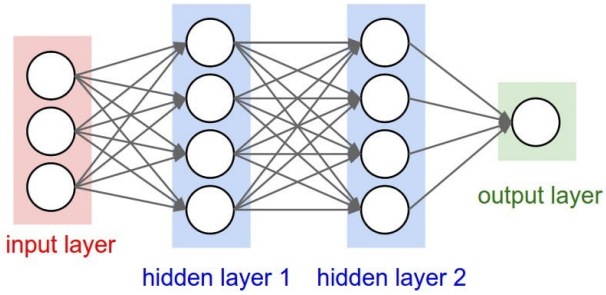
DL là là tập con của ML [21] sử dụng nhiều lớp nơ-ron để trích xuất dần dần các đặc trưng ở mức cao hơn từ đầu vào thô. Ví dụ, trong xử lý hình ảnh, các lớp nơ-ron thấp hơn có thể xác định các cạnh, trong khi các lớp nơ-ron cao hơn có thể xác định các đặc trưng khác như chữ số, chữ cái hoặc khuôn mặt. Từ "sâu"trong "học sâu"dùng để chỉ số lớp nơ-ron mà dữ liệu được truyền qua đó.

DL khác với ML truyền thống ở chỗ nó có thể tự động phát hiện các đặc trưng từ dữ liệu mà không cần sự can thiệp của con người. Nhờ kiến trúc linh hoạt và tiên tiến, DL tạo được những đột phá trong lĩnh vực trí tuệ nhân tạo (Artificial Intelligence - AI). Chương trình cờ vây AlphaGo đánh bại kỳ thủ số một thế giới năm 2017, xe tự lái, trợ lý ảo thông minh... là những minh chứng cho những đột phá này. DL đang trở thành một trong những lĩnh vực nóng nhất trong khoa học máy tính. Chỉ trong vài năm, DL đã thúc đẩy tiến bộ trong nhiều lĩnh vực như phát hiện đối tượng, dịch máy, nhận dạng giọng nói... những vấn đề từng rất khó khăn với các nhà nghiên cứu trí tuệ nhân tạo [22, 23].

3.4.4 Mạng nơ-ron sâu (Deep Neural Network - DNN)

DNN, như trong Hình 3.2, là một mạng nơ-ron nhân tạo có nhiều lớp nơ-ron ẩn giữa các lớp đầu vào và đầu ra dùng để mô hình hóa các mối quan hệ phi tuyến tính phức tạp.

Mục đích chính của DNN là nhận một tập hợp các đầu vào, thực hiện các phép tính phức tạp dần và tạo đầu ra để giải quyết các vấn đề thực tế như phân loại hình ảnh, phân loại văn bản, nhận dạng đối tượng...



Hình 3.2. Một mạng nơ-ron sâu

3.4.5 Huấn luyện DNN

Việc huấn luyện DNN tốn rất nhiều thời gian do những xử lý phức tạp bên trong. Tập dữ liệu huấn luyện càng lớn thì càng tốn nhiều thời gian huấn luyện. Để giải quyết vấn đề này, chúng ta có hai giải pháp: scale-up (vertical scaling) và scale-out (horizontal scaling) hệ thống [24]:

- Scale-up: một máy tính được bổ sung hoặc nâng cấp tài nguyên (CPU, GPU, TPU, memory, storage, network) để đạt được hiệu suất mong muốn. Scale-up gắn liền với kiến trúc cục bộ nên nó có những ưu điểm và khuyết điểm của kiến trúc cục bộ. Ưu điểm dễ thấy nhất là chi phí trao đổi dữ liệu giữa các luồng xử lý (thread) là không đáng kể. Tuy nhiên, nhược điểm là khả năng nâng cấp tài nguyên bị giới hạn. Ví dụ, việc gắn thêm CPU hoặc bộ nhớ là có giới hạn, không phải gắn thêm bao nhiêu cũng được.
- Scale-out: hệ thống tính toán được bổ sung máy tính để đạt được hiệu suất mong muốn. Scale-out gắn liền với kiến trúc phân tán nên nó có những ưu điểm và khuyết điểm của kiến trúc phân tán. Ưu điểm là việc nâng cấp tài nguyên hầu như không bị hạn chế. Ví dụ, Spark Cluster lớn nhất được biết có tới 8000 nodes [25]. Tuy nhiên, nhược điểm là chi phí liên lạc giữa các worker nodes trong cụm khá lớn, ảnh hưởng đáng kể đến hiệu quả xử lý của toàn bộ hệ thống.

Có hai cách để huấn luyện mô hình DL: huấn luyện cục bộ và huấn luyện phân tán. Nếu tập dữ liệu huấn luyện nhỏ (vừa với bộ nhớ của một máy tính) và mô hình DL đơn giản (ít nơ-ron), chúng ta nên sử dụng huấn luyện cục bộ. Ngược lại, chúng ta nên chọn hình thức huấn luyện phân tán. Tuy nhiên, huấn luyện phân tán phức tạp hơn và phải chịu thêm chi phí liên lạc giữa các máy trong cụm xử lý.

Độ chính xác của mô hình có thể tăng lên theo số lượng mẫu huấn luyện, số lượng tham số của mô hình, hoặc cả hai. Tuy nhiên, việc huấn luyện các mô hình lớn rất phức tạp và tốn nhiều thời gian khi được huấn luyện trên một máy, ngay cả khi có hỗ trợ đa luồng. Điều này yêu cầu mở rộng việc huấn luyện các mô hình trên nhiều máy tính được kết nối theo cách phân tán.

Để phân tích tầm quan trọng của phương pháp huấn luyện phân tán, trong phần này chúng tôi sẽ trình bày những nội dung sau:

- Huấn luyện DNN trong môi trường cục bộ.

- Huấn luyện DNN trong môi trường phân tán.

3.4.5.1 Huấn luyện DNN trong môi trường cục bộ

Hiện tại, có rất nhiều nền tảng để triển khai DL. Năm nền tảng tốt nhất có thể kể là TensorFlow, Keras, PyTorch, Apache MXNet và Microsoft Cognitive Toolkit [26]. Trong đó, Keras đơn giản và dễ sử dụng nhất. Đặc điểm nổi bật của Keras bao gồm: *1) API dễ hiểu và nhất quán; 2) Có thể tích hợp với nhiều nền tảng DL bên dưới như: TensorFlow, Theano và CNTK; 3) Hỗ trợ huấn luyện phân tán và song song trên nhiều GPU.* Bên cạnh đó, TensorFlow là nền tảng tốt nhất và được ưa thích nhất hiện nay. Đặc điểm nổi bật của Tensorflow gồm: *1) Hỗ trợ nhiều GPU mạnh mẽ; 2) Trực quan hóa đồ thị tính toán; 3) Tài liệu và hỗ trợ cộng đồng tuyệt vời.* Tuy nhiên, Tensorflow được xem là nền tảng phức tạp, khó sử dụng. Chính vì vậy, Keras đã được chọn làm API cấp cao của Tensorflow 2 nhằm dễ tiếp cận và đạt hiệu quả cao khi giải quyết các vấn đề DL. Từ những phân tích trên, chúng tôi chọn Keras làm nền tảng huấn luyện mô hình DL trong môi trường cục bộ trong nghiên cứu này.

Thuật toán 3.1 trình bày khái quát quá trình huấn luyện mô hình DL ở chế độ cục bộ bằng Keras. Đầu tiên, hai tập dữ liệu huấn luyện và kiểm thử được tải vào hệ thống. Sau đó, mô hình DL sẽ được tạo theo nhu cầu. Trước khi huấn luyện, cần cấu hình quá trình học bằng phương thức *model.compile()*. Tiếp theo, gọi phương thức *model.fit()* để tiến hành quá trình huấn luyện mô hình bằng tập dữ liệu huấn luyện. Cuối cùng, đánh giá mô hình bằng phương thức *model.evaluate()* với tập dữ liệu kiểm thử. Lưu ý, các tham số trong thuật toán này có thể thay đổi tùy theo nhu cầu thực tế.

Thuật toán 3.1. Huấn luyện mô hình DL trong môi trường cục bộ

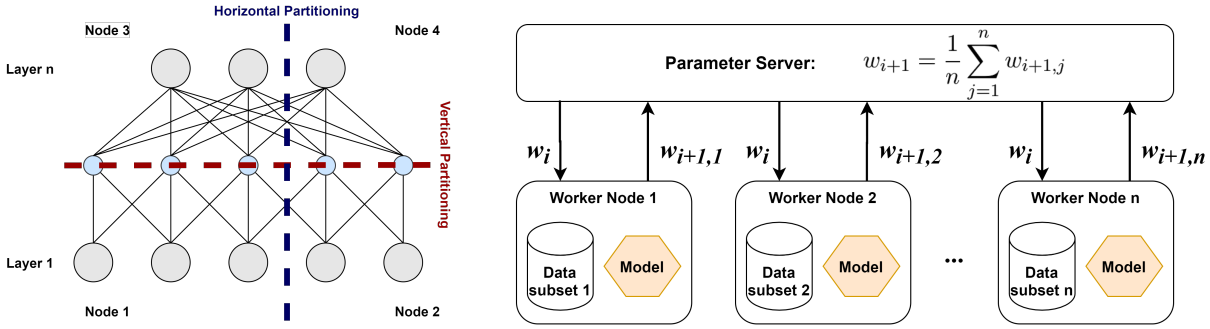
```
1  (x_train, y_train), (x_test, y_test) ← load_dataset()
2  model ← create_model()
3  model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
4  model.fit(x_train, y_train, epochs=10, batch_size=32)
5  loss_and_metrics = model.evaluate(x_test, y_test, batch_size=32)
```

3.4.5.2 Huấn luyện mô hình DL trong môi trường phân tán

Có hai cách huấn luyện phân tán chính: song song hóa mô hình (model parallelism) và song song hóa dữ liệu (data parallelism) [27–29].

Trong song song hóa mô hình, như trong Hình 3.3a, mô hình được chia thành các phần khác nhau để có thể chạy đồng thời ở các máy khác nhau và mỗi phần sẽ chạy trên cùng một tập dữ liệu. Các worker nodes chỉ cần đồng bộ hóa các trọng số được dùng chung, thường là một lần cho mỗi bước lan truyền tiến hoặc lùi. Do phương pháp này khá phức tạp, chúng ta chỉ nên dùng khi mô hình lớn không vừa trong một máy. Vì tính chất phức tạp của phương pháp này và phạm vi giới hạn của chuyên đề nên trong nghiên cứu này chúng tôi chỉ tập trung vào phương pháp song song hóa dữ liệu.

Song song hóa dữ liệu, như trong Hình 3.3b, là phương pháp phổ biến nhất, dễ thực hiện nhất và phù hợp với hầu hết mọi trường hợp. Trong phương pháp này, dữ liệu được chia thành các phân vùng và được phân phối cho các worker nodes. Mô hình được tạo ở

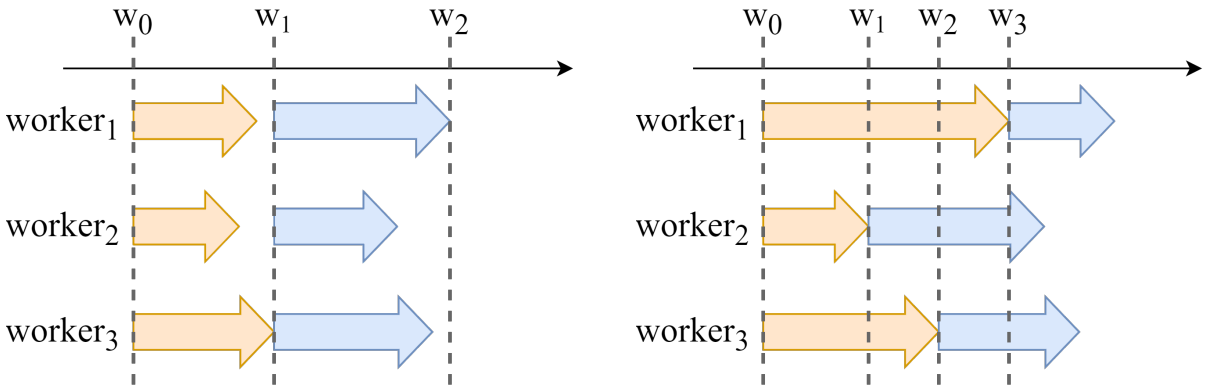


(a) Song song hóa mô hình (b) Song song hóa dữ liệu

Hình 3.3. Hai phương pháp huấn luyện phân tán DNN

master node, sau đó được gửi đến các worker nodes và được huấn luyện một cách độc lập bằng các phân vùng dữ liệu đã phân phối trước đó. Sau khi huấn luyện mô hình trong một khoảng thời gian có kiểm soát, các trọng số của mô hình được worker nodes gửi về master node và được tính trung bình để cập nhật tập trọng số cho mô hình ở master node.

Phương pháp song song hóa dữ liệu bao gồm gồm hai loại chính: 1) Huấn luyện đồng bộ (Hình 3.4a): tất cả các worker nodes đồng bộ với nhau trong quá trình huấn luyện; 2) Huấn luyện bất đồng bộ (Hình 3.4b): mỗi máy huấn luyện theo tốc độ riêng của nó.



(a) Huấn luyện đồng bộ (b) Huấn luyện bất đồng bộ

Hình 3.4. Huấn luyện phân tán DNN đồng bộ và bất đồng bộ

A. Huấn luyện DNN kiểu đồng bộ

Trong huấn luyện đồng bộ, hệ thống huấn luyện phải đợi tất cả worker nodes hoàn thành đợt huấn luyện, gradients của chúng được gửi về master node để cập nhật mô hình chính, master node sẽ gửi bộ trọng số mới đến tất cả worker nodes để cập nhật mô hình của chúng cùng một lúc. Phương pháp này có một nhược điểm là không tận dụng được hết khả năng của cụm tính toán do các worker nodes nhanh phải đợi các worker nodes chậm. Để hạn chế tối thiểu nhược điểm này, cấu hình của các worker nodes phải giống nhau.

Thuật toán 3.2 trình bày quá trình huấn luyện DNN kiểu đồng bộ. Hàm $master\_train()$

được thực thi trên master node, và hàm $worker_j\_train()$ được thực thi trên $worker_j$. Trên master node, đầu tiên, tập dữ liệu được tải vào hệ thống. Kế tiếp, master node tạo *model* và gửi đến các worker nodes. Trong vòng lặp huấn luyện với số epochs là M, tập dữ liệu được xáo trộn, được phân hoạch thành K partitions (K là số lượng worker node trong cụm máy tính), và partition P_j được gửi đến $worker_j$. Sau đó, bộ trọng số w được gửi đến tất cả worker nodes để cập nhật và huấn luyện trên các worker nodes. Khi tất cả các workers hoàn thành đợt huấn luyện, master node thu thập các gradients và cập nhật mô hình chính. Nếu mô hình đã thỏa mãn điều kiện hội tụ, master node sẽ gửi tín hiệu *STOP* đến các worker nodes để ngưng quá trình huấn luyện. Ở $worker_j$, nó đợi nhận *model* từ master node để tiến hành quá trình huấn luyện. Trong vòng lặp huấn luyện, $worker_j$ đợi nhận partition P_j và bộ trọng số w_j từ master node, cập nhật $model_j$, và bắt đầu quá trình huấn luyện. Sau khi huấn luyện xong N epochs, gradients g_j được gửi về master node để cập nhật mô hình chính. Nếu nhận được tín hiệu *STOP* từ master node, $worker_j$ sẽ kết thúc quá trình huấn luyện, ngược lại nó sẽ tiếp tục đợt huấn luyện kế tiếp cho đến khi đủ M epochs.

Thuật toán 3.2. Huấn luyện phân tán DNN kiểu đồng bộ trên K worker nodes

```
1  function master_train() do
2      dataset ← load_dataset()
3      model ← create_model()
4      broadcast_model(model)
5      w ← model.get_weights()
6      i ← 0
7      while i < M do
8          dataset ← shuffle(dataset)
9          ∪_{j=1}^{K} P_j ← partition(dataset)
10         send_data(∪_{j=1}^{K} P_j)
11         broadcast_weights(w)
12         g ← await collect g_j from all workers
13         w ← w − (1/K) × Σ_{j=1}^{K} g_j
14         model.set_weights(w)
15         if is_convergent(model) then
16             broadcast_signal(STOP)
17             break
18         end if
19         i ← i + N
20     end while
21 end function
22
23 function worker_j_train() do
24     model_j ← await receive_model()
25     i ← 0
26     while i < M do
27         P_j ← await receive_data()
28         w_j ← await receive_weights()
29         model_j.set_weights(w_j)
30         model_j.fit(P_j, epochs = N, batch_size = BS)
```

```
31          g_j ← model_j.get_gradients()
32          send g_j to the master node
33          i ← i + N
34          if has_signal(STOP) then
35              break
36          end if
37      end while
38  end function
```

B. Huấn luyện DNN kiểu bất đồng bộ

Trong phương pháp huấn luyện bất đồng bộ, khả năng của cụm máy tính được tận dụng tối đa do không có hiện tượng các worker node đợi nhau. Trong phương pháp này, khi một worker node hoàn thành đợt huấn luyện, gradients của nó được gửi về master node để cập nhật mô hình chính, bộ trọng số mới sẽ được gửi trở lại worker node đó để tiến hành đợt huấn luyện kế tiếp. Ngoài việc dùng gradients cập nhật mô hình chung ở master node, các worker nodes hoạt động hoàn toàn độc lập với nhau. Điều này dẫn đến một hệ quả là các worker nodes khác không được cập nhật kịp thời bộ trọng số mới, tiếp tục sử dụng bộ trọng số cũ đến hết đợt huấn luyện và có thể đi chệch mục tiêu. Thuộc tính này được gọi là *staleness*. Một cách để giảm bớt hiện tượng này là thay đổi tốc độ học tùy thuộc vào thuộc tính *staleness* - một tốc độ học nhỏ hơn đối với các worker nodes cũ hơn. Bằng cách này, khi worker node đi chệch hướng, mức độ chệch hướng tương đối nhỏ.

Thuật toán 3.3 trình bày quá trình huấn luyện DNN kiểu bất đồng bộ. Tương tự như kiểu huấn luyện đồng bộ, đầu tiên master node cũng tải dữ liệu, tạo mô hình, gửi mô hình đến các worker nodes. Do các worker nodes hoạt động độc lập nhau, master node sẽ tạo một thread để điều khiển việc huấn luyện trên một worker node. Xem xét $thread_j$, đầu tiên, tập dữ liệu được xáo trộn, được phân hoạch, và phân hoạch P_j được gửi đến $worker_j$. Sau đó, master node gửi bộ trọng số đến $worker_j$ để huấn luyện, và đợi nhận gradients kết quả để cập nhật mô hình chính. Do mô hình chính được dùng chung bởi K threads nên trước khi cập nhật, $thread_j$ phải khóa độc quyền mô hình chính để tránh sự xung đột giữa các threads. Hoạt động huấn luyện ở các worker nodes diễn ra giống như trong phương pháp huấn luyện đồng bộ.

Thuật toán 3.3. Huấn luyện phân tán DNN kiểu bất đồng bộ trên K worker nodes

```
1   function master_train() do
2       dataset ← load_dataset()
3       model ← create_model()
4       broadcast_model(model)
5       w ← model.get_weights()
6       for j ← 1 to K do
7           w_j ← w
8           run thread(j) do
9               i ← 0
10              while i < M do
11                  dataset ← shuffle(dataset)
12                  ∪_{j=1}^{K} P_j ← partition(dataset)
```

```
13                  send_data(P_j) to worker_j
14                  send_weights(w_j) to worker_j
15                  g_j ← await collect g_j from worker_j
16                  lock(model) do
17                      w_j ← model.get_weights()
18                      w_j ← w_j − (1/K) × g_j
19                      model.set_weights(w_j)
20                      if is_convergent(model) then
21                          broadcast_signal(STOP)
22                          break
23                      end if
24                  end lock
25                  i ← i + N
26              end while
27          end run thread
28      end for
29  end function
30
31  function worker_j_train() do
32      model_j ← await receive_model()
33      i ← 0
34      while i < M do
35          P_j ← await receive_data()
36          w_j ← await receive_weights()
37          model_j.set_weights(w_j)
38          model_j.fit(P_j, epochs = N, batch_size = BS)
39          g_j ← model_j.get_gradients()
40          send g_j to the master node
41          i ← i + N
42          if has_signal(STOP) then
43              break
44          end if
45      end while
46  end function
```

3.5 Phương pháp luận

3.5.1 Chọn phương pháp lưu trữ dữ liệu lớn

Mạng thông tin lớn là một loại dữ liệu lớn (Big Data) không thể lưu trữ trên một máy tính nên nó phải được lưu trữ phân tán. Lưu trữ dữ liệu phân tán là một cách lưu trữ dữ liệu trên nhiều máy tính thay vì một máy tính duy nhất. Điều này có thể mang lại nhiều lợi ích, chẳng hạn như khả năng mở rộng, độ tin cậy, và hiệu suất.

Có nhiều phương pháp lưu trữ dữ liệu lớn khác nhau, mỗi phương pháp có ưu và nhược điểm riêng. Dưới đây là một số phương pháp lưu trữ dữ liệu lớn phổ biến:

- Hệ thống tập tin phân tán (Distributed File System - DFS): Hệ thống tập tin phân tán là một hệ thống lưu trữ dữ liệu trong đó dữ liệu được lưu trữ trên nhiều máy

tính. HDFS (Hadoop Distributed File System) là một hệ thống tập tin phân tán phổ biến được phát triển bởi Apache Hadoop. HDFS được sử dụng rộng rãi cho các ứng dụng dữ liệu lớn, chẳng hạn như lưu trữ dữ liệu thô, xử lý dữ liệu lớn, và phân tích dữ liệu.

- Hệ thống cơ sở dữ liệu NoSQL: Hệ thống cơ sở dữ liệu NoSQL là một loại hệ thống cơ sở dữ liệu không sử dụng mô hình quan hệ truyền thống. Các hệ thống cơ sở dữ liệu NoSQL có thể cung cấp khả năng mở rộng và hiệu suất cao hơn các hệ thống cơ sở dữ liệu quan hệ. Một số hệ thống cơ sở dữ liệu NoSQL phổ biến bao gồm Cassandra, MongoDB, và Redis.
- Hệ thống lưu trữ đám mây: Hệ thống lưu trữ đám mây là một loại hệ thống lưu trữ dữ liệu được cung cấp dưới dạng dịch vụ. Các hệ thống lưu trữ đám mây có thể cung cấp khả năng mở rộng, độ tin cậy, và hiệu suất cao. Một số nhà cung cấp dịch vụ lưu trữ đám mây phổ biến bao gồm Amazon S3, Microsoft Azure Blob Storage và Google Cloud Storage.
- Hệ thống lưu trữ phân tán chia ô: Hệ thống lưu trữ phân tán chia ô là một loại hệ thống lưu trữ dữ liệu trong đó dữ liệu được chia thành các ô. Các ô có thể được phân tán trên nhiều máy tính. Các hệ thống lưu trữ phân tán chia ô có thể cung cấp khả năng mở rộng và hiệu suất cao. Một số hệ thống lưu trữ phân tán chia ô phổ biến bao gồm Cassandra, HBase, và MongoDB.

Để lưu trữ mạng thông tin lớn, chúng tôi chọn HDFS vì các lý do sau:

- Khả năng mở rộng: HDFS có thể được mở rộng dễ dàng bằng cách thêm các máy tính mới vào cụm. Điều này làm cho HDFS trở thành một lựa chọn tốt cho các ứng dụng cần lưu trữ một lượng lớn dữ liệu.
- Độ tin cậy: HDFS sử dụng một mô hình lưu trữ dữ liệu dư thừa trên nhiều máy tính. Điều này giúp bảo vệ dữ liệu khỏi bị mất trong trường hợp một máy tính bị lỗi.
- Hiệu suất: HDFS được thiết kế để tối ưu hóa hiệu suất cho các ứng dụng dữ liệu lớn. HDFS sử dụng một mô hình phân tán để phân phối tải truy cập trên nhiều máy chủ.
- Khả năng tương thích: HDFS tương thích với các ứng dụng và công cụ tiêu chuẩn. Điều này làm cho HDFS dễ dàng sử dụng và quản lý.
- Tính ổn định: HDFS đã được sử dụng trong sản xuất trong nhiều năm. HDFS là một hệ thống ổn định và đáng tin cậy.

Nhìn chung, HDFS là một hệ thống tệp phân tán mạnh mẽ và linh hoạt. HDFS là một lựa chọn tốt cho các ứng dụng cần lưu trữ một lượng lớn dữ liệu, có yêu cầu về độ mở rộng, độ tin cậy, và hiệu suất cao.

3.5.2 Nghiên cứu triển khai hệ thống huấn luyện DNN phân tán trên Apache Spark

Apache Spark là nền tảng xử lý phân tán chuyên dùng xử lý và phân tích dữ liêu lớn rất hiệu quả. Nhờ có nhiều ưu điểm, Apache Spark đã trở thành nền tảng xử lý dữ liêu lớn được dùng phổ biến nhất hiện nay. Tuy nhiên, khi triển khai ứng dụng huấn luyện DNN phân tán, Apache Spark cho thấy có nhiều nhược điểm, gây khó khăn trong việc triển khai ứng dụng và ảnh hưởng xấu đến toàn bộ hệ thống. Ví dụ 3.1 trình bày việc hiện thực Thuật toán 3.2 để huấn luyện DNN phân tán kiểu đồng bộ trên Apache Spark. Trong hiện thực này, chúng tôi kết hợp Keras với Apache Spark.

Ví dụ 3.1. Hiện thực Thuật toán 3.2 trên Apache Spark

```
def master_train():
    # Load the dataset
    (x_train, y_train), (x_test, y_test) = load_dataset()
    # Create the model
    model = create_model()
    # Broadcast the yaml string to workers for restoring the model
    bc_yaml = sc.broadcast(model.to_yaml())
    weights = model.get_weights()
    i = 0
    while(i < M):
        # Shuffle the training data
        permutation = np.random.permutation(len(x_train))
        x_train = x_train[permutation]
        y_train = y_train[permutation]
        # Build RDD from the training data and partition the RDD into K partitions
        rdd = build_rdd(sc, x_train, y_train, K)
        # Broadcast weights to workers
        bc_weights = sc.broadcast(weights)
        # Call worker_train() on workers and await to collect gradients from workers
        gradients = rdd.mapPartitions(worker_train).collect()
        # Update weights of the model at the master node
        for gradient in gradients:
            weighted_grad = divide(gradient, K)
            weights = subtract(weights, weighted_grad)
        model.set_weights(weights)
        i += N

def worker_train(data_iterator):
    # Restore the model from a yaml string
    model = model_from_yaml(bc_yaml.value)
    # Load x_train, y_train from the data_iterator
    x_train, y_train = load_partition(data_iterator)
    # Compile the model
    compile(model)
    # Get weights from the master node
    weights = bc_weights.value
```

```
37      # set weights to the model
38      model.set_weights(weights)
39      # Train the model
40      model.fit(x_train, y_train, epochs = N, batch_size = BS)
41      # calculate gradients
42      new_weights = model.get_weights()
43      gradients = subtract(weights, new_weights)
44      yield gradients
```

Trong hiện thực trên có ba vấn đề lớn, ảnh hưởng xấu đến hiệu suất của toàn bộ hệ thống: *1) Trong mỗi lần lặp, việc xáo trộn dữ liệu, phân hoạch lại dữ liệu và gửi các phân hoạch mới đến các worker nodes là xử lý tốn nhiều thời gian; 2) Tư duy lập trình bị phụ thuộc vào cấu trúc map/reduce, người lập trình không được tự do phát triển ý tưởng; 3) Hàm worker_train() được gửi từ master node đến các worker nodes nhiều lần, trong khi nội dung của hàm này không thay đổi, làm tăng lưu lượng trên mạng không cần thiết.* Dựa trên góc nhìn khi phát triển ứng dụng huấn luyện DNN phân tán, chúng tôi nhận thấy Apache Spark có các nhược điểm như sau:

- Khi huấn luyện DNN phân tán, việc xáo trộn dữ liệu, phân hoạch lại dữ liệu và gửi các phân hoạch mới đến các worker nodes là không thể tránh khỏi. Nhưng đây lại là những thao tác tốn nhiều thời gian xử lý trên Apache Spark. Điều này làm ảnh hưởng đến hiệu quả của toàn bộ hệ thống huấn luyện DNN phân tán.
- Cấu trúc *map/reduce* của Apache Spark được áp dụng rất hiệu quả trong xử lý dữ liêu lớn nhưng không phù hợp trong việc huấn luyện DNN phân tán vì việc lập trình bị gò bó trong cấu trúc *map/reduce*, trói buộc tư duy lập trình, dẫn đến việc triển khai ứng dụng huấn luyện DNN kém linh hoạt và không hiệu quả.
- Do phụ thuộc vào cấu trúc *map/reduce*, hàm *worker_train(data_iterator)* chỉ nhận một tham số duy nhất là *data_iterator* (là 1 data partition). Người lập trình không thể truyền thông tin khác cho hàm *worker_train()* khi cần. Dùng biện pháp thay thế thì quá phiền phức. Điều này làm khó triển khai ứng dụng.
- Cơ chế chia sẻ dữ liệu giữa master node và worker nodes rất hạn chế thông qua *broadcast và accumulated variables*. Trong khi broadcast variables là biến chỉ đọc, còn accumulated variables là biến chỉ cộng.
- Người sử dụng không thể chủ động thêm/bớt biến hay phương thức vào worker node.
- Spark không hỗ trợ liên lạc bất đồng bộ giữa master node và các worker nodes. Do đó, để triển khai ứng dụng huấn luyện DNN kiểu bất đồng bộ, người phát triển ứng dụng phải tự bổ sung cơ chế liên lạc bất đồng bộ vào Apache Spark hoặc sử dụng một framework có hỗ trợ.

3.5.3 Xây dựng DDLF

Để tránh những nhược điểm trên của Apache Spark khi phát triển ứng dụng huấn luyện DNN phân tán, chúng tôi cân nhắc 2 giải pháp: 1) Sử dụng framework có sẵn; 2)

Xây dựng framework mới. Hiện tại, có nhiều framework hỗ trợ huấn luyện DNN phân tán như: Distributed TensorFlow[7], Elephas, Horovod, TensorFlowOnSpark, BigDL, PyTorch, Ray. Dùng framework có sẵn giúp phát triển nhanh ứng dụng nhưng mặt trái là phải chấp nhận những khuyết điểm nội tại của nó và khó cải tiến, dẫn đến kém linh hoạt. Chúng tôi chọn giải pháp xây dựng framework mới vì các lý do sau đây:

- Không phụ thuộc vào framework có sẵn: Thật là khó chịu khi dùng framework bị lỗi mà không biết rõ nguyên nhân. Hơn nữa, biết nguyên nhân lỗi nhưng không thể khắc phục do những khuyết điểm thiết kế và kỹ thuật áp dụng không thay đổi được.
- Làm chủ công nghệ: Framework do chúng tôi phát triển nên chúng tôi hiểu rõ các kỹ thuật được áp dụng, hiểu rõ bản chất hoạt động bên trong của nó, làm chủ những kỹ thuật then chốt.
- Làm sâu sắc nội dung nghiên cứu: Khi phát triển framework, chúng tôi phải tiến hành nhiều nghiên cứu, khảo sát, làm sâu sắc thêm nội dung đang nghiên cứu.
- Linh hoạt: Chúng tôi không bị gò bó vào những cấu trúc lập trình có sẵn, chẳng hạn như *map/reduce*, giúp giải phóng tư duy lập trình, tự do, thoải mái phát triển ý tưởng.
- Khả năng mở rộng: hiểu rõ tường tận kỹ thuật giúp dễ dàng cải tiến và mở rộng framework.
- Khắc phục những hạn chế đã gặp: thông qua kinh nghiệm sử dụng các framework có sẵn, chúng tôi có thể cung cấp giải pháp cho các vấn đề đã gặp.

3.5.3.1 Kiến trúc của DDLF

Kiến trúc của DDLF là cụm máy tính gồm một master node và nhiều worker nodes như Hình 3.5. Master node sẽ điều khiển các worker nodes để xử lý tác vụ theo kiểu phân tán. Interface IWorker đặc tả chức năng của worker nodes. Các worker nodes được hiện thực bằng class Worker. Mỗi master node bao gồm 3 thành phần chính: 1) ProxyWorker: đại diện cho worker node; 2) Cluster: biểu diễn cluster; 3) App: biểu diễn một ứng dụng xử lý phân tán. Lớp *Request* biểu diễn một yêu cầu Remote Procedure Call (RPC) được gửi từ lớp master node đến các worker nodes.

3.5.3.2 Lớp Request

Lớp Request biểu diễn một yêu cầu RPC. Bảng 3.1 mô tả các thuộc tính của lớp Request.

Bảng 3.1. Các thuộc tính của lớp Request

#	Thuộc Tính	Mô Tả
1.	command	Chuỗi ký tự được dùng để xác định yêu cầu.
2.	args	Tập hợp các tham số loại *Non Keyword Arguments* cần khi xử lý yêu cầu.

#	Thuộc Tính	Mô Tả
3.	kwargs	Tập hợp các tham số loại *Keyword Arguments* cần khi xử lý yêu cầu.

Ví dụ, để gửi phương thức *add()* đến các worker node và thực thi, master node phải tạo đối tượng của lớp *Request* như trong Ví dụ 3.2. Trong đó, thuộc tính *command = 'run_method'* xác định yêu cầu là *"gửi phương thức đến worker node để thực thi"*; thuộc tính *args = ['add']* xác định phương thức được gửi đến worker node là phương thức *add()*; và thuộc tính *kwargs = {a=20, b=60}* cung cấp các tham số cho phương thức *add()* khi thực thi.

Ví dụ 3.2. Ví dụ tạo yêu cầu *"gửi phương thức add() đến worker node để thực thi"*

```
1  def add(a, b):
2      return a + b
3
4  request = Request('run_method', add, a=20, b=60)
```

3.5.3.3 Interface IWorker

Interface IWorker đặc tả chức năng cho các worker nodes và được dùng làm giao diện liên lạc giữa master node và các worker nodes. Interface này được hiện thực bởi lớp Worker và lớp ProxyWorker. Bảng 3.2 mô tả các chức năng chính của interface IWorker.

Bảng 3.2. Các phương thức của interface IWorker

#	Phương Thức	Mô Tả
1.	async def connect(self)	Mở kết nối TCP (Transmission Control Protocol) giữa master node và worker node.
2.	async def close(self)	Đóng kết nối.
3.	async def load_cifar10(self)	Tải dataset CIFAR10.
4.	async def load_mnist(self)	Tải dataset MNIST.
5.	async def add_method(self, method_code, method_name)	Thêm một phương thức vào worker node.
6.	async def remove_method(self, method_name)	Xóa một phương thức ở worker node.
7.	async def run(self, method_name, **kwargs)	Thực thi một phương thức ở worker node.
8.	async def run_code(self, code)	Gửi code đến worker node và thực thi đoạn code đó ở worker node.

#	Phương Thức	Mô Tả
9.	async def run_method(self, method_code, method_name, **kwargs)	Gửi một phương thức đến worker node và thực thi phương thức đó ở worker node.
10.	async def shutdown(self)	shut down the worker node.

3.5.3.4 Lớp Worker

Lớp Worker hiện thực interface IWorker và định nghĩa các phương thức được thực thi ở một worker node. Ngoài các phương thức được đặc tả trong interface IWorker, lớp Worker có định nghĩa thêm một số phương thức xử lý đặc thù được mô tả trong Bảng 3.3.

Bảng 3.3. Các phương thức bổ sung của lớp Worker

#	Phương Thức	Mô Tả
1.	def start(self)	Khởi động worker node.
2.	async def control(self, reader: StreamReader, writer: StreamWriter)	Thực hiện vòng lặp: nhận yêu cầu từ master node ⟶ xử lý yêu cầu ⟶ gửi kết quả về master node.
3.	async def handle(self, req: Request)	Xử lý một yêu cầu của master node.

3.5.3.5 Lớp ProxyWorker

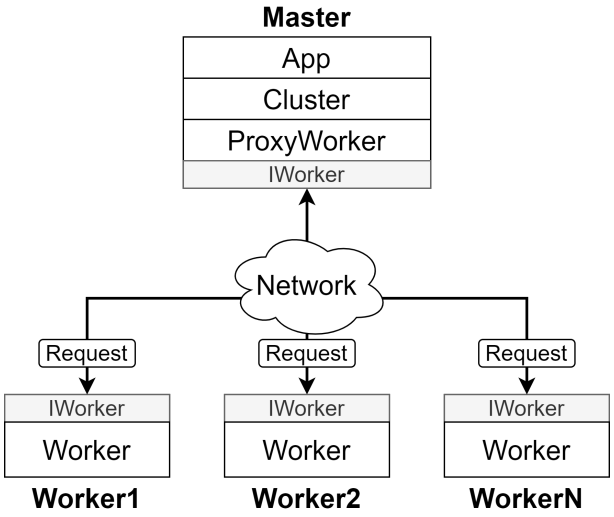
Một đối tượng của lớp *ProxyWorker* đại diện cho một worker node ở phía master node. Do đó lớp *Worker* và *ProxyWorker* cùng hiện thực một interface chung là *IWorker*. Ngoài những phương thức được đặc tả trong interface *IWorker*, lớp *ProxyWorker* cũng bổ sung những phương thức riêng của nó như trong Bảng 3.4.

Bảng 3.4. Các phương thức bổ sung của lớp ProxyWorker

#	Phương Thức	Mô Tả
1.	async def connect(self)	Tạo kết nối với worker node.
2.	async def rpc(self, req)	Thực hiện gọi phương thức từ xa.

3.5.3.6 Lớp Cluster

Lớp *Cluster* biểu diễn một cluster gồm nhiều worker nodes. Lớp này cung cấp các chức năng mà cluster đảm nhận. Các phương thức của lớp *Cluster* tương tự như các phương thức được đặc tả trong interface *IWorker* nhưng khác nhau ở chỗ: các phương thức trong interface *IWorker* đặc tả các chức năng của một worker node, còn các phương thức của lớp *Cluster* đặc tả các chức năng của một cluster. Bảng 3.5 mô tả các phương thức quan trọng của lớp *Cluster*.



Hình 3.5. Kiến trúc của DDLF

Bảng 3.5. Các phương thức quan trọng của lớp Cluster

#	**Phương Thức**	**Mô Tả**
1.	async def connect(self)	Tạo kết nối giữa master node với các worker nodes trong cluster.
2.	async def close(self)	Đóng kết nối giữa master node với các worker nodes trong cluster.
3.	async def add_method(self, method)	Thêm một phương thức vào các worker nodes trong cluster.
4.	async def remove_method(self, method)	Xóa một phương thức khỏi các worker nodes trong cluster.
5.	async def run(self, method, **kwargs)	Thực thi một phương thức ở các worker nodes trong cluster.
6.	async def run_at(self, worker, method, **kwargs)	Thực thi một phương thức ở một worker node trong cluster.
7.	async def run_code(self, code)	Gửi một đoạn code đến các worker nodes trong cluster và thực thi đoạn code đó ở các worker nodes.
8.	async def run_method(self, method, **kwargs)	Gửi một phương thức đến các worker nodes trong cluster và thực thi phương thức đó ở các worker nodes.
9.	async def shutdown(self)	shutdown các worker nodes trong cluster, hay nói cách khác là shutdown cluster.

3.5.4 Triển khai ứng dụng phân tán trên DDLF

3.5.4.1 Triển khai ứng dụng đơn giản

Để triển khai một ứng dụng phân tán đơn giản, chúng ta thực hiện các bước như trong Thuật toán 3.4.

Thuật toán 3.4. Triển khai ứng dụng phân tán đơn giản trên DDLF

```
1  Read the cluster configuration
2  Create an object of the Cluster class
3  Connect the master node to the cluster
4  Perform distributed processing on the cluster
5  Close the connection between the master node and the cluster
```

Ví dụ 3.3 trình bày một ứng dụng phân tán đơn giản tính biểu thức $a+b-c$. Đầu tiên, chúng ta định nghĩa phương thức *calculate()* để tính biểu thức trên. Trong hàm *main()*, gọi hàm *read_config()* để đọc thông tin cấu hình của cụm máy tính. Biến *hosts* chứa tên các máy tính làm worker node, biến *ports* chứa các ports mà các worker node lắng nghe các kết nối dựa trên giao thức TCP/IP. Sau đó, chúng ta tạo đối tượng *cluster* của lớp *Cluster* dựa vào danh sách worker nodes và ports. Đối tượng *cluster* này đại diện cho cluster xử lý phân tán. Kế tiếp, gọi phương thức *cluster.connect()* để kết nối master node với cluster. Để tiến hành xử lý phân tán, chúng ta gọi phương thức *cluster.run_method(calculate, a=10, b=8, c=2)* để gửi phương thức *calculate()* và các tham số *a=10, b=8, c=2* đến các worker node trong cluster để thực thi. Kết quả trả về từ các worker nodes trong cluster được chứa trong biến *results*. Cuối cùng, gọi phương thức *cluster.close()* để đóng kết nối giữa master node và cluster. Chương trình này khi chạy sẽ thực hiện phương thức *calculate(a=10, b=8, c=2)* trên cả 3 worker nodes nên kết quả sẽ là danh sách *[16, 16, 16]*.

Ví dụ 3.3. Một ứng dụng tính toán phân tán đơn giản

```
1  async def calculate(self, a, b, c):
2      return a + b - c
3
4  async def main():
5      hosts, ports = read_config()
6      cluster = Cluster(hosts, ports)
7      await cluster.connect()
8      results = await cluster.run_method(calculate, a=10, b=8, c=2)
9      print(f"Results: {results}") # Results: [16, 16, 16]
10     await cluster.close()
```

Để thực thi phương thức *calculate()* với những bộ tham số khác nhau trên các worker nodes, chúng ta sẽ thay dòng lệnh số 9 trong Ví dụ 3.3 bằng dòng lệnh sau:

```
results = await cluster.run_method(calculate,
dict(a=10, b=8, c=2),
dict(a=100, b=80, c=20),
```

`dict`*(a=1000, b=800, c=200))*

Dòng lệnh trên sẽ thực hiện phương thức *calculate(a=10, b=8, c=2)* trên *computer1*, thực hiện phương thức *calculate(a=100, b=80, c=20)* trên *computer2*, và thực hiện phương thức *calculate(a=1000, b=800, c=200)* trên *computer3*. Kết quả sẽ là *[16, 160, 1600]*.

Nếu chúng ta thay dòng lệnh số 9 trong Ví dụ 3.3 bằng dòng lệnh sau:

results = await cluster.run_ method(calculate,
`dict`*(a=10, b=8, c=2),*
`dict`*(a=100, b=80, c=20))*

Kết quả sẽ là *[16, 160]*. Ở dòng lệnh trên, chúng ta chỉ cung cấp hai bộ tham số *(a, b, c)* nên chỉ có hai worker nodes thực thi phương thức *calculate()*, do đó chỉ có hai kết quả trả về.

3.5.4.2 Triển khai ứng dụng huấn luyện DNN

Để triển khai ứng dụng huấn luyện DNN phân tán trên DDLF, chúng ta thực hiện các bước như trong Thuật toán 3.5.

Thuật toán 3.5. Triển khai ứng dụng huấn luyện DNN trên DDLF

1 *Read the cluster configuration*
2 *Create an object of the Cluster class*
3 *Connect the master node to the cluster*
4 *Download the dataset*
5 *Create the model*
6 *Train the model*
7 *Close the connection between the master node and the cluster*
8 *Evaluate the model*
9 *Save the model*

Để thuận tiện khi triển khai một ứng dụng phân tán, chúng tôi đã tạo interface *IApp* để đặc tả các chức năng cần thiết. Khi triển khai ứng dụng, người phát triển ứng dụng chỉ cần tạo lớp *App* (có thể đặt tên khác theo ý muốn) hiện thực interface *IApp*, và override một vài phương thức cần thiết. Bảng 3.6 mô tả các phương thức được đặc tả trong interface *IApp*.

Bảng 3.6. Các phương được đặc tả trong interface *IApp*

#	Phương Thức	Mô Tả
1.	async def connect(self)	Tạo kết nối giữa master node với cluster.
2.	async def close(self)	Đóng kết nối giữa master node với cluster.
3.	async def load_dataset(self)	Tải dataset.

#	Phương Thức	Mô Tả
4.	async def create_model(self)	Tạo mô hình.
5.	async def evaluate_model(self)	Đánh giá mô hình.
6.	async def save_model(self, path)	Lưu mô hình vào file.
7.	async def load_model(self, path)	Tải mô hình từ file.
8.	async def train(self, weights, worker_epochs, batch_size)	Huấn luyện mô hình ở các worker nodes.
9.	async def train_sync(self, master_epochs, worker_epochs, batch_size)	Huấn luyện mô hình kiểu đồng bộ ở master node.
10.	async def train_async(self, master_epochs, worker_epochs, batch_size)	Huấn luyện mô hình kiểu bất đồng bộ ở master node.
11.	async def shutdown(self)	Shutdown cluster.

Ví dụ 3.4 trình bày một ứng dụng huấn luyện DNN phân tán kiểu đồng bộ trên DDLF. Đầu tiên, thông tin cấu hình của cụm máy tính được đọc để tạo đối tượng của lớp *App*. Sau đó, gọi phương thức *app.connect()* để tạo kết nối giữa master node và cluster. Phương thức *app.load_dataset()* được gọi để tải dataset về master node và các worker nodes. Tùy vào nhu cầu thực tế, chúng ta có thể tải toàn bộ dataset hoặc một partition của dataset về mỗi worker nodes. Gọi phương thức *app.create_model()* để tạo mô hình DL trên master node và worker nodes. Để huấn luyện mô hình, chúng ta gọi phương thức *app.train_sync()* cho kiểu đồng bộ hoặc *app.train_async()* cho kiểu bất đồng bộ. Tham số *master_epochs* dùng để khai báo số *epochs* cần thiết để huấn luyện mô hình trên master node. Tham số *worker_epochs* dùng để khai báo số *epochs* mà mô hình được huấn luyện trên mỗi worker node. Sau khi thực hiện đủ số *worker_epochs*, các worker nodes sẽ gửi *gradients* về master node để cập nhật mô hình chính ở master node. Cần lưu ý rằng, tham số *master_epochs* nên là bội số của tham số *worker_epochs*. Sau khi hoàn thành việc huấn luyện mô hình, chúng ta có thể đánh giá mô hình bằng phương thức *app.evaluate_model()*, và lưu mô

hình để dùng lại sau này bằng phương thức *app.save_model(path)*. Cuối cùng, gọi phương thức *app.close()* để đóng kết nối giữa master node và cluster.

Ví dụ 3.4. Một ứng dụng huấn luyện DNN phân tán kiểu đồng bộ trên DDLF

```
async def main():
    hosts, ports = read_config()
    app = App(Cluster(hosts, ports))
    await app.connect()
    await app.load_dataset()
    await app.create_model()
    await app.train_sync(master_epochs=100, worker_epochs=10, batch_size=32)
    await app.evaluate_model()
    await app.save_model(path)
    await app.close()
```

Ví dụ 3.5 trình bày việc hiện thực sơ lược của lớp *App*. Lớp *App* phải thừa kế từ interface *IApp* để dùng lại những phương thức đã được hiện thực trong interface *IApp*. Do ứng dụng phải tương tác với cluster nên bên trong lớp *App* phải lưu tham chiếu của đối tượng lớp *Cluster*. Các phương thức *load_dataset()* và *create_model()* bắt buộc phải hiện thực theo yêu cầu cụ thể. Các phương thức *train()*, *train_sync()*, và *train_async()*, nếu chấp nhận sự hiện thực mặc định, chúng ta không cần override chúng; ngược lại, chúng ta có thể override chúng một cách thích hợp.

Ví dụ 3.5. Hiện thực lớp App

```
class App(IApp):
def __init__(self, cluster: Cluster):
super().__init__(cluster)

async def load_dataset(self):
# your code here

async def create_model(self):
# your code here

[async def train(self, worker_epochs, batch_size):
# your code here]

[async def train_sync(self, master_epochs, worker_epochs, batch_size):
# your code here]

[async def train_async(self, master_epochs, worker_epochs, batch_size):
# your code here]
```

Ví dụ 3.6 trình bày chi tiết hiện thực phương thức *load_dataset()* để tải dataset MNIST.

Ví dụ 3.6. Phương thức *load_dataset()* của lớp *App*

```
async def load_dataset(self):
    # load the MNIST dataset
    (self.x_train, self.y_train), (self.x_test, self.y_test) = datasets.mnist.load_data()
    # Make sure images have shape (28, 28, 1)
    self.x_train = np.expand_dims(self.x_train, -1)
    self.x_test = np.expand_dims(self.x_test, -1)
    # Normalize pixel values from [0, 255] to [-0.5, 0.5] to to make it easier to work with
    self.x_train, self.x_test = (self.x_train / 255.0) - 0.5, (self.x_test / 255.0) - 0.5
```

Ví dụ 3.7 trình bày chi tiết hiện thực phương thức *create_model()* để tạo mô hình CNN cho việc phân loại trên dataset MNIST.

Ví dụ 3.7. Phương thức *create_model()* của lớp *App*

```
async def create_model(self):
    input_shape = (28, 28, 1) # for MNIST
    # input_shape = (32, 32, 3) # for CIFAR10
    # create the model
    self.model = models.Sequential()
    self.model.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
    self.model.add(layers.MaxPooling2D(pool_size=(2, 2)))
    self.model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
    self.model.add(layers.Flatten())
    self.model.add(layers.Dense(64, activation='relu'))
    self.model.add(layers.Dense(10, activation='softmax'))
```

3.5.5 Đặc điểm của DDLF

DDLF có các ưu điểm sau đây:

1. Đơn giản: Mục đích thiết kế là giữ cho DDLF càng đơn giản càng tốt. Đơn giản khi cài đặt và sử dụng giúp dễ dàng tiếp cận và triển khai nhanh ứng dụng, dẫn đến tiết kiệm thời gian và công sức, cuối cùng đạt được mục tiêu hiệu quả về kinh tế. Để cài đặt DDLF trên cluster chỉ cần chạy script đơn giản. Để triển khai ứng dụng huấn luyện DNN phân tán chỉ cần mở rộng interface *IApp* là có thể sử dụng lại các phương thức tiện ích sẵn có, không cần lập trình phức tạp.

2. Đa nhiệm hiệu quả: DDLF dùng package *asyncio* của *Python 3.7+* để thực hiện đa nhiệm hiệu quả hơn *multi-threading*, giúp hiệu quả xử lý tốt hơn.

3. Đa nền tảng: DDLF có thể cài đặt trên các hệ điều hành khác nhau, chỉ cần hệ điều hành hỗ trợ Python 3.7+.

4. Linh hoạt: DDLF cho phép master node điều khiển các worker nodes mà không có bất kỳ hạn chế nào, tạo điều kiện thuận lợi để giải phóng tư duy lập trình, việc lập trình không bị phụ thuộc hay bị gò bó vào bất kỳ cấu trúc lập trình cứng ngắc nào.

5. Khả năng mở rộng: DDLF cho phép master node thêm động biến và phương thức vào các worker nodes để phục vụ nhu cầu thực tế đa dạng. Các biến và phương thức bổ sung tồn tại lâu dài trên worker nodes cho đến khi thực hiện thao tác xóa hoặc shutdown cluster. Điều này cho phép sử dụng lại biến và phương thức, giảm thiểu nhu cầu truyền dữ liệu hoặc chương trình từ master node đến worker nodes, dẫn đến rút ngắn thời gian xử lý. Đặc điểm này cũng giúp cho người lập trình tổ chức động dữ liệu và chương trình ở các worker nodes một cách dễ dàng và hợp lý, từ đó việc lập trình phân tán sẽ tự nhiên hơn, không gò bó trong khuôn khổ, và hiệu suất được nâng cao.

6. Hiệu quả: Qua thực nghiệm, DDLF chứng tỏ có hiệu quả tốt vì nó giúp huấn luyện nhanh DNN và tạo ra mô hình chất lượng.

DDLF cũng có những nhược điểm sau đây:

1. Chưa hỗ trợ đa ngôn ngữ: Hiện tại DDLF chỉ hỗ trợ ngôn ngữ Python.

2. Chức năng tiện ích chưa phong phú: Việc cung cấp nhiều chức năng tiện ích giúp phát triển nhanh ứng dụng. Tuy nhiên, trong phiên bản này chức năng tiện ích chưa nhiều, cần bổ sung thêm.

3. Tối ưu chi phí liên lạc: Trong quá trình huấn luyện DNN phân tán, việc liên lạc giữa master node và các worker nodes diễn ra thường xuyên và thường phải truyền số lượng lớn dữ liệu như weights/gradients, data. Vì vậy, một nền tảng tốt phải có biện pháp tối ưu hóa chi phí liên lạc. Tuy nhiên, trong phiên bản này DDLF chưa hiện thực các biện pháp tối ưu chi phí liên lạc.

4. Phương pháp huấn luyện chưa đa dạng: Hiện tại, DDLF chỉ hỗ trợ phương pháp huấn luyện như: song song hóa dữ liệu đồng bộ và bất đồng bộ.

3.5.6 Những hạn chế của Apache Spark và giải pháp trên DDLF

3.5.6.1 Vấn đề tổ chức động dữ liệu và chức năng ở worker nodes

Trên DDLF, người dùng có thể thêm/bớt biến hay phương thức vào worker nodes mà không có bất kỳ sự ràng buộc nào. Điều này giúp cho người dùng phát triển và triển khai ứng dụng phân tán dễ dàng. Trong khi đó, Apache Spark cho phép thực hiện việc này một cách hạn chế thông qua cấu trúc *map/reduce* và *shared variables*.

A. Vấn đề của cấu trúc map/reduce

Cấu trúc *map/reduce* của Apache Spark được áp dụng rất hiệu quả trong xử lý dữ liêu lớn nhưng không phù hợp trong việc huấn luyện DNN phân tán vì việc lập trình bị gò bó trong cấu trúc *map/reduce*, ràng buộc tư duy lập trình, dẫn đến việc triển khai ứng dụng huấn luyện DNN kém linh hoạt và không hiệu quả. Mặt khác, do phụ thuộc vào cấu trúc *map/reduce*, hàm *worker_ train(data_ iterator)* chỉ nhận một tham số *data_ iterator* (là 1 data partition). Người lập trình không thể truyền thông tin khác cho hàm *worker_ train()* khi cần. Dùng biện pháp thay thế thì quá phiền phức. Điều này làm khó triển khai ứng dụng.

Để giải quyết vấn đề này, chúng tôi cho phép master node tự do gửi code đến các worker nodes để thực thi. Nội dung code có thể là một đoạn code hoặc một method. Ngoài ra, chúng tôi còn cho phép master node thêm động các method vào worker nodes. Các method này sẽ tồn tại trên các worker nodes cho đến khi master node yêu cầu xóa các methods hoặc shutdown cluster. Trong tương lai, chúng tôi sẽ cho phép các phương thức bổ sung này có thể tồn tại vĩnh viễn ở các worker nodes kể cả sau khi shutdown cluster. Giải pháp này mang đến những tiện ích sau:

- Code gửi đến các worker nodes không phụ thuộc vào bất kỳ cấu trúc lập trình nào (như *map/reduce*). Các method có thể có số lượng tham số và kiểu dữ liệu tham số tùy ý. Điều này giúp giải phóng tư duy lập trình, tạo điều kiện thuận lợi để phát triển ý tưởng.
- Đối với những phương thức được sử dụng lặp đi lặp lại nhiều lần, chúng ta không cần gửi chúng đến các worker nodes trước mỗi lần sử dụng như trong *map/reduce* mà chỉ cần gửi chúng đến các worker nodes một lần, sau đó sử dụng lại nhiều lần. Điều này giúp giảm lưu lượng truyền tải trên mạng.

B. Shared variables

Trong Apache Spark, cơ chế chia sẻ dữ liệu giữa master node và worker nodes rất hạn chế thông qua *broadcast và accumulated variables*. Broadcast variables là biến chỉ đọc, accumulated variables là biến chỉ cộng. Hơn nữa, các biến này có nội dung giống nhau trên tất cả các worker nodes và chúng sẽ bị mất khi kết thúc một Spark session.

Để việc chia sẻ dữ liệu được thuận tiện hơn, trên DDLF, chúng tôi cho phép master node có thể thêm động biến vào các worker nodes và thao tác thoải mái trên chúng. Nội dung các biến này có thể giống nhau hoặc khác nhau trên các worker nodes. Giống như các phương thức bổ sung, các biến này sẽ tồn tại lâu dài trên các worker nodes cho đến khi master node yêu cầu xóa chúng hoặc shutdown cluster. Ngoài ra, master node cũng có thể yêu cầu worker nodes truy cập hệ thống file cục bộ hoặc phân tán để tải hoặc lưu trữ dữ liệu ở các worker nodes. Giải pháp này có những ích lợi sau:

- Thuận tiện trong việc tổ chức dữ liệu ở các worker nodes, tạo điều kiện thuận lợi để phát triển ý tưởng và triển khai các thuật toán.
- Do các biến được thêm động có thể tồn tại cho đến khi shutdown cluster nên master node chỉ cần gửi dữ liệu đến worker nodes một lần rồi sử dụng lại nhiều lần, thậm chí qua các ứng dụng khác nhau. Điều này giúp giảm chi phí liên lạc giữa master node và các worker nodes.

3.5.6.2 Việc xử lý dữ liệu huấn luyện

Trong Thuật toán 3.2 và Thuật toán 3.3, việc xáo trộn dữ liệu, phân hoạch lại dữ liệu và gửi các phân hoạch mới đến các worker nodes trong mỗi lần lặp là không thể tránh khỏi. Nhưng đây lại là những thao tác tốn nhiều thời gian xử lý, làm ảnh hưởng đến hiệu quả của toàn bộ hệ thống huấn luyện DNN phân tán. Chúng tôi đề xuất giải pháp như sau:

- Worker nodes sẽ tải sẵn toàn bộ dữ liệu huấn luyện nếu trong cache chưa có.
- Ở mỗi lần lặp, master node chỉ cần xáo trộn chỉ số của tập dữ liệu huấn luyện, phân hoạch tập chỉ số, và gửi các phân hoạch chỉ số đến worker nodes.
- Worker nodes sẽ lấy phân hoạch dữ liệu huấn luyện của riêng nó dựa vào tập dữ liệu huấn luyện đã tải sẵn trước đó và phân hoạch chỉ số đã nhận.

Giải pháp này có các ưu điểm sau:

- Giảm tối đa kích thước dữ liệu được gửi từ master node đến các worker nodes vì kích thước của 1 chỉ số, là một số nguyên, rất nhỏ so với kích thước của 1 dữ liệu huấn luyện, nhất là dữ liệu ảnh. Điều này dẫn đến giảm tối đa thời gian gửi dữ liệu, giúp cải thiện đáng kể hiệu suất của hệ thống huấn luyện DNN phân tán.
- Các datasets có thể được cache trong bộ nhớ hoặc hệ thống file cục bộ của worker nodes để sử dụng lại trong các ứng dụng huấn luyện DNN khác nhau.
- Tránh tạo một điểm thắt cổ chai tại master node do master node không phải phân phối khối lượng lớn dữ liệu huấn luyện cho tất cả worker nodes.

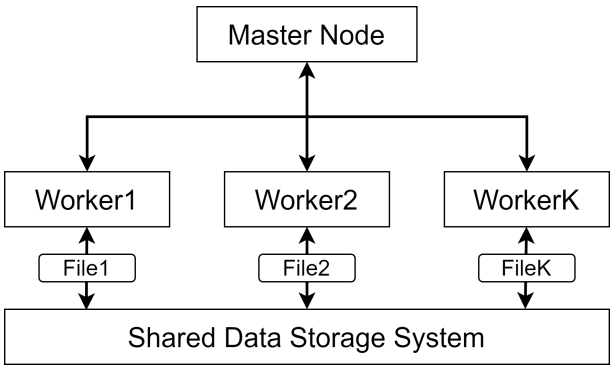
Thuật toán 3.6 là phiên bản cải tiến của Thuật toán 3.2 dựa vào giải pháp trên. Thuật toán 3.3 cũng có thể được cải tiến tương tự.

Thuật toán 3.6. Huấn luyện phân tán DNN kiểu đồng bộ
trên K worker nodes được cải tiến

```
1  function master_train() do
2      dataset ← load_dataset()
3      indices ← get_indices(dataset)
4      model ← create_model()
5      broadcast_model(model)
6      w ← model.get_weights()
7      i ← 0
8      while i < M do
9          indices ← shuffle(indices)
10         ∪_{j=1}^{K} I_j ← partition(indices)
11         send ∪_{j=1}^{K} I_j to all workers
12         broadcast_weights(w)
13         g ← await collect(g_j) from all workers
14         w ← w − (1/K) × Σ_{j=1}^{K} g_j
15         model.set_weights(w)
16         if is_convergent(model) then
17             broadcast_signal(STOP)
18             break
19         end if
20         i ← i + N
21     end while
22 end function
```

```
23
24  function worker_j_train() do
25      dataset ← load_dataset()
26      model_j ← await receive_model()
27      i ← 0
28      while i < M do
29          I_j ← await receive_indices()
30          P_j ← get_partition(dataset, I_j)
31          w_j ← await receive_weights()
32          model_j.set_weights(w_j)
33          model_j.fit(P_j, epochs = N, batch_size = BS)
34          g_j ← model_j.get_gradients()
35          send g_j to the master node
36          i ← i + N
37          if has_signal(STOP) then
38              break
39          end if
40      end while
41  end function
```

Giải pháp trên giải quyết tốt cho trường hợp toàn bộ tập dữ liệu huấn luyện có thể lưu trữ vừa trong một worker node. Tuy nhiên, khi kích thước của tập dữ liệu huấn luyện quá lớn, vượt quá khả năng lưu trữ và xử lý trên một worker node; chúng ta cần cải tiến giải pháp trên như Hình 3.6. Đầu tiên, chúng ta có thể phân hoạch tập dữ liệu huấn luyện thành K files, với K là số lượng worker nodes. Chúng ta lưu các file dữ liệu này trong bất kỳ hệ thống lưu trữ dùng chung nào mà tất cả worker nodes có thể truy cập được như: HDFS, S3, Cloud Storage. Sau đó, mỗi worker node sẽ tải file dữ liệu dành riêng cho nó để thực hiện việc huấn luyện DNN nếu trong cache chưa có dữ liệu. Sau khi tải, các file dữ liệu này có thể được cache trong bộ nhớ hoặc hệ thống file cục bộ của worker nodes để sử dụng lại. Giải pháp này rất hữu ích khi một tập dữ liệu được dùng cho nhiều ứng dụng huấn luyện DNN khác nhau, hoặc khi một DNN cần phải huấn luyện nhiều lần; vì không phải tải lại một khối lượng lớn dữ liệu.



Hình 3.6. Tải các datasets vào DDLF

Những giải pháp trên được hiện thực dễ dàng trên DDLF nhờ vào tính linh hoạt của nó (như cho phép thêm động biến và phương thức vào worker nodes, cho phép truy cập hệ thống file cục bộ của worker nodes). Tuy nhiên, trên Apache Spark, chúng ta rất khó

triển khai các giải pháp này do Apache Spark không cho phép master node truy cập vào bộ nhớ và hệ thống file của worker nodes một cách linh hoạt.

3.5.6.3 Vấn đề liên lạc bất đồng bộ giữa master node và các worker nodes

Apache Spark không hỗ trợ liên lạc bất đồng bộ giữa master node và các worker nodes. Do đó, để triển khai ứng dụng huấn luyện DNN kiểu bất đồng bộ, người phát triển ứng dụng phải tự bổ sung cơ chế liên lạc bất đồng bộ vào Apache Spark hoặc sử dụng một framework có hỗ trợ cơ chế liên lạc bất đồng bộ.

Trên DDLF, master node tự do điều khiển các hoạt động ở worker node mà không có bất kỳ sự hạn chế nào. Vì vậy, khi cần, master node có thể liên lạc với worker nodes theo kiểu đồng bộ hoặc bất đồng bộ rất tự nhiên.

3.5.7 Thách thức trong huấn luyện DNN phân tán

Một thách thức lớn trong huấn luyện DNN phân tán là việc truyền dữ liệu giữa master node và worker nodes. Việc truyền một khối lượng dữ liệu lớn (data, gradients) thường xuyên sẽ làm chậm toàn bộ quá trình tính toán, ngay cả khi sử dụng mạng tốc độ cao.

Cụ thể, trong song song dữ liệu, các partition dữ liệu được gửi cho nhiều worker nodes. Mỗi worker node sẽ huấn luyện mô hình của mình bằng partition dữ liệu đã nhận để rút ra gradients. Các gradients này được gửi về master node để tính toán bộ lại bộ trọng số. Sau đó bộ trọng số mới lại được gửi đến tất cả worker nodes để cập nhật mô hình của chúng. Quá trình này có thể rất chậm vì số lượng gradients bằng với số lượng trọng số. Mạng nơ-ron sâu thường chứa hàng triệu, thậm chí hàng tỷ tham số với độ chính xác 32-bit cần được trao đổi trong mỗi bước cập nhật.

Ngoài ra, việc xáo trộn dữ liệu, phân hoạch lại dữ liệu và gửi các phân hoạch mới đến các worker nodes trong mỗi lần lặp là không thể tránh khỏi. Nhưng đây lại là những thao tác tốn nhiều thời gian xử lý, làm ảnh hưởng đến hiệu quả của toàn bộ hệ thống huấn luyện DNN phân tán. Chúng tôi đề xuất giải pháp như sau:

- Worker nodes sẽ tải sẵn toàn bộ dữ liệu huấn luyện nếu trong cache chưa có.
- Ở mỗi lần lặp, master node chỉ cần xáo trộn chỉ số của tập dữ liệu huấn luyện, phân hoạch tập chỉ số, và gửi các phân hoạch chỉ số đến worker nodes.
- Worker nodes sẽ lấy phân hoạch dữ liệu huấn luyện của riêng nó dựa vào tập dữ liệu huấn luyện đã tải sẵn trước đó và phân hoạch chỉ số đã nhận.

Giải pháp này có các ưu điểm sau:

- Giảm tối đa kích thước dữ liệu được gửi từ master node đến các worker nodes vì kích thước của 1 chỉ số, là một số nguyên, rất nhỏ so với kích thước của 1 dữ liệu huấn luyện, nhất là dữ liệu ảnh. Điều này dẫn đến giảm tối đa thời gian gửi dữ liệu, giúp cải thiện đáng kể hiệu suất của hệ thống huấn luyện CNN phân tán.
- Các datasets có thể được cache trong bộ nhớ hoặc hệ thống file cục bộ của worker nodes để sử dụng lại trong các ứng dụng huấn luyện CNN khác nhau.

- Tránh tạo một điểm thắt cổ chai tại master node do master node không phải phân phối khối lượng lớn dữ liệu huấn luyện cho tất cả worker nodes.

3.6 Kết quả nghiên cứu

Một phần kết quả nghiên cứu được trình bày trong bài báo sau và đã được đăng trong tạp chí Intelligent Data Analysis, IOS Press:

[1] Phuc Do, Trung Phan, Hung Le, Brij Gupta; *"Building a Knowledge Graph by using Cross Lingual Transfer Method and Distributed MinIE Algorithm on Apache Spark"*; Neural Computing and Applications - Springer, 2020 (SCIE – Q1 - IF: 5.102).

[2] Trung Phan, Phuc Do. *"A Novel Framework to Enhance the Performance of Training Distributed Deep Neural Networks"*. Intelligent Data Analysis, IOS Press, 2023.

3.7 Đóng góp của nghiên cứu

Nghiên cứu của chúng tôi có những đóng góp như sau:

1. Phân tích những khó khăn khi triển khai ứng dụng phân tán trên Apache Spark.
2. Xây dựng nền tảng xử lý phân tán DDLF giúp triển khai ứng dụng phân tán và huấn luyện phân tán DNN dễ dàng và hiệu quả.
3. Cung cấp một giải pháp huấn luyện phân tán DNN hiệu quả với tập dữ liệu huấn luyện lớn.
4. Tạo điều kiện thuận lợi cho việc nghiên cứu và triển khai các ứng dụng phân tán.

Chương 4

NHÚNG ĐỒ THỊ & TÌM KIẾM SỰ TƯƠNG ĐỒNG

4.1 Giới thiệu bài toán

Trong nhiều lĩnh vực khác nhau, việc tìm kiếm các vector tương đồng với một vector truy vấn cho trước trong tập vector đã trở thành một nhiệm vụ cơ bản và quan trọng, thu hút sự chú ý đáng kể. Khi các tập vector ngày càng mở rộng về số chiều và qui mô, vấn đề này trở nên ngày càng quan trọng hơn. Việc xác định các vector tương đồng yêu cầu tính toán khoảng cách giữa vector truy vấn và tất cả các vector khác trong tập hợp. Tuy nhiên, đối với các tập vector có số chiều cao bao gồm hàng trăm chiều và bao gồm hàng triệu vector hoặc hơn, việc tính toán trực tiếp tất cả khoảng cách đôi một trở nên không khả thi do yêu cầu thời gian và tài nguyên tính toán lớn.

Để giải quyết thách thức này, đã có một số kỹ thuật lập chỉ mục tập vector được nghiên cứu, bao gồm KD-Tree [30], Ball Tree [31] và Cover Tree [32]. Trong đó, KD-Tree nổi bật với khả năng dễ triển khai và hiệu quả. Tuy nhiên, KD-Tree chỉ có thể lập chỉ mục các tập vector cục bộ, vừa với khả năng lưu trữ của một máy tính. Trong thời đại Big Data, các tập vector thường rất lớn, vượt quá khả năng lưu trữ của một máy tính. Do đó, KD-Tree không thể lập chỉ mục cho các tập vector qui mô lớn như vậy.

Chương này giới thiệu một phương pháp lập chỉ mục mới gọi là DKD-Tree (Distributed KD-Tree), một sự mở rộng của cấu trúc KD-Tree, được thiết kế để lập chỉ mục các tập vector qui mô lớn theo cách phân tán, giúp giải quyết bài toán tìm kiếm các vector tương đồng với một vector truy vấn cho trước một cách hiệu quả. Ban đầu, cấu trúc DKD-Tree được triển khai trên nền tảng tính toán phân tán Apache Spark [15, 33, 34], tuy nhiên gặp khó khăn và không đạt được kết quả như kỳ vọng. Sau đó, cấu trúc DKD-Tree được triển khai trên nền tảng DDLF (Distributed Deep Learning Framework - một nền tảng xử lý phân tán do chúng tôi phát triển) [35], vượt qua các hạn chế gặp phải trong quá trình triển khai ứng dụng phân tán trên Apache Spark và cho kết quả rất khả quan. Bên cạnh đó, chúng tôi cũng tiến hành các thực nghiệm trên nhiều tập vector với qui mô khác nhau để so sánh và đánh giá hiệu suất của DKD-Tree trên cả hai nền tảng Apache Spark và DDLF.

4.2 Mục tiêu nghiên cứu

Trong bài toán này, chúng tôi tập trung giải quyết bài toán: *"Lập chỉ mục phân tán tập vector nhiều chiều qui mô lớn để tìm kiếm nhanh các vector tương đồng với vector truy vấn cho trước"*. Bài toán này được chia thành ba phần:

- Xây dựng cấu trúc DKD-Tree để lập chỉ mục phân tán tập vector nhiều chiều qui mô lớn.
- Dùng cấu trúc DKD-Tree để thực hiện các truy vấn phạm vi và k láng giềng gần nhất.
- So sánh hiệu suất của DKD-Tree trên cụm Spark và cụm DDLF.

4.3 Những công trình liên quan

KD-Tree là một cấu trúc dữ liệu được sử dụng rộng rãi để lập chỉ mục tập vector. Khi tập vector tiếp tục tăng kích thước và số chiều, nhu cầu về các giải pháp lập chỉ mục có khả năng mở rộng đã thúc đẩy việc nghiên cứu kỹ thuật phân tán *KD-Tree*. Trong khảo sát này, chúng ta sẽ xem xét các nghiên cứu tiêu biểu về việc phân tán *KD-Tree*.

Năm 2011, Aly và cộng sự [36] đề xuất sử dụng KD-Tree phân tán để giải quyết vấn đề truy vấn hình ảnh từ các bộ sưu tập rất lớn. Họ đã chia bộ sưu tập thành các phân hoạch nhỏ hơn và xây dựng KD-Tree cho mỗi phân hoạch này. Quá trình xây dựng KD-Tree phân tán có thể được thực hiện song song trên nhiều máy tính, giúp tận dụng tài nguyên và tăng tốc quá trình xây dựng cây. Kết quả thử nghiệm trên tập dữ liệu thực tế đã cho thấy hiệu suất truy vấn của phương pháp đề xuất cải thiện đáng kể so với các phương pháp truyền thống. Bài báo cũng đề cập đến các ứng dụng tiềm năng của phương pháp này trong việc truy vấn hình ảnh từ các nguồn dữ liệu rất lớn như bộ sưu tập hình ảnh từ mạng xã hội hoặc cơ sở dữ liệu y học.

Phương pháp được đề xuất trong bài báo có các ưu điểm sau: 1) Hiệu suất cao: Phương pháp này tận dụng khả năng tính toán song song của các máy tính trong hệ thống phân tán để tăng hiệu suất xây dựng và truy vấn trên KD-Tree. Cụ thể, KD-Tree được chia thành các cây con, mỗi cây con được lưu trữ trên một máy tính riêng. Khi có truy vấn, máy chủ sẽ phân phối truy vấn cho các máy tính lưu trữ các cây con phù hợp. Điều này giúp giảm thời gian truy vấn tổng thể, đặc biệt là đối với các bộ sưu tập hình ảnh lớn. 2) Khả năng mở rộng: Phương pháp này có thể mở rộng dễ dàng theo quy mô của bộ sưu tập hình ảnh. Khi số lượng hình ảnh tăng lên, có thể thêm các máy tính mới vào hệ thống để lưu trữ các cây con mới. 3) Khả năng chịu lỗi: Phương pháp này có khả năng chịu lỗi cao. Nếu một máy tính trong hệ thống bị lỗi, các cây con được lưu trữ trên máy tính đó vẫn có thể được truy cập từ các máy tính khác.

Tuy nhiên, phương pháp này có các nhược điểm sau: 1) Việc mở rộng phương pháp này để xử lý các bộ sưu tập càng lớn có thể đối mặt với các khó khăn về tài nguyên tính toán và quản lý. 2) Hiệu suất của phương pháp có thể bị ảnh hưởng bởi các yếu tố như môi trường mạng và tình trạng của các máy tính tham gia vào quá trình xử lý. 3) Hệ thống phân tán có thể đối mặt với các vấn đề về độ tin cậy và khả năng chịu lỗi.

Năm 2018, Wehr và cộng sự [37] đề xuất phương pháp xây dựng KD-Tree song song cho các điểm ba chiều trên GPU sử dụng thuật toán sắp xếp duy trì tính song song cao trong suốt quá trình xây dựng. Theo các tác giả, đây là phương pháp hiệu quả để xây dựng cây KD-Tree trong môi trường song song, đồng thời cải thiện hiệu suất của quá trình xây dựng cây so với các phương pháp truyền thống.

Ưu điểm của phương pháp này là: 1) Hiệu suất cao: Trong bài báo, các tác giả đã so sánh phương pháp của họ với một số phương pháp xây dựng KD-Tree song song khác trên GPU. Kết quả cho thấy phương pháp của họ có thể đạt hiệu suất cao hơn đáng kể, đặc biệt là đối với các tập dữ liệu lớn. Ví dụ, đối với một tập dữ liệu 10 triệu điểm 3 chiều, phương pháp của họ có thể xây dựng KD-Tree trong khoảng 5 giây, trong khi các phương pháp khác có thể mất tới 10 giây hoặc hơn. 2) Thích ứng: Phương pháp này sử dụng chiến lược phân chia và sắp xếp thích ứng, có thể cải thiện hiệu suất cho các tập dữ liệu có kích thước khác nhau. Ví dụ, đối với các tập dữ liệu nhỏ, phương pháp này có thể sử dụng các kỹ thuật sắp xếp đơn giản hơn để tiết kiệm thời gian. 3) Đơn giản: Phương pháp này tương đối đơn giản để triển khai, giúp dễ dàng sử dụng trong các ứng dụng thực tế. Phương pháp này chỉ sử dụng một số kỹ thuật cơ bản như sắp xếp và phân hoạch.

Tuy nhiên, phương pháp này cũng có những nhược điểm sau: 1) Không phải mọi hệ thống đều có GPU mạnh. Điều này có thể hạn chế khả năng áp dụng phương pháp của bài báo trong một số môi trường. 2) Chiến lược chia và sắp xếp linh hoạt có thể phụ thuộc vào dữ liệu cụ thể và tình trạng của cây tại mỗi bước. Điều này có thể làm cho việc tinh chỉnh và áp dụng phương pháp trở nên phức tạp hơn, và có thể không phù hợp trong mọi trường hợp. 3) Mặc dù linh hoạt, chiến lược chia và sắp xếp có thể không cho kết quả tối ưu nhất. Điều này có thể dẫn đến KD-Tree không được xây dựng một cách tối ưu hoặc không đạt được hiệu suất dự kiến.

Năm 2022, Chakravorty và cộng sự [38] trình bày một cơ chế có thể mở rộng để xây dựng KD-Tree cho dữ liệu phân tán, dựa trên các giá trị trung vị gần đúng cho từng phân chia đệ quy của dữ liệu. Họ cung cấp sự đảm bảo về mặt lý thuyết về chất lượng của phép tính gần đúng của phương pháp này.

Ưu điểm của phương pháp này là: 1) Tính hiệu quả: Phương pháp này có hiệu quả cao trong các truy vấn tìm kiếm gần nhất. 2) Khả năng mở rộng: Phương pháp này có thể mở rộng theo chiều ngang, tức là có thể thêm các nút mới vào cây phân tán mà không cần thay đổi cấu trúc của cây. Điều này giúp cho phương pháp này có thể xử lý được lượng dữ liệu lớn. 3) Khả năng chịu lỗi: Phương pháp này có khả năng chịu lỗi tốt, tức là khi một nút trong cây phân tán bị lỗi, các nút còn lại vẫn có thể tiếp tục xử lý các truy vấn.

Phương pháp này cũng có những hạn chế sau: 1) Hiệu suất phụ thuộc vào môi trường mạng và tình trạng của các máy tính tham gia vào quá trình xử lý. 2) Phải giải quyết các vấn đề liên quan đến độ tin cậy và khả năng chịu lỗi.

KD-Tree phân tán là một giải pháp hứa hẹn để giải quyết những thách thức của việc quản lý tập vector nhiều chiều quy mô lớn trong các hệ thống phân tán. Các nghiên cứu đã khảo sát trình bày các phương pháp khác nhau để xây dựng và truy vấn KD-Tree phân tán. Tuy nhiên, các công trình trên chưa có hiện thực KD-Tree phân tán trên cụm Spark và cụm DDLF, cũng như so sánh hiệu suất của *KD-Tree* phân tán trên hai hệ thống này. Vì vậy, nội dung nghiên cứu này như là một sự bổ sung vào các công trình đang có, làm

phong phú và sâu sắc thêm những nghiên cứu về cấu trúc *KD-Tree* phân tán.

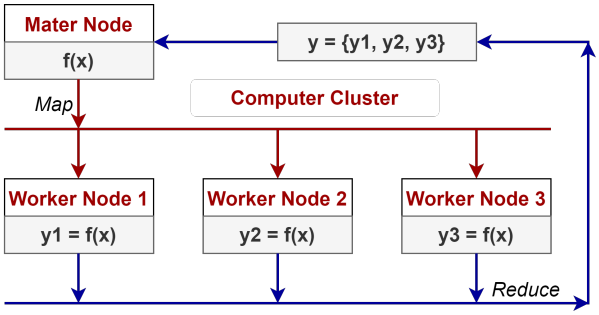
4.4 Cơ sở lý thuyết

4.4.1 HDFS

HDFS (Hadoop Distributed File System) là hệ thống tập tin phân tán và bền vững, được tối ưu hóa để lưu trữ và quản lý dữ liệu lớn. Nó được phát triển bởi dự án Apache Hadoop và là một phần không thể thiếu trong việc triển khai và sử dụng Hadoop để xử lý các tác vụ phân tích và tính toán trên dữ liệu lớn [11, 39, 40].

4.4.2 Cụm máy tính

Cụm máy tính là nhiều máy tính được kết nối mạng và chúng hoạt động giống như một thực thể duy nhất [19, 41]. Hình 4.1 minh họa một cụm máy tính tiêu biểu gồm bốn máy tính. Mỗi máy tính tham gia vào cụm được gọi là một node. Trong đó: 1) Master node: là máy tính chạy chương trình chính, gửi yêu cầu đến các worker node để thực thi song song và thu gom kết quả. 2) Worker node: là máy tính tham gia xử lý các yêu cầu của master node.



Hình 4.1. Cụm máy tính

Việc thực hiện một phương thức trên cụm máy tính gồm hai giai đoạn: *map* và *reduce*. Giai đoạn *map* (ánh xạ) biến đổi lệnh gọi phương thức ở master node thành lệnh gọi phương thức ở các worker node. Giai đoạn *reduce* (tổng hợp) thu gom kết quả từ các worker node về master node để tạo ra kết quả cuối cùng. Cả *map* và *reduce* đều được thực hiện song song trên các worker node.

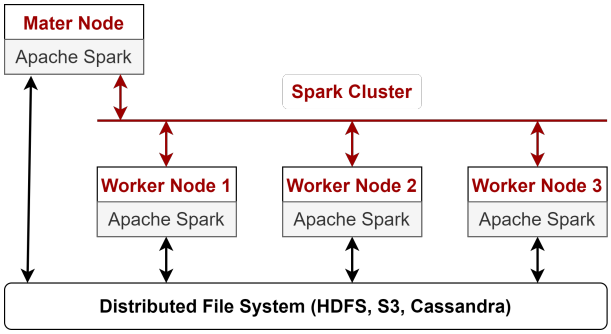
Ví dụ, như trong Hình 4.1, giai đoạn *map* biến đổi lệnh gọi phương thức *f(x)* ở master node thành lệnh gọi phương thức *f(x)* ở các worker node và trả về các kết quả y_1, y_2, y_3. Giai đoạn *reduce* thu gom kết quả từ các worker node về master node để tạo ra kết quả cuối cùng $y = \{y_1, y_2, y_3\}$.

4.4.3 Cụm Spark

Apache Spark (gọi tắt là Spark) là một nền tảng xử lý phân tán mã nguồn mở mạnh mẽ được phát triển bởi *Apache Software Foundation* [15, 33, 34]. Nó được xây dựng để xử lý và phân tích dữ liệu lớn một cách hiệu quả. *Apache Spark* cung cấp một nền tảng xử lý dữ liệu linh hoạt, có thể mở rộng từ một máy tính đơn lên cụm máy tính.

Các ứng dụng của *Apache Spark* bao gồm xử lý dữ liệu phân tán, xử lý dữ liệu trực tuyến, học máy, và nhiều ứng dụng khác trong lĩnh vực dữ liệu lớn và trí tuệ nhân tạo. Với tính năng mạnh mẽ và hiệu suất cao, *Apache Spark* đã trở thành một công cụ quan trọng trong việc xử lý và phân tích dữ liệu trong các doanh nghiệp và tổ chức hiện đại.

Cụm Spark là một hệ thống xử lý phân tán được xây dựng dựa trên nền tảng *Apache Spark*, được sử dụng rộng rãi trong các ứng dụng xử lý dữ liệu lớn, đồng thời cũng giúp tối ưu hóa hiệu suất tính toán. Hình 4.2 minh họa một cụm Spark gồm một master node và ba worker nodes. *Apache Spark* không có hệ thống tập tin riêng, nhưng nó có thể kết hợp với nhiều hệ thống tập tin phân tán như S3 [42], Cassandra [43, 44], HDFS [11, 39, 40]. *Cụm Spark* kết hợp với *HDFS* là một mô hình rất phổ biến.



Hình 4.2. Cụm Spark gồm bốn máy tính

4.4.4 Cụm DDLF

Để phục vụ công việc nghiên cứu, chúng tôi đã phát triển nền tảng *DDLF (Distributed Deep Learning Framework)*, chuyên dùng để triển khai các ứng dụng phân tán, khắc phục các khó khăn gặp phải trên *Apache Spark*. *DDLF* có những đặc điểm nổi bật như sau:

1. Đơn giản: Mục đích thiết kế là giữ cho *DDLF* càng đơn giản càng tốt. Đơn giản khi cài đặt và sử dụng giúp dễ dàng tiếp cận và triển khai nhanh ứng dụng, dẫn đến tiết kiệm thời gian và công sức, cuối cùng đạt được mục tiêu hiệu quả về kinh tế. Để cài đặt *DDLF* trên cụm máy tính chỉ cần chạy script đơn giản. Để triển khai ứng dụng phân tán chỉ cần mở rộng interface *IApp* là có thể sử dụng lại các phương thức tiện ích sẵn có, không cần lập trình phức tạp.

2. Linh hoạt và dễ mở rộng: *DDLF* cho phép *master node* điều khiển các *worker nodes* mà không có bất kỳ hạn chế nào. Người lập trình có thể tổ chức động dữ liệu và chương trình xử lý ở các *worker nodes* một cách dễ dàng và hợp lý. Điều này giúp giải phóng tư duy lập trình, việc lập trình không bị phụ thuộc hay bị gò bó trong bất kỳ cấu trúc lập trình nào (như *map/reduce*). Từ đó việc lập trình phân tán sẽ tự nhiên hơn, không gò bó trong khuôn khổ, và hiệu suất được nâng cao.

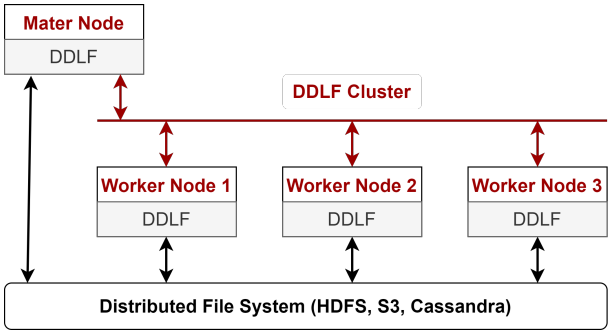
3. Hiệu suất cao: *DDLF* không thực hiện những kiểm tra ràng buộc phức tạp mà chỉ cung cấp nền tảng quản lý cụm, điều khiển việc truyền/nhận dữ liệu/code trong cụm và cung cấp những chức năng xử lý cơ bản, quan trọng nhất. Mặt khác, *DDLF* rất linh hoạt. Người lập trình có thể tự tổ chức dữ liệu và code xử lý cho ứng dụng

sao cho tối ưu. Chính vì sự đơn giản và linh hoạt đã giúp phát triển ứng dụng phân tán hiệu suất cao.

4. Khả năng mở rộng/thu hẹp: DDLF cho phép mở rộng/thu hẹp hệ thống dễ dàng bằng cách thêm/bớt worker node trong cụm và chỉ thay đổi một file cấu hình. Điều này giúp dễ dàng điều chỉnh qui mô của hệ thống xử lý cho phù hợp với qui mô dữ liệu.

Tương tự như *Apache Spark*, *DDLF* không có hệ thống tập tin riêng, nhưng nó có thể kết hợp với nhiều hệ thống tập tin phân tán như S3, Cassandra, HDFS. Để có thêm thông tin về DDLF, vui lòng tham khảo bài báo [35].

Cụm DDLF là một hệ thống xử lý phân tán được xây dựng dựa trên nền tảng *DDLF*. *Cụm DDLF* cho phép triển khai các ứng dụng phân tán đơn giản, linh hoạt và hiệu quả. Hình 4.3 minh họa *cụm DDLF* gồm một master node và ba worker nodes.



Hình 4.3. Cụm DDLF gồm bốn máy tính

4.4.5 Không gian Metric

Định nghĩa 4.8. Không gian Metric

Cho trước M là một tập hợp khác rỗng. Một ánh xạ d:M×M $\longrightarrow$ R sao cho với mọi $x, y, z \in M$*, các điều kiện sau đây được thỏa mãn:*
1) $d(x, y) = d(y, x)$ *(tính đối xứng)*
2) $0 < d(x, y) < \infty (x \neq y)$ *và* $d(x, x) = 0$ *(tính không âm)*
3) $d(x, y) \leq d(x, z) + d(z, y)$ *(bất đẳng thức tam giác)*
thì d được gọi là một khoảng cách hay một metric trên M và một cặp có thứ tự (M,d) được gọi là không gian metric [45].

Không gian metric (M,d) thường được viết là M, trong đó d được hiểu ngầm khi không nhầm lẫn. Có rất nhiều khoảng cách phổ biến như Manhattan, Chebyshev, Euclid, Minkowski. Tuy nhiên, trong bài báo này chỉ trình bày khoảng cách Euclid.

Định nghĩa 4.9. Khoảng cách Euclid

Trong R^n*, khoảng cách Euclid hoặc Euclid metric giữa hai điểm (hoặc vector)* $x = (x_1, x_2, \ldots, x_n)$ *và* $y = (y_1, y_2, \ldots, y_n)$ *là chiều dài của đoạn thẳng nối chúng* $(\overline{xy})$ *và được*

thể hiện bằng công thức Pythagore:

$$d(x,y) = \left[\sum_{i=1}^{n}(x_i - y_i)^2\right]^{\frac{1}{2}} \tag{4.1}$$

Ví dụ, trong R^2, cho $x = (12, 18)$ và $y = (9, 14)$ thì $d(x,y) = \sqrt{(12-9)^2 + (18-14)^2} = 5$.

Để tìm kiếm các vector tương đồng của một vector cho trước, chúng ta dùng truy vấn phạm vi hoặc k láng giềng gần nhất. Sau đây là định nghĩa của chúng [45, 46].

Định nghĩa 4.10. Truy vấn phạm vi (Range query)

Cho trước một tập hợp vector V, một hàm khoảng cách d, một vector truy vấn $v \in V$, và một bán kính r. Truy vấn phạm vi $range(v,r)$ là chọn tất cả các vector $v_i \in V$ sao cho $d(v_i, v) \leq r$.

Định nghĩa 4.11. Truy vấn k láng giềng gần nhất (kNN query)

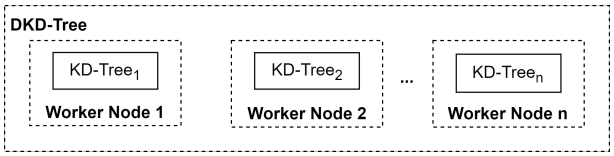
Cho trước một tập hợp vector V, một hàm khoảng cách d, một vector truy vấn $v \in V$, và một số nguyên $k \geq 1$. Truy vấn k láng giềng gần nhất $kNN(v,k)$ là chọn k vector $v_i \in V$ gần vector v nhất.

4.4.6 KD-Tree

KD-Tree (K-Dimensional Tree), được phát minh bởi *Jon Louis Bentley* vào năm 1975, là một cây nhị phân phân hoạch không gian k chiều, hay nói cách khác là lập chỉ mục cho tập vector k chiều, giúp thực hiện nhanh *kNN* và *Range* query. Độ phức tạp của thuật toán truy vấn trong trường hợp trung bình là $O(log\ n)$ [30].

4.4.7 DKD-Tree

DKD-Tree (Distributed KD-Tree) là một cấu trúc dữ liệu được mở rộng từ cấu trúc *KD-Tree* và được đề xuất trong bài báo này. Nó là cấu trúc dữ liệu phân tán dùng để lập chỉ mục tập vector nhiều chiều qui mô lớn, giúp thực hiện nhanh truy vấn phạm vi và truy vấn k láng giềng gần nhất. Thực chất, mỗi cấu trúc *DKD-Tree* là một tập hợp các cấu trúc *KD-Tree* được phân tán trong cụm máy tính. Mỗi worker node chứa ít nhất một cấu trúc *KD-Tree*. Hình 4.4 trình bày cấu trúc *DKD-Tree* mà mỗi worker node chỉ chứa một cấu trúc *KD-Tree*.



Hình 4.4. Cấu trúc DKD-Tree

4.5 Phương pháp luận

Đầu tiên, tập vector được chia thành n phân hoạch. Để đảm bảo việc cân bằng tải giữa các worker nodes thì 1) n phải là bội số của số worker nodes trong cụm xử lý; 2) Kích thước của các phân hoạch càng đều nhau càng tốt. Sau đó, *KD-Tree* được tạo ra cho từng phân hoạch. Tập hợp các *KD-Tree* này chính là cấu trúc *DKD-Tree*. Cấu trúc *DKD-Tree* được hiện thực trên *cụm Spark* và *cụm DDLF*. Nhiều thực nghiệm được thực hiện trên cả hai loại cụm để so sánh hiệu quả của hai cách hiện thực. Sau đây là nội dung chi tiết của giải pháp.

4.5.1 Xây dựng cấu trúc DKD-Tree

Quá trình xây dựng cấu trúc *DKD-Tree* từ tập vectơ V được mô tả trong Thuật toán 4.7 và được minh họa trong Hình 4.5. Tập vector V có thể rất lớn nên được lưu trên hệ thống file phân tán (Distributed File System - DFS). Đầu tiên, tập vectơ V được tải từ hệ thống file phân tán và được chia thành n phân vùng. Phân vùng thứ i được dùng để tạo một cấu trúc *KD-Tree*$_i$. Mỗi cấu trúc *KD-Tree*$_i$ được lưu vào file *kdtree*$_i$*.bin* trên hệ thống file cục bộ hoặc phân tán để sử dụng lại sau này.

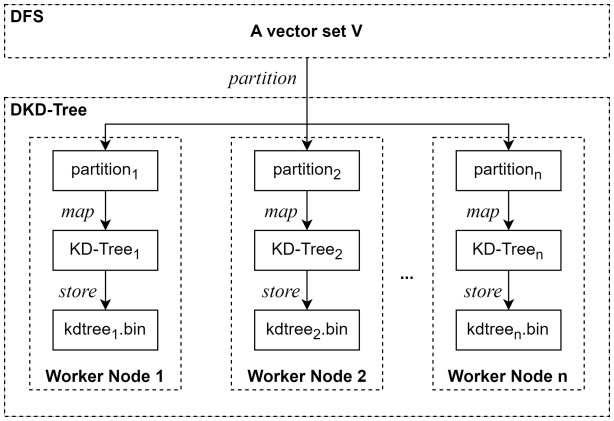
Thuật toán 4.7. Xây dựng cấu trúc *DKD-Tree*

Input : *V: tập vector.*
n: số phân vùng.
Output: *n files kdtree*$_i$*.bin chứa n cấu trúc KD-Tree*$_i$*.*

```
1  ⋃_{i=1}^{n} partition_i ← load and partition V into n partitions
2  foreach partition_i in ⋃_{i=1}^{n} partition_i do parallel
3      kdtree_i ← new KDTree(partition_i)
4      store kdtree_i to file kdtree_i.bin
5  end for parallel
```

Khi chọn n cần xem xét hai yếu tố: 1) n phải là bội số của số máy trong cụm để việc cân bằng tải giữa các máy trong cụm tốt hơn vì mỗi máy sẽ xử lý số lượng cấu trúc *KD-Tree* giống nhau. 2) Tổng kích thước của các cấu trúc *KD-Tree* mà một máy xử lý phải nhỏ hơn kích thước RAM của máy đó để tránh tình trạng sử dụng bộ nhớ ảo, dẫn đến giảm hiệu quả khi truy vấn. Trong trường hợp tổng kích thước của các cấu trúc *KD-Tree* mà một máy xử lý lớn hơn kích thước RAM của máy đó thì phải mở rộng cụm xử lý bằng cách gắn thêm worker nodes và phân hoạch lại tập vector. Như vậy, n phụ thuộc vào số máy trong cụm và dung lượng RAM của các máy trong cụm. Để đơn giản hóa việc trình bày, trong hình chỉ thể hiện n là số máy trong cụm.

Việc phân hoạch tập vector càng đều nhau càng tốt vì khi kích thước các phân hoạch chênh lệch lớn thì ảnh hưởng đến việc cân bằng tải giữa các worker nodes, dẫn đến giảm hiệu suất của toàn bộ hệ thống xử lý do worker nodes nhanh (do xử lý ít dữ liệu) phải đợi worker nodes chậm (do xử lý nhiều dữ liệu).



Hình 4.5. Xây dựng cấu trúc DKD-Tree

4.5.2 Truy vấn phạm vi

Việc thực hiện truy vấn phạm vi dựa trên cấu trúc *DKD-Tree* được mô tả trong Thuật toán 4.8 và được minh họa trong Hình 4.6. Trong giai đoạn *map*, lệnh gọi phương thức $range(v, r)$ ở master node sẽ được chuyển thành lệnh gọi phương thức $range(v, r)$ ở các worker nodes. Trong giai đoạn *reduce*, các tập vector kết quả V_i ở các worker nodes được thu gom về master node để tạo ra kết quả cuối cùng là $\bigcup_{i=1}^{n} V_i$.

Thuật toán 4.8. Thực thi *Range query* dựa trên *DKD-Tree*

```
Input : v: vector truy vấn.
        r: số thực dương.
Output: tập vectors v_i sao cho d(v, v_i) ≤ r.
```

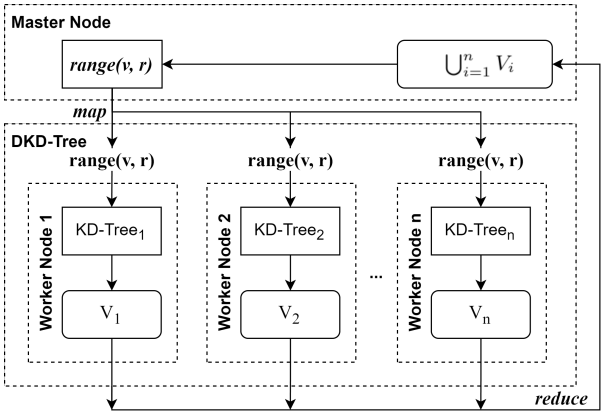
```
1  //map
2  foreach kdtree_i in dkdtree do parallel
3      V_i ← kdtree_i.range(v, r)
4  end foreach parallel
5  //reduce
6  ⋃_{i=1}^{n} V_i ← collect V_i from worker nodes in parallel
7  return ⋃_{i=1}^{n} V_i
```

4.5.3 Truy vấn k láng giềng gần nhất

Tương tự như thực hiện truy vấn phạm vi, thực hiện truy vấn k láng giềng gần nhất dựa trên cấu trúc *DKD-Tree* được mô tả trong Thuật toán 4.9 và được minh họa trong Hình 4.7. Trong giai đoạn *map*, lệnh gọi phương thức $kNN(v, k)$ ở master node sẽ được chuyển thành lệnh gọi phương thức $kNN(v, k)$ ở các worker node. Trong giai đoạn *reduce*, các tập vector kết quả V_i ở các worker node được thu gom về master node. Sau đó, sắp thứ tự tập vector kết quả theo khoảng cách tăng dần và lấy *topk* để tạo ra kết quả cuối cùng.

Thuật toán 4.9. Thực thi truy vấn k láng giềng gần nhất dựa trên *DKD-Tree*

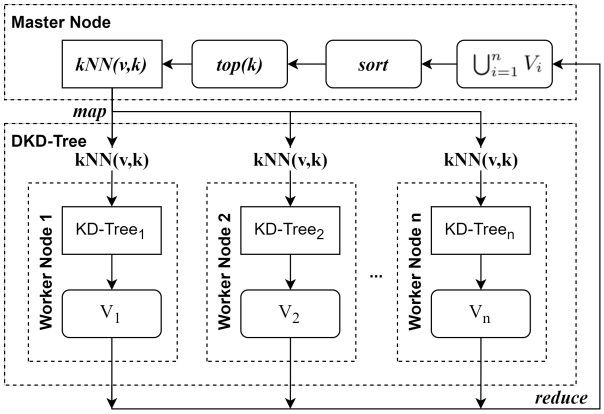
```
Input : v: vector truy vấn.
```



Hình 4.6. Thực thi truy vấn phạm vi trên DKD-Tree

k: số nguyên dương.
`Output:` *k vectors gần vector v nhất.*

```
1  //map
2  foreach kdtree_i in dkdtree do parallel
3      V_i ← kdtree_i.kNN(v, k)
4  end foreach parallel
5  //reduce
6  ⋃_{i=1}^{n} V_i ← collect V_i from worker nodes in parallel
7  V ← sort ⋃_{i=1}^{n} V_i by distance ascending
8  return V.top(k)
```



Hình 4.7. Thực thi truy vấn k láng giềng gần nhất trên DKD-Tree

4.5.4 Những hạn chế khi triển khai ứng dụng phân tán trên Apache Spark

Apache Spark là nền tảng chuyên dùng xử lý và phân tích dữ liệu lớn rất hiệu quả. Nhờ có nhiều ưu điểm, *Apache Spark* đã trở thành nền tảng xử lý dữ liêu lớn được dùng phổ biến nhất hiện nay. Tuy nhiên, không phải mọi loại ứng dụng phân tán đều có thể triển khai thuận lợi trên *Apache Spark*. Khi triển khai ứng dụng phân tán, như *DKD-Tree*

hay huấn luyện mạng neural, trên nền tảng *Apache Spark*, chúng tôi gặp một số trở ngại như sau:

1. Phụ thuộc vào cấu trúc map/reduce của Apache Spark: Cấu trúc *map/reduce* của Apache Spark rất hiệu quả trong xử lý dữ liêu lớn nhưng không phải phù hợp cho mọi loại ứng dụng phân tán vì việc lập trình bị gò bó trong cấu trúc *map/reduce*, ràng buộc tư duy lập trình, dẫn đến việc triển khai ứng dụng phân tán kém linh hoạt và không hiệu quả. Ví dụ, hàm được truyền cho cấu trúc *map/reduce*, chỉ nhận một tham số, người lập trình không thể truyền thông tin khác cho hàm khi cần.

2. Việc chia sẻ dữ liệu hạn chế: Cơ chế chia sẻ dữ liệu giữa *master node* và *worker nodes* trên *Apache Spark* rất hạn chế thông qua *broadcast variables* và *accumulated variables*. *Broadcast variables* là biến chỉ đọc, *accumulated variables* là biến chỉ cộng. Hơn nữa, các biến này có nội dung giống nhau trên tất cả các *worker nodes* và chúng sẽ bị mất khi kết thúc một *Spark session*.

3. Không thể can thiệp sâu vào worker nodes: Trên *Apache Spark*, người dùng không thể chủ động thêm/bớt biến hay phương thức vào *worker nodes*. Điều này khó khăn cho việc tổ chức dữ liệu và tổ chức xử lý ở các worker nodes, dẫn đến khó triển khai ứng dụng phân tán một cách linh hoạt.

4.5.5 So sánh hiệu suất của DKD-Tree trên cụm Spark và cụm DDLF

4.5.5.1 Tập dữ liệu thực nghiệm

Trong nghiên cứu này, tập dữ liệu thực nghiệm là ba triệu vectors được tạo từ tập dữ liệu *YAGO 4* (download tại https://yagoknowledge.org/data/yago4/en) [47]. Đây là một đồ thị tri thức lớn, biểu diễn tri thức về thế giới thực bằng các mối quan hệ giữa các thực thể, được sử dụng trong lĩnh vực trí tuệ nhân tạo và xử lý ngôn ngữ tự nhiên. *YAGO* là viết tắt của *"Yet Another Great Ontology"*.

Từ tập dữ liệu *YAGO 4*, ba triệu bộ ba (triples) được trích ra ngẫu nhiên để tạo thành ba triệu vectors 768 chiều bằng công cụ *Sentence Transformer* [48, 49]. Tập vector này khá lớn (kích thước 41.1 GB) nên được lưu trên hệ thống file phân tán *HDFS*.

4.5.5.2 Hệ thống thực nghiệm

Hệ thống thực nghiệm là cụm gồm bốn máy tính. Trong đó, một máy tính làm master node, ba máy còn lại làm worker node. Cấu hình chi tiết của hệ thống thực nghiệm được liệt kê trong Bảng 4.1.

Bảng 4.1. Cấu hình của hệ thống thực nghiệm

#	Property	Master Node	Worker Node
1.	Processor	Intel(R) Core™ i7-9750HF CPU @ 2.60GHz	Intel(R) Core™ i7-9750HF CPU @ 2.60GHz

#	Property	Master Node	Worker Node
2.	Cores	6	6
3.	Thread(s) per core	2	2
4.	RAM	16.0 GB	16.0 GB

Thực nghiệm được thực hiện trong hai môi trường: cụm Spark và cụm DDLF. Ngoài ra, để hiện thực *HDFS*, mỗi máy tính được cài đặt *Apache Hadoop*. Bảng 4.2 trình bày chi tiết các phần mềm được cài đặt trên các cụm máy tính.

Bảng 4.2. Các phần mềm được cài đặt

#	Phần Mềm	Cụm Spark	Cụm DDLF
1.	Python 3.8.5	√	√
2.	Apache Spark 3.0.0	√	√
3.	DDLF 2.0		√
4.	Hadoop 3.2.4	√	√

4.5.5.3 So sánh hiệu suất các loại truy vấn trên *cụm Spark* và *cụm DDLF*

Bảng 4.3 trình bày kết quả thực nghiệm so sánh hiệu suất của các loại truy vấn dựa trên cấu trúc *DKD-Tree* trên *cụm Spark* và *cụm DDLF*. Cột *Vectors* cho biết kích thước của cấu trúc *DKD-Tree*, tức là số lượng vectors được lập chỉ mục bởi cấu trúc *DKD-Tree* (đơn vị là nghìn). Cột *Range Query1* và *kNN Query1* là thời gian thực thi *Range Query* và *kNN Query* trên *cụm Spark* (đơn vị là giây). Cột *Range Query2* và *kNN Query2* là thời gian thực thi *Range Query* và *kNN Query* trên *cụm DDLF* (đơn vị là giây).

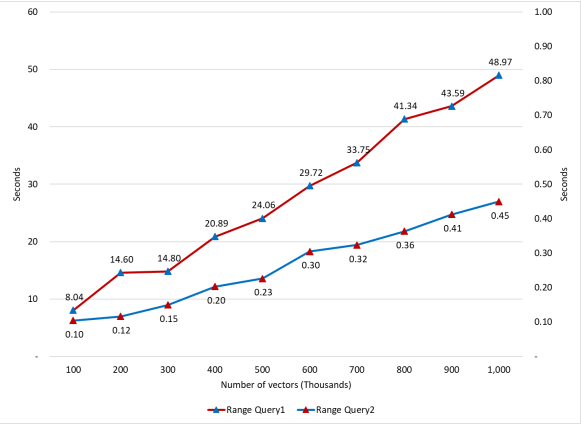
Kết quả thực nghiệm cho thấy việc thực hiện các truy vấn trên *cụm Spark* rất chậm so với *cụm DDLF*. Ví dụ, trường hợp kích thước *DKD-Tree* là 300 nghìn vectors, thời gian thực hiện *Range Query* trên *cụm DLLF* nhanh hơn trên *cụm Spark* $\frac{8.04}{0.10} \approx 84$ lần; thời gian thực hiện *kNN Query* trên *cụm DLLF* nhanh hơn trên *cụm Spark* $\frac{5.29}{0.07} \approx 76$ lần. Trường hợp kích thước *DKD-Tree* là ba triệu vectors, thời gian thực hiện *Range Query* trên *cụm DLLF* nhanh hơn trên *cụm Spark* $\frac{48.97}{0.45} \approx 109$ lần; thời gian thực hiện *kNN Query* trên *cụm DLLF* nhanh hơn trên *cụm Spark* $\frac{57.95}{0.47} \approx 123$ lần. Trong cả mười trường hợp, thời gian thực hiện các loại truy vấn trên *cụm DDLF* rất ấn tượng, chưa đến 0.5 giây. Trong khi đó, thời gian thực hiện các loại truy vấn trên *cụm Spark* chậm đến mức không chấp nhận được.

Bảng 4.3. So sánh thời gian thực hiện các truy vấn dựa trên DKD-Tree trên cụm Spark và cụm DDLF

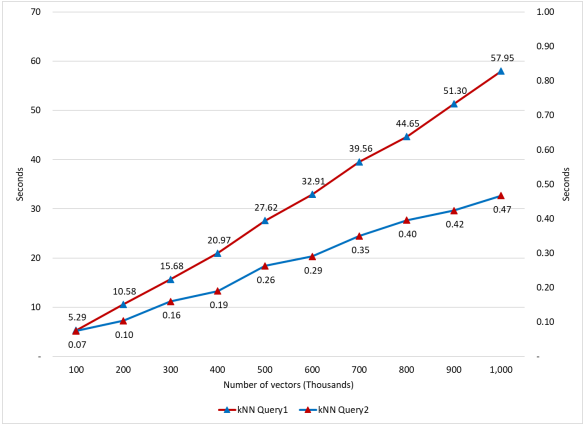
Vectors	Range Query1	Range Query2	kNN Query1	kNN Query2
300	8.04	0.10	5.29	0.07
600	14.60	0.12	10.58	0.10
900	14.80	0.15	15.68	0.16
1,200	20.89	0.20	20.97	0.19
1,500	24.06	0.23	27.62	0.26
1,800	29.72	0.30	32.91	0.29
2,100	33.75	0.32	39.56	0.35
2,400	41.34	0.36	44.65	0.40
2,700	43.59	0.41	51.30	0.42
3,000	48.97	0.45	57.95	0.47

Hình 4.8 là biểu đồ so sánh thời gian thực hiện *Rang Query* trên *cụm Spark* và *cụm DDLF*. Biểu đồ cho thấy thời gian thực hiện *Range Query* trên *cụm DDLF* rất nhỏ so với trên *cụm Spark*. Để thấy rõ đường *Range Query2*, nó được vẽ trong hệ trục phụ (bên phải) với tỷ lệ bằng 1:60 so với hệ trục chính (bên trái).

Tương tự, Hình 4.9 là biểu đồ so sánh thời gian thực hiện *kNN Query* trên *cụm Spark* và *cụm DDLF*. Biểu đồ cũng cho thấy thời gian thực hiện *kNN Query* trên *cụm DDLF* rất nhỏ so với trên *cụm Spark*. Để thấy rõ đường *kNN Query2*, nó cũng được vẽ trong hệ trục phụ (bên phải) với tỷ lệ bằng 1:70 so với hệ trục chính (bên trái).



Hình 4.8. So sánh thời gian thực hiện Range Query dựa trên DKD-Tree trên cụm Spark và cụm DDLF



Hình 4.9. So sánh thời gian thực hiện kNN Query dựa trên DKD-Tree trên cụm Spark và cụm DDLF

Một lưu ý quan trọng là trong cả Hình 4.8 và Hình 4.9 đều cho thấy rằng: khi kích thước của tập vector lớn lên thì thời gian thực hiện các truy vấn trên *cụm DDLF* tăng

chậm, trong khi đó thời gian tăng rất nhanh trên *cụm Spark*.

4.6 Kết quả nghiên cứu

Trong quá trình nghiên cứu và giải quyết bài toán, chúng tôi đã đạt được một số kết quả như sau:

[1] Phuc Do, Trung Phan, Huong Duong; *"DM-Tree: A Novel Indexing Method for Finding Similarities in Large Vector Sets"*; International Journal of Advanced Computer Science and Applications(IJACSA) - The SAI, 2020 (Scopus – Q3 - IF: 1.8).

[2] Phuc Do, Trung Phan; *"Distributed M-Tree for Similarity Search in Large Multimedia Database on Apache Spark"*; Chapter in "Handbook of Research on Multimedia Cyber Security IGI Global, 2020.

[3] Trung Phan, Phuc Do; *"DKD-Tree: Phương Pháp Lập Chỉ Mục Phân Tán Tập Vector Nhiều Chiều Qui Mô Lớn"*; Hội nghị KHCN Quốc gia lần thứ XVI về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin, FAIR XVI-2023, Đà Nẵng, ngày 28/09/2023, 2023.

4.7 Đóng góp của nghiên cứu

Nghiên cứu của chúng tôi có những đóng góp như sau:

1. Cung cấp giải pháp lập chỉ mục phân tán tập vector nhiều chiều qui mô lớn bằng DKD-Tree để tìm kiếm nhanh các vector tương đồng với vector truy vấn cho trước.
2. Phát triển các thuật toán phân tán thực hiện các truy vấn phạm vi và k láng giềng gần nhất.
3. So sánh hiệu suất của DKD-Tree trên cụm Spark và cụm DDLF.

Chương 5

ỨNG DỤNG

5.1 Giới thiệu bài toán

Hệ thống trả lời câu hỏi (Query Answering System - QAS) là một công cụ đối thoại giữa người và máy tính bằng ngôn ngữ tự nhiên. Vì tầm quan trọng thiết thực của nó, nhiều QAS đã được nghiên cứu và triển khai trong nhiều lĩnh vực như dịch vụ khách hàng, thương mại điện tử, chăm sóc sức khỏe, giáo dục và đào tạo...

Trong thế giới hiện đại, làm thế nào chúng ta có thể tận dụng thông tin để trả lời một câu hỏi phức tạp đã trở thành một thách thức đối với các hệ thống thông tin và trí tuệ nhân tạo. Bài toán phát triển hệ thống trả lời câu hỏi nhiều bước (Multi-hop Query Answering System - MQAS) đang nổi lên như một lĩnh vực quan trọng, đòi hỏi sự kết hợp của nhiều kỹ thuật và mô hình để trả lời các câu hỏi phức tạp nhanh chóng và chính xác.

Đối mặt với độ phức tạp của dữ liệu và yêu cầu đa chiều của câu hỏi, bài toán đặt ra những thách thức đáng kể, bao gồm tổ chức dữ liệu, xử lý ngôn ngữ tự nhiên, trích xuất thông tin nhanh và chính xác. Những thách thức này đồng thời mở ra những cơ hội để phát triển các phương pháp và mô hình mới để cải thiện khả năng trả lời câu hỏi nhiều bước.

Phát triển MQAS đang trở thành một lĩnh vực nghiên cứu quan trọng. Nó đưa ra nhiều thách thức nhưng cũng mang đến những cơ hội mới trong sự phát triển của trí tuệ nhân tạo và xử lý ngôn ngữ tự nhiên.

5.2 Mục tiêu nghiên cứu

Chúng tôi tập trung vào các chủ đề sau:

- Xây dựng một mạng thông tin lớn từ bộ dữ liệu YAGO 4, bao gồm các tri thức về thực thể, mối quan hệ và sự kiện.
- Thiết kế và phát triển một hệ thống MQAS dựa trên mạng thông tin đã xây dựng. Hệ thống sẽ có khả năng trả lời các câu hỏi nhiều bước thông qua việc sử dụng các mối quan hệ giữa các thực thể trên mạng thông tin.

- Tối ưu hóa hệ thống MQAS để trả lời câu hỏi nhanh chóng và chính xác.

5.3 Những công trình liên quan

MQAS được thiết kế để trả lời các câu hỏi yêu cầu lập luận trên nhiều mẩu thông tin, thường là từ nhiều nguồn. Trong những năm gần đây, người ta quan tâm nhiều đến việc phát triển các hệ thống như vậy do tiềm năng của chúng trong việc giải quyết các vấn đề phức tạp trong thế giới thực. Chúng tôi sẽ thảo luận về một số phương pháp và kỹ thuật cơ bản được sử dụng trong MQAS.

1. **Phương pháp tiếp cận dựa trên mạng nơ-ron (Neural Networks-Based Approaches):** Phương pháp tiếp cận dựa trên mạng nơ-ron đã được sử dụng rộng rãi trong các MQAS. Các cách tiếp cận này thường sử dụng một chuỗi các mạng thần kinh để trích xuất thông tin từ văn bản và thực hiện suy luận đối với thông tin được trích xuất. Sau đây là một số cách tiếp cận dựa trên mạng thần kinh phổ biến:

    - **Mạng bộ nhớ (Memory Networks):** Mạng bộ nhớ sử dụng bộ nhớ có kích thước cố định để lưu trữ thông tin từ văn bản đầu vào. Thông tin này sau đó được sử dụng để trả lời câu hỏi bằng cách thực hiện các bước lập luận. Một số MQAS đã sử dụng Mạng bộ nhớ, chẳng hạn như bộ dữ liệu bAbI và HotpotQA [50, 51].
    - **Mạng biến đổi (Transformer Networks):** Mạng biến áp sử dụng cơ chế tự chú ý để trích xuất thông tin từ văn bản đầu vào. Một số MQAS gần đây đã sử dụng các mạng này, chẳng hạn như bộ dữ liệu Câu hỏi tự nhiên và TriviaQA [52, 53].
    - **Mạng dựa trên đồ thị (Graph-based Networks):** Mạng dựa trên đồ thị sử dụng đồ thị để thể hiện mối quan hệ giữa các thực thể khác nhau trong văn bản đầu vào. Một số MQAS đã sử dụng các mạng này, chẳng hạn như bộ dữ liệu WikiHop và OpenBookQA [54, 55].

2. **Phương pháp tiếp cận dựa trên đồ thị tri thức (Knowledge Graph-Based Approaches):** Phương pháp tiếp cận dựa trên Sơ đồ tri thức sử dụng các biểu đồ tri thức bên ngoài để trả lời các câu hỏi phức tạp đòi hỏi phải suy luận về nhiều sự kiện. Các cách tiếp cận này thường sử dụng kết hợp khai thác thông tin dựa trên văn bản và lập luận dựa trên biểu đồ tri thức để trả lời các câu hỏi. Sau đây là một số cách tiếp cận dựa trên biểu đồ tri thức phổ biến:

    - **Trích xuất thông tin mở (OpenIE - Open Information Extraction):** OpenIE là một kỹ thuật trích xuất thông tin từ văn bản phi cấu trúc và chuyển đổi nó thành đồ thị tri thức. Đồ thị này sau đó có thể được sử dụng để suy luận về dữ liệu được trích xuất để trả lời các câu hỏi phức tạp. Một số MQAS đã sử dụng OpenIE, chẳng hạn như bộ dữ liệu ARC và QAngaroo [56, 57].
    - **Phương pháp tiếp cận dựa trên Ontology (Ontology-Based Approaches):** Suy luận dựa trên Ontology sử dụng các ontology chính thức để biểu diễn tri thức miền và thực hiện suy luận trên thông tin được trích xuất. Một số MQAS

đã sử dụng phương pháp này, chẳng hạn như bộ dữ liệu WebQuestions và BioASQ [58, 59].

3. **Phương pháp lai (Hybrid Approaches):** Phương pháp kết hợp kết hợp nhiều kỹ thuật, chẳng hạn như mạng thần kinh, biểu đồ tri thức và hệ thống dựa trên quy tắc, để trả lời các câu hỏi phức tạp. Những cách tiếp cận này tận dụng điểm mạnh của từng phương pháp để khắc phục những hạn chế của các kỹ thuật riêng lẻ. Sau đây là một số phương pháp lai phổ biến:

   - **Tạo truy xuất tăng cường (Retrieval-Augmented Generation):** Tạo truy xuất tăng cường sử dụng một hệ thống truy xuất để lấy các tài liệu liên quan và một hệ thống tạo để tạo ra câu trả lời cuối cùng. Một số MQAS gần đây đã sử dụng phương pháp này, chẳng hạn như tập dữ liệu Natural Questions và OpenBookQA [60, 61].
   - **Phân tích ngữ nghĩa (Semantic Parsing):** Phân tích ngữ nghĩa sử dụng các kỹ thuật xử lý ngôn ngữ tự nhiên và ngữ pháp chính thức để phân tích cú pháp văn bản đầu vào và tạo biểu diễn câu hỏi có cấu trúc. Biểu diễn có cấu trúc này sau đó được sử dụng để lập luận và tạo câu trả lời. Một số MQAS đã sử dụng phương pháp này, chẳng hạn như bộ dữ liệu Spider và WikiSQL [62, 63].

Ngoài ra, các xu hướng nghiên cứu mới nhất trong MQAS bao gồm:

1. **Mạng nơ-ron đồ thị (Graph Neural Networks - GNN):** GNN đã nổi lên như một cách tiếp cận phổ biến trong các hệ thống QA đa bước do khả năng mô hình hóa các mối quan hệ phức tạp giữa các thực thể trong biểu đồ tri thức. GNN đã được sử dụng để mã hóa cấu trúc biểu đồ và thông tin thực thể, cho phép lập luận hiệu quả hơn trên biểu đồ. Ví dụ về các hệ thống QA đa chặng dựa trên GNN gần đây bao gồm KEP-GNN và GraphQA [64, 65].

2. **Mô hình ngôn ngữ được huấn luyện trước (Pre-trained Language Models):** Các mô hình ngôn ngữ được huấn luyện trước, chẳng hạn như GPT-3 và T5, cũng đã được áp dụng cho QA đa bước, đạt được hiệu suất cao nhất trên một số bộ dữ liệu chuẩn. Các mô hình này được huấn luyện trên một lượng lớn dữ liệu văn bản và có thể tạo câu trả lời bằng ngôn ngữ tự nhiên cho các câu hỏi phức tạp. Nghiên cứu gần đây đã tập trung vào việc tinh chỉnh các mô hình này cho QA đa chặng, cũng như phát triển các phương pháp để kết hợp các nguồn tri thức bên ngoài [66, 67].

3. **Phương pháp tiếp cận đa phương thức (Multi-Modal Approaches):** Phương pháp tiếp cận đa phương thức kết hợp văn bản, hình ảnh và các phương thức khác đã cho thấy sự hứa hẹn trong QA đa bước. Ví dụ: bộ dữ liệu GQA chứa các câu hỏi yêu cầu suy luận về thông tin hình ảnh và văn bản, và nghiên cứu gần đây đã tập trung vào việc phát triển các phương pháp để kết hợp các phương thức này một cách hiệu quả. Ví dụ về các MQAS đa phương thức hiện tại bao gồm VCR-Net và FusionQA [68, 69].

5.4 Cơ sở lý thuyết

5.4.1 Mạng thông tin (Information Network)

Định nghĩa 5.12. Mạng thông tin

Mạng thông tin là một đồ thị có hướng $G = (V, E)$ kết hợp với hàm ánh xạ loại đối tượng $\varphi : V \longrightarrow A$ và hàm ánh xạ loại liên kết $\psi : E \longrightarrow R$. Mỗi đối tượng $v \in V$ thuộc một loại đối tượng cụ thể trong tập hợp $A : \varphi(v) \in A$ và mỗi liên kết $e \in E$ thuộc một loại quan hệ cụ thể trong tập hợp $R : \psi(e) \in R$ [1].

5.4.2 Mạng thông tin không đồng nhất (Heterogeneous Information Network - HIN)

Định nghĩa 5.13. Mạng thông tin không đồng nhất

Một mạng thông tin được gọi là mạng thông tin không đồng nhất nếu số loại đối tượng $|A| > 1$ hoặc số loại quan hệ $|R| > 1$; ngược lại, nó là một mạng thông tin đồng nhất [1].

5.4.3 Mạng thông tin không đồng nhất giàu nội dung (Content-Enriched Heterogeneous Information Network - C-HIN)

Định nghĩa 5.14. Mạng thông tin không đồng nhất giàu nội dung

Nếu tất cả các đỉnh và cạnh của HIN có văn bản mô tả thì HIN được gọi là C-HIN.

Ví dụ về *C-HIN* được trình bày trong Table 5.1 và Table 5.2.

Bảng 5.1. Các thực thể và mô tả của chúng

Thực Thể	Mô Tả
Hà Nội	Hà Nội là thủ đô của Việt Nam. Hà Nội là trung tâm thương mại, văn hóa và giáo dục của miền Bắc Việt Nam.
Huế	Huế là thành phố cổ của Việt Nam. Có rất nhiều cảnh đẹp ở Huế.
Đà Nẵng	Đà Nẵng là một thành phố ven biển ở miền Trung Việt Nam. Đà Nẵng là trung tâm thương mại và giáo dục của miền Trung Việt Nam.
Quảng Ngãi	Quảng Ngãi là một thành phố ở miền Trung Việt Nam. Sông Trà là một thắng cảnh thiên nhiên của Quảng Ngãi.
Hồ Tây	Hồ Tây là hồ nước ngọt lớn nhất của Hà Nội, Việt Nam; nằm về phía Tây Bắc của trung tâm thành phố.
Núi Ngự Bình	Núi Ngự Bình là một ngọn núi ở Huế. Núi Ngự Bình nằm cách trung tâm Huế 30 km.
Núi Ngũ Hành	Núi Ngũ Hành, hay còn gọi là Ngũ Hành Sơn, là một cụm năm ngọn núi đá cẩm thạch và đá vôi ở Đà Nẵng. Tất cả các ngọn núi đều có lối vào hang động và nhiều đường hầm.

Bảng 5.2. Các quan hệ/vị từ và mô tả của chúng

Quan Hệ/Vị Từ	**Mô Tả**
Specialty_Dish_Of	là đặc sản của
Born_In	được sinh ở
Located_At	được đặt ở
Is_In	ở trong
Folk_Song_Of	là dân ca của
Organized_At	được tổ chức ở
Is_Beautiful_Sight_Of	là thắng cảnh của

5.4.4 Đồ thị tri thức

Đồ thị tri thức là một đồ thị biểu diễn những mối quan hệ giữa các thực thể. Một thực thể có thể là một con người, một địa danh, một sự kiện, một tổ chức... Một mối quan hệ là sự kết hợp giữa hai thực thể thông qua một predicate và được biểu diễn bằng một bộ ba *<thực thể đầu, vị từ, thực thể đuôi>*. Đồ thị tri thức dùng để lưu trữ và biểu diễn tri thức. Nó cho phép con người và máy tính khai thác tốt hơn các mối quan hệ giữa các thực thể và suy luận ra những mối qua hệ mới. Về bản chất, đồ thị tri thức là một HIN hoặc một C-HIN. Đồ thị tri thức được xem là một trong số các nền tảng của AI. Ngày nay, nhiều công ty như Google, Microsoft, Facebook... đã phát triển đồ thị tri thức để phục vụ công việc của mình. Đồ thị tri thức của Google và Microsoft phục vụ cho công cụ tìm kiếm, đồ thị tri thức của Facebook phục vụ cho mạng xã hội. Về mặt hình thức, chúng ta có thể định nghĩa đồ thị tri thức như sau:

Định nghĩa 5.15. Đồ thị tri thức

Đồ thị tri thức $G = (V, E)$ là một đồ thị có hướng với các đỉnh là thực thể và các cạnh là bộ ba có dạng <thực thể đầu, vị từ, thực thể đuôi>. Mỗi bộ ba được ký hiệu là <h, p, t> để biểu diễn mối quan hệ p từ thực thể h đến thực thể t.

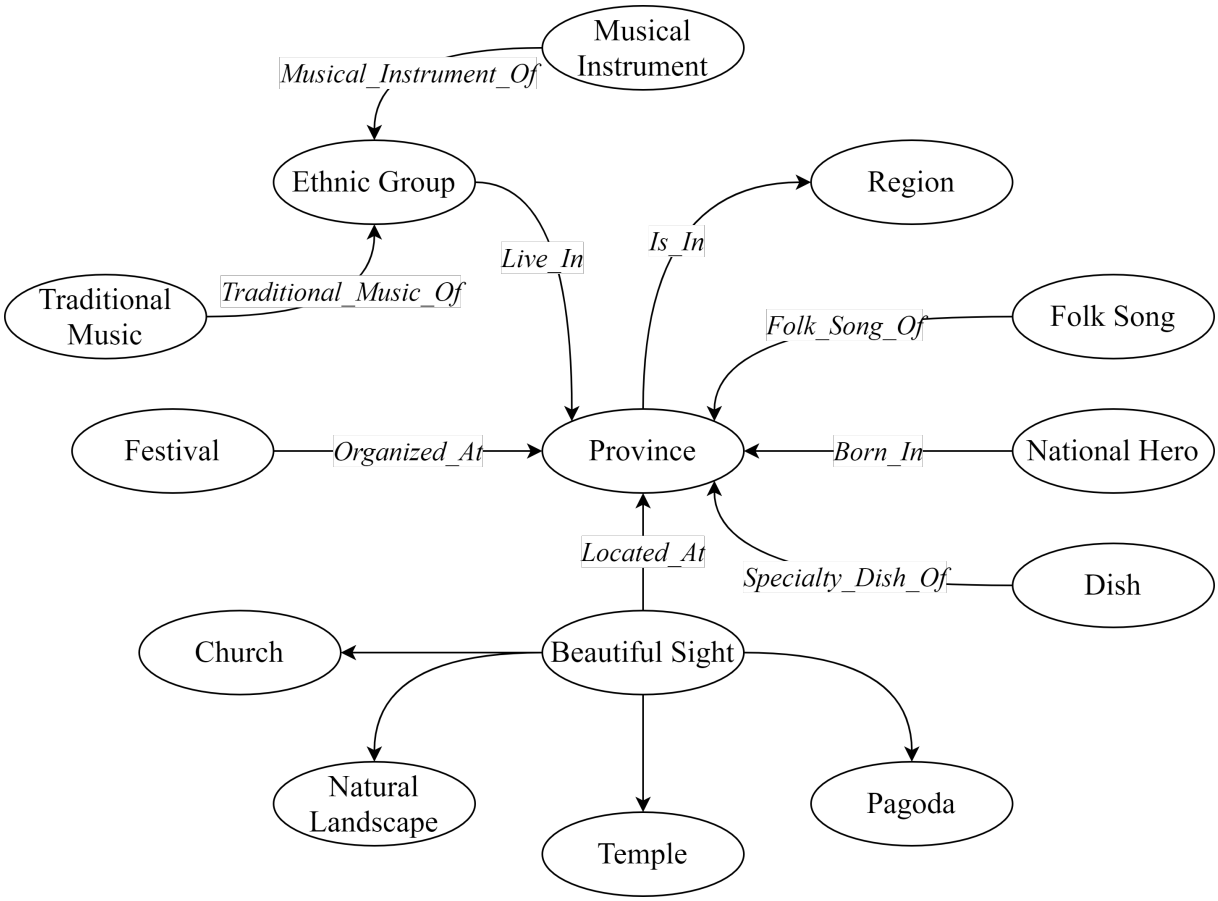
Đồ thị tri thức có thể dùng để cung cấp tri thức cho *QAS*. Khi kết hợp *đồ thị tri thức* với một tập luật chúng ta có thể suy luận ra những tri thức mới từ những tri thức có sẵn. Chúng ta cũng có thể thu thập thông tin tự động từ những nguồn tin cậy, phân tích thành những bộ ba và bổ sung vào đồ thị tri thức để tạo *QAS* có khả năng tự học. Với cơ chế suy luận và khả năng tự học 24x7, *QAS* sẽ ngày càng thông minh hơn; và chắc chắn một ngày không xa sẽ vượt qua khả năng của con người.

5.4.5 Lược đồ mạng

Định nghĩa 5.16. Lược đồ mạng (Network Schema)

Cho trước HIN $G = (V, E)$ cùng với ánh xạ kiểu đối tượng $\varphi : V \longrightarrow A$ và ánh xạ kiểu liên kết $\psi : E \longrightarrow R$. Lược đồ mạng của HIN là một đồ thị có hướng được định nghĩa trên các loại đối tượng A và các loại quan hệ R, ký hiệu $TG = (A, R)$ [1].

Chúng tôi đã xây dựng *C-HIN* về Du lịch Việt Nam. Lược đồ mạng của *C-HIN* này được hiển thị trong Figure 5.1. Trong *C-HIN* này, chúng tôi có một tập hợp các loại đối tượng $A = \{Beautiful\_Sight, Ethnic\_Group, Province, Dish, National\_Hero, Folk\_Song, Traditional\_Music, Festival, Region...\}$ và một tập hợp các loại quan hệ $R = \{Organized\_At, Located\_At, Folk\_Song\_Of, Born\_In, Specialty\_Dish...\}$.



Hình 5.1. Lược đồ mạng của *C-HIN* về Du lịch Việt Nam

5.4.6 Meta-path

Định nghĩa 5.17. Meta-path

Một meta-path P là một con đường được định nghĩa trên lược đồ mạng $TG = (A, R)$ của một HIN cho trước, được ký hiệu bằng $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} ... \xrightarrow{R_l} A_{l+1}$, định nghĩa một quan hệ kết hợp $R = R_1 \circ R_2 \circ ... \circ R_l$ giữa kiểu A_1 và A_{l+1}, trong đó $\circ$ là phép toán kết hợp trên quan hệ, l là chiều dài của meta-path [2].

Như vậy, nếu $l = 1$ thì R là quan hệ trực tiếp, nếu $l > 1$ thì R là quan hệ gián tiếp. Table 5.3 trình bày một số meta-path được rút trích ra từ lược đồ mạng của *C-HIN* về Du lịch Việt Nam.

Bảng 5.3. Một số meta-path của *C-HIN* về Du lịch Việt Nam

#	**Meta-Path**	**Relation Label**	**Relation Type**
1	Beautiful_Sight → Province		
		Located_At	direct
2	National_Hero → Province		
		Born_In	direct
3	Dish → Province		
		Specialty_Dish	direct
4	Folk_Song → Province		
		Folk_Song_of_Province	direct
5	Festival → Province		
		Organized_at_Province	direct
6	Beautiful_Sight → Province → Region		
		Beautiful_Sight_of_Region	indirect
7	Dish → Province → Region		
		Specialty_Dish_of_Region	indirect
8	Folk_Song → Province → Region		
		Folk_Song_of_Region	indirect

5.4.7 Path-instance

Định nghĩa 5.18. Path-instance

Một path-instance của một meta-path $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} ... \xrightarrow{R_l} A_{l+1}$ *là một đường đi p trong HIN, trong đó* $p : a_1 \xrightarrow{R_1} a_2 \xrightarrow{R_2} ... \xrightarrow{R_l} a_{l+1}$, a_i *là một đỉnh của HIN,* $a_j \xrightarrow{R_j} a_{j+1}$ *là một cạnh với đỉnh nguồn là* a_j *và đỉnh đích là* a_{j+1} *trong HIN.*

Một vài path-instance của meta-path $Beautiful\_Sight \xrightarrow{Located\_At} Province$ được trình bày trong Table 5.4.

Bảng 5.4. Một vài path-instance của meta-path $Beautiful\_Sight \xrightarrow{Located\_At} Province$

#	**Path-instance**
1	$(Ngu\ Binh\ Mountain) \xrightarrow{Located\_At} (Hue)$
2	$(Ngu\ Hanh\ Mountain) \xrightarrow{Located\_At} (Da\ Nang)$
3	$(Hoan\ Kiem\ Lake) \xrightarrow{Located\_At} (Ha\ Noi)$

Bảng 5.4. Một vài path-instance của meta-path $Beautiful\_Sight \xrightarrow{Located\_At} Province$

#	**Path-instance**
4	$(Ha\ Long\ Bay) \xrightarrow{Located\_At} (Quang\ Ninh)$

5.4.8 Câu hỏi đơn giản

Định nghĩa 5.19. Câu hỏi đơn giản

Cho trước một HIN $G = (V, E)$. q là câu hỏi bằng ngôn ngữ tự nhiên, q được xem là câu hỏi đơn giản nếu ∃! path-instance của một meta-path ∈ HIN có độ dài bằng 1, cung cấp nội dung trả lời cho q.

Ví dụ, câu hỏi q = *"Where was Nguyen Hue born in?"* là câu hỏi đơn giản vì ∃! path-instance *"*$(Nguyen\ Hue) \xrightarrow{Born\_In} (Binh\ Dinh)$*"* của meta-path *"*$National\_Hero \xrightarrow{Born\_In} Provin$ có độ dài bằng 1 cung cấp nội dung câu trả lời cho q.

5.4.9 Câu hỏi nhiều bước

Định nghĩa 5.20. Câu hỏi nhiều bước

Cho trước một HIN $G = (V, E)$. q là câu hỏi bằng ngôn ngữ tự nhiên, q được xem là câu hỏi phức tạp nếu ∃! path-instance của một meta-path ∈ HIN có độ dài lớn hơn 1, cung cấp nội dung trả lời cho q.

Ví dụ, câu hỏi q = *"Which region was Nguyen Hue born in?"* là câu hỏi phức tạp vì ∃! path-instance *"*$(Nguyen\ Hue) \xrightarrow{Born\_In} (Binh\ Dinh) \xrightarrow{Is\_In} (Central\ Vietnam)$*"* của meta-path *"*$National\_Hero \xrightarrow{Born\_In} Province \xrightarrow{Is\_In} Region$*"* có độ dài bằng 2 cung cấp nội dung câu trả lời cho q.

5.5 Phương pháp luận

5.5.1 Tạo đồ thị tri thức

Để tạo KG cung cấp tri thức cho QAS, chúng tôi tạo hai tập tin bộ ba: 1) types.csv: định nghĩa kiểu của các thực thể. 2) facts.csv: chứa các dữ kiện. Tập tin types.csv gồm 3 cột: Entity, Predicate, Type như trong Bảng 5.5. Tập tin facts.csv gồm 3 cột: Head Entity, Predicate, Tail Entity như trong Bảng 5.6. Trong công trình này, dữ liệu của KG được chúng tôi trích xuất từ bộ dữ liệu YAGO 4.

Bảng 5.5. Nội dung mẫu của tập tin types.csv

#	**Entity**	**Predicate**	**Type**
1.	<Titanic>	<type>	<Movie>
2.	<James Camerony>	<type>	<Human>

#	Entity	Predicate	Type
3.	<Leonardo DiCaprio>	<type>	<Human>
4.	<Kate Winslet>	<type>	<Human>
5.	<Twentieth Century Fox>	<type>	<Company>
6.	<Paramount Pictures>	<type>	<Company>
7.	<Los Angeles>	<type>	<City>
8.	<USA>	<type>	<Country>
9.	<UK>	<type>	<Country>

Bảng 5.6. Nội dung mẫu của tập tin facts.csv

#	Head Entity	Predicate	Tail Entity
1.	<Titanic>	<director>	<James Camerony>
2.	<Titanic>	<star>	<Leonardo DiCaprio>
3.	<Titanic>	<star>	<Kate Winslet>
4.	<Leonardo DiCaprio>	<birthPlace>	<Los Angeles>
5.	<Kate Winslet>	<birthPlace>	<Reading, Berkshire>
6.	<Los Angeles>	<belongTo>	<USA>
7.	<Reading, Berkshire>	<belongTo>	<UK>
8.	<Twentieth Century Fox>	<location>	<Los Angeles>
9.	<Paramount Pictures>	<location>	<Los Angeles>

Hình 5.2 và Hình 5.3 trình bày một phần của *KG*.

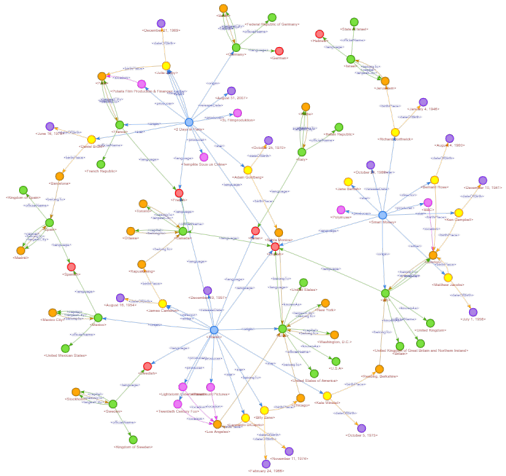
5.5.2 Thiết kế và phát triển một hệ thống MQAS

Hình 5.4 trình bày qui trình xây dựng MQAS gồm 10 công đoạn.

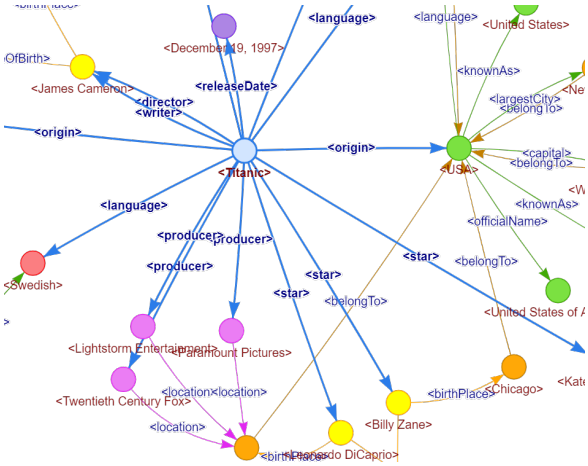
Hình 5.5 trình bày kiến trúc của MQAS. Đầu tiên, tập các cặp (câu hỏi, trả lời) được tạo ra từ KG. Tập các câu hỏi được chuyển thành tập vectors n chiều bằng *BERT*. Khi nhận một câu hỏi từ người sử dụng, MQAS sẽ dùng BERT để biến đổi câu hỏi thành vector q và thực hiện kNN(q, 1) (k-nearest neighbor query) trên tập vectors để tìm vector tương tự với vector q nhất, sau đó lấy câu trả lời tương ứng với vector tương tự để trả lời cho câu hỏi. Để thực hiện kNN() nhanh chóng, *DKD-Tree* được dùng để lập chỉ mục tập vectors.

5.5.3 Xử lý multi-hop meta-paths

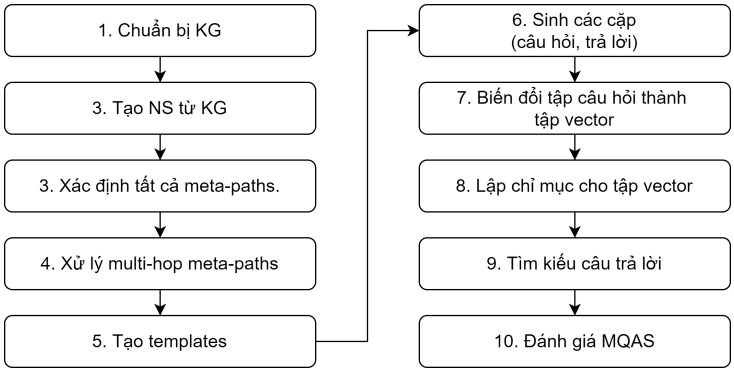
Các multi-hop meta-paths sẽ được chuyển thành one-hop meta-paths bằng cách xác định các quan hệ dẫn xuất nếu có. Bước này cần phải đánh giá quan hệ dẫn xuất có ý



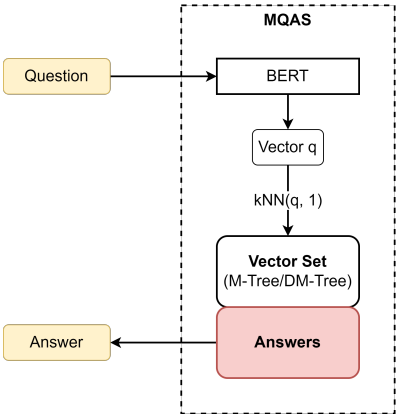
Hình 5.2. Một phần của *KG*



Hình 5.3. Một phần của *KG* được phóng to



Hình 5.4. Các công đoạn xây dựng MQAS



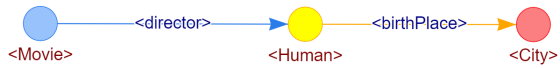
Hình 5.5. Kiến trúc của MQAS

nghĩa hay không. Nếu quan hệ dẫn xuất có ý nghĩa, chúng tôi sẽ tạo one-hop meta-path biểu diễn nó, ngược lại nó sẽ bị loại bỏ. Sau khi xử lý tất cả các multi-hop meta-paths, chúng tôi có tất cả one-hop meta-paths biểu diễn mối quan hệ có ý nghĩa giữa các kiểu thực thể. Hiện tại, chúng tôi đánh giá quan hệ dẫn xuất bằng phương pháp thủ công. Chúng tôi sẽ nghiên cứu thêm để tìm giải pháp thực hiện tự động.

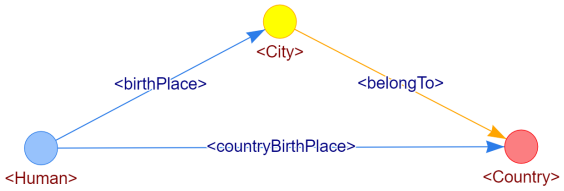
Ví dụ, từ two-hop meta-path $(Movie) - [director] \rightarrow (Human) - [birthPlace] \rightarrow (City)$ như trong Hình 5.6, chúng tôi không suy ra được quan hệ có ý nghĩa nên chúng tôi bỏ qua meta-path này. Từ two-hop meta-path $(Human) - [birthPlace] \rightarrow (City) - [belongTo] \rightarrow (Country)$ như trong Hình 5.7, chúng tôi suy ra được một quan hệ có ý nghĩa giữa kiểu thực thể <Human> và <Country> là $(Human) - [countryBirthPlace] \rightarrow (Country)$, nó cho biết <Human> sinh ở <Country> nào.

5.5.4 Tối ưu hóa hệ thống MQAS

Để tìm nhanh vector q trong tập vector lớn, chúng tôi phát triển cấu trúc *DKD-Tree* để lập chỉ mục tập vector lớn trong môi trường phân tán. Về bản chất, *DKD-Tree* là tập

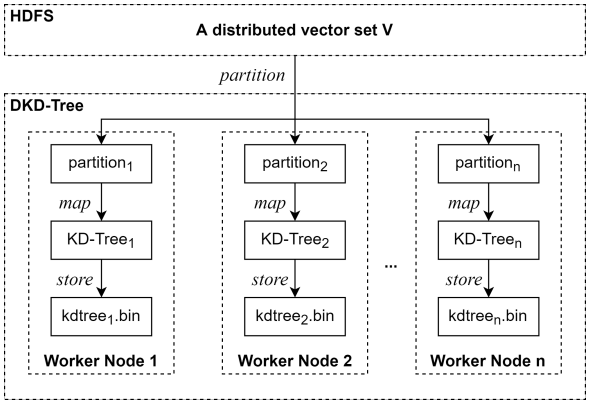


Hình 5.6. Two-hop meta-path không suy ra được quan hệ có ý nghĩa

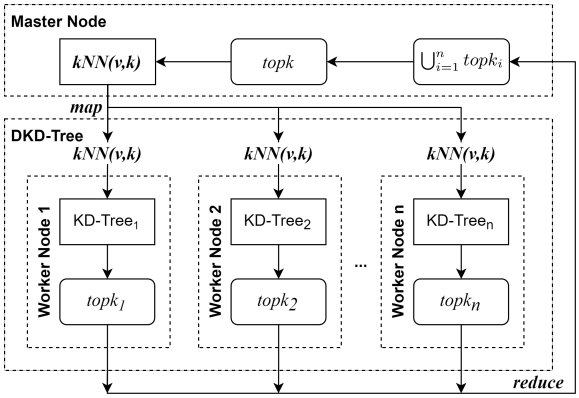


Hình 5.7. Two-hop meta-path suy ra được quan hệ có ý nghĩa

hợp các *KD-Tree*, mỗi *KD-Tree* được lưu trữ trong một node trong cụm máy tính như trong Hình 5.8. Hình 5.9 trình bày quá trình thực thi kNN(v,k) query dựa trên cấu trúc *DKD-Tree*.



Hình 5.8. Cấu trúc *DKD-Tree*



Hình 5.9. Quá trình thực thi kNN Query phân tán dựa trên cấu trúc *DKD-Tree*

5.6 Kết quả nghiên cứu

Trong quá trình nghiên cứu và giải quyết bài toán, chúng tôi đã đạt được một số kết quả như sau:

[1] Trung Phan, Phuc Do; *"Building a Vietnamese Question Answering System Based on Knowledge Graph and Distributed CNN"*; Neural Computing and Applications - Springer, 2021 (SCIE – Q1 - IF: 5.102).

[2] Trung Phan, Phuc Do; *"Một Hệ Thống Mới Để Trả Lời Câu Hỏi Nhiều Bước Dựa Trên Đồ Thị Tri Thức Lớn Theo Tiếp Cận Phân Tán"*; Hội nghị KHCN Quốc gia lần thứ XV về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin, FAIR XV-2022, Hà Nội, ngày 03-04/11/2022, 2022.

5.7 Đóng góp của nghiên cứu

Những đóng góp của chúng tôi có thể được tóm tắt như sau:

1. Đề xuất phương pháp xây dựng MQAS mô phỏng hoạt động học theo thời gian của con người.
2. Dùng kỹ thuật lưu trữ và xử lý phân tán để lưu trữ và xử lý KG lớn.
3. Cung cấp giải pháp xử lý câu hỏi nhiều bước dựa vào multi-hop meta-paths.
4. Cung cấp các thuật toán quan trọng trong quá trình xây dựng MQAS.
5. Cung cấp giải pháp lập chỉ mục phân tán tập vectors lớn để tìm kiếm nhanh câu trả lời.

Chương 6

KẾT LUẬN & HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Trong tiểu luận tổng quan này, chúng tôi đã trình bày tổng quan về ba bài toán của luận án cũng như hướng tiếp cận để giải quyết từng bài toán. Các bài toán của luận án bao gồm:

1. PHÁT TRIỂN NỀN TẢNG XỬ LÝ PHÂN TÁN: Chúng tôi đã nghiên cứu và phát triển nền tảng xử lý phân tán DDLF, cho phép triển khai ứng dụng phân tán bất kỳ, bao gồm huấn luyện DNN với tập dữ liệu lớn và phức tạp một cách hiệu quả.

2. LẬP CHỈ MỤC PHÂN TÁN: Chúng tôi đã nghiên cứu và phát triển cấu trúc DKD-Tree để lập chỉ mục phân tán tập vector nhiều chiều qui mô lớn, giúp tìm kiếm nhanh vector tương đồng với vector cho trước.

3. PHÁT TRIỂN MQAS THEO TIẾP CẬN PHÂN TÁN: Chúng tôi đã thiết kế và phát triển MQAS dựa trên mạng thông tin trên nền tảng DDLF. Hệ thống có khả năng trả lời các câu hỏi nhiều bước thông qua việc sử dụng các mối quan hệ giữa các thực thể trên mạng thông tin. Tuy nhiên, việc tạo template và xác định quan hệ mới dựa vào multi-hop meta-paths vẫn còn được thực hiện thủ công. Chúng tôi sẽ tiếp tục nghiên cứu để tự động hóa những công việc này.

6.2 Hướng phát triển

Tiếp tục với những kết quả đạt được, chúng tôi tiếp tục nghiên cứu để giải quyết các vấn đề còn lại. Cụ thể như sau:

1. Hiện thực việc phân tán model DNN.

2. Tự động hóa việc xác định quan hệ mới dựa vào multi-hop meta-paths.

3. Tạo mạng thông tin từ văn bản.

Đây là các vấn đề rất quan trọng nhưng không đơn giản. Nếu giải quyết được, chúng tôi có thể xử lý được lược đồ mạng lớn và tự động hóa hoàn toàn quá trình xây dựng MQAS. Hơn nữa, MQAS có thể được bổ sung tri thức thuộc mọi lĩnh vực một cách dễ dàng và không hạn chế, làm cho nó ngày càng uyên thâm. Nhờ tốc độ xử lý siêu nhanh và khả năng ghi nhớ tri thức gần như vô hạn của hệ thống máy tính, chắc chắn trong tương lai, MQAS có thể trả lời hầu hết các câu hỏi do người dùng đặt ra.

KẾT QUẢ ĐẠT ĐƯỢC

Trong quá trình nghiên cứu và giải quyết các bài toán đặt ra của luận án, chúng tôi đã đạt được các kết quả, như sau:

[1] Trung Phan, Phuc Do; *"Improving the shortest path finding algorithm in Apache Spark GraphX"*; ICMLSC '18: Proceedings of the 2nd International Conference on Machine Learning and Soft Computing, 2018.

[2] Trung Phan, Phuc Do; *"Combining Apache Spark & OrientDb to Find the Influence of a Scientific Paper in a Citation Network"*; Conference: 2018 10th International Conference on Knowledge and Systems Engineering (KSE), 2018.

[3] Phuc Do, Phu Pham, Trung Phan, Thuc Nguyen; *"T-MPP: A Novel Topic-Driven Meta-path-Based Approach for Co-authorship Prediction in Large-Scale Content-Based Heterogeneous Bibliographic Network in Distributed Computing Framework by Spark"*; International Conference on Intelligent Computing & Optimization, ICO 2018: Intelligent Computing & Optimization, 2018.

[4] Phuc Do, Trung Phan, Huong Duong; *"DM-Tree: A Novel Indexing Method for Finding Similarities in Large Vector Sets"*; International Journal of Advanced Computer Science and Applications(IJACSA) - The SAI, 2020 (Scopus – Q3 - IF: 1.8).

[5] Phuc Do, Trung Phan; *"Distributed M-Tree for Similarity Search in Large Multimedia Database on Apache Spark"*; Chapter in "Handbook of Research on Multimedia Cyber Security IGI Global, 2020.

[6] Phuc Do, Phu Pham, Trung Phan; *"Some Research Issues of Harmful and Violent Content Filtering for Social Networks in the Context of Large-Scale and Streaming Data with Apache Spark"*; Chapter in "Recent Advances in Security, Privacy, and Trust for Internet of Things (IoT) and Cyber-Physical Systems (CPS) Taylor & Francis Group, 2020.

[7] Phuc Do, Trung Phan, Hung Le, Brij Gupta; *"Building a Knowledge Graph by using Cross Lingual Transfer Method and Distributed MinIE Algorithm on Apache Spark"*; Neural Computing and Applications - Springer, 2020 (SCIE – Q1 - IF: 5.102).

[8] Trung Phan, Phuc Do; *"Building a Vietnamese Question Answering System Based on Knowledge Graph and Distributed CNN"*; Neural Computing and Applications - Springer, 2021 (SCIE – Q1 - IF: 5.102).

[9] Truong Phan Ho Viet, Trung Phan, Phuc Do; *"Using Bert And Meta-Path To Improve Question Answering"*; Hội nghị KHCN Quốc gia lần thứ XIV về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin, FAIR XIV-2021, TP HCM, ngày 23-24/12/2021; http://dx.doi.org/10.15625/vap.2021.0096, 2021.

[10] Trung Phan, Phuc Do; *"Một Hệ Thống Mới Để Trả Lời Câu Hỏi Nhiều Bước Dựa Trên Đồ Thị Tri Thức Lớn Theo Tiếp Cận Phân Tán"*; Hội nghị KHCN Quốc gia lần thứ XV về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin, FAIR XV-2022, Hà Nội, ngày 03-04/11/2022, 2022.

[11] Trung Phan, Phuc Do; *"A Novel Framework to Enhance the Performance of Training Distributed Deep Neural Networks"*; Intelligent Data Analysis, IOS Press, 2023 (SCIE – Q3 - IF: 1.321).

[12] Trung Phan, Phuc Do; *"DKD-Tree: Phương Pháp Lập Chỉ Mục Phân Tán Tập Vector Nhiều Chiều Qui Mô Lớn"*; Hội nghị KHCN Quốc gia lần thứ XVI về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin, FAIR XVI-2023, Đà Nẵng, ngày 28/09/2023, 2023.

TÀI LIỆU THAM KHẢO

[1] C Shi, Y Li, J Zhang, Y Sun **and** P. S. Yu, "A survey of heterogeneous information network analysis," *IEEE Transactions on Knowledge and Data Engineering*, **jourvol** 29, **number** 1, **pages** 17–37, 2017. DOI: 10.1109/TKDE.2016.2598561.

[2] Y. Sun, J. Han, X. Yan, P. S. Yu **and** T. Wu, "PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks," *Proc. VLDB Endow.*, **jourvol** 4, **number** 11, 992–1003, **august** 2011, ISSN: 2150-8097. DOI: 10.14778/3402707.3402736. **url**: https://doi.org/10.14778/3402707.3402736.

[3] S. Sakr, F. M. Orakzai, I. Abdelaziz **and** Z. Khayyat, *Large-Scale Graph Processing Using Apache Giraph*, 1st. Springer Publishing Company, Incorporated, 2017, ISBN: 3319474308.

[4] N. Upadhyay, P. Patel, U. Cheramangalath **and** Y. N. Srikant, "Large Scale Graph Processing in a Distributed Environment," **in***Euro-Par 2017: Parallel Processing Workshops* D. B. Heras **andothers**, **editors**, Cham: Springer International Publishing, 2018, **pages** 465–477, ISBN: 978-3-319-75178-8.

[5] M. Li **andothers**, "Scaling Distributed Machine Learning with the Parameter Server," **in***Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* **jourser** OSDI'14, Broomfield, CO: USENIX Association, 2014, 583–598, ISBN: 9781931971164.

[6] M. Li, D. G. Andersen, A. J. Smola **and** K. Yu, "Communication Efficient Distributed Machine Learning with the Parameter Server," **in***Advances in Neural Information Processing Systems* Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence **and** K. Q. Weinberger, **editors**, **volume** 27, Curran Associates, Inc., 2014. **url**: https://proceedings.neurips.cc/paper/2014/file/1ff1de774005f8da13f42943881c655f-Paper.pdf.

[7] M. Abadi **andothers**, *TensorFlow: A system for large-scale machine learning*, 2016. arXiv: 1605.08695 [cs.DC].

[8] A. Harlap **andothers**, *PipeDream: Fast and Efficient Pipeline Parallel DNN Training*, 2018. arXiv: 1806.03377 [cs.DC].

[9] P. Moritz **andothers**, *Ray: A Distributed Framework for Emerging AI Applications*, 2018. arXiv: 1712.05889 [cs.DC].

[10] A. Wang, X. Jia, L. Jiang, J. Zhang, Y. Li **and** W. Lin, "Whale: A Unified Distributed Training Framework," *CoRR*, **jourvol** abs/2011.09208, 2020. arXiv: 2011.09208. **url**: https://arxiv.org/abs/2011.09208.

[11] Apache Software Foundation. "Apache Hadoop." Accessed: 2023-04-10. (2023), **url**: https://hadoop.apache.org/.

[12] K. Shvachko, H. Kuang, S. Radia **and** R. Chansler, "The Hadoop Distributed File System," **in***2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* 2010, **pages** 1–10. DOI: 10.1109/MSST.2010.5496972.

[13] Databricks, "A Gentle Introduction To Apache Spark," *Communication*, 2017.

[14] A. d. S. Veith **and** M. D. de Assuncao, "Apache Spark," **in***Encyclopedia of Big Data Technologies* 2019. DOI: 10.1007/978-3-319-77525-8_37.

[15] E. Shaikh, I. Mohiuddin, Y. Alufaisan **and** I. Nahvi, "Apache Spark: A Big Data Processing Engine," **in***2019 2nd IEEE Middle East and North Africa COMMunications Conference, MENACOMM 2019* 2019, ISBN: 9781728136875. DOI: 10.1109/MENACOMM46666.2019.8988541.

[16] I. Cloudera, *Spark Guide*. 2017, ISBN: 1650362048.

[17] P. Zecevic **and** M. Bonaci, *Spark in Action*. Manning Publications Co, 2017, ISBN: 9781617292606.

[18] R. Rajak, "Cluster Computing: Emerging Technologies, Benefits, Architecture, Tools and Applications," *International Journal of Advanced Science and Technology*, **jourvol** 29, **number** 04, **pages** 4008 –, 2020. **url**: http://sersc.org/journals/index.php/IJAST/article/view/24569.

[19] F. Es-Sabery **and** A. Hair, "Big Data Solutions Proposed for Cluster Computing Systems Challenges: A Survey," **in***Proceedings of the 3rd International Conference on Networking, Information Systems & Security* **jourser** NISS2020, Marrakech, Morocco: Association for Computing Machinery, 2020, ISBN: 9781450376341. DOI: 10.1145/3386723.3387826. **url**: https://doi.org/10.1145/3386723.3387826.

[20] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine **and** S. Khan, "A Survey of Distributed Data Stream Processing Frameworks," *IEEE Access*, **jourvol** 7, **pages** 154 300–154 316, 2019.

[21] L. Deng **and** D. Yu, "Deep Learning: Methods and Applications," Microsoft, techreport MSR-TR-2014-21, 2014. **url**: https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/.

[22] D. Lei, X. Chen **and** J. Zhao, "Opening the black box of deep learning," **may** 2018. arXiv: 1805.08355v1 [cs.LG].

[23] D. T. Chang, "Concept-Oriented Deep Learning," **june** 2018. arXiv: 1806.01756v1 [cs.AI].

[24] M. Naumov **andothers**, *Deep Learning Training in Facebook Data Centers: Design of Scale-up and Scale-out Systems*, 2020. arXiv: 2003.09518 [cs.DC].

[25] SparkTeam, *Apache Spark FAQ*, https://spark.apache.org/faq.html, accessed 6/10/2021, 2021.

[26] O. G. Yalçın, *Top 5 Deep Learning Frameworks to Watch in 2021 and Why TensorFlow*, https://towardsdatascience.com/top-5-deep-learning-frameworks-to-watch-in-2021-and-why-tensorflow-98d8d6667351, accessed 6/8/2021.

[27] M. Langer, Z. He, W. Rahayu **and** Y. Xue, "Distributed Training of Deep Learning Models: A Taxonomic Perspective," *IEEE Transactions on Parallel and Distributed Systems*, **jourvol** 31, **number** 12, 2802–2818, 2020, ISSN: 2161-9883. DOI: 10.1109/tpds.2020.3003307. **url**: http://dx.doi.org/10.1109/TPDS.2020.3003307.

[28] A. Sharma, "Guided parallelized stochastic gradient descent for delay compensation," *CoRR*, **jourvol** abs/2101.07259, 2021. arXiv: 2101.07259. **url**: https://arxiv.org/abs/2101.07259.

[29] K. S. Chahal, M. S. Grover, K. Dey **and** R. R. Shah, "A Hitchhiker's Guide On Distributed Training Of Deep Neural Networks," *Journal of Parallel and Distributed Computing*, **jourvol** 137, **pages** 65–76, 2020, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2019.10.004. **url**: https://www.sciencedirect.com/science/article/pii/S0743731518308712.

[30] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Commun. ACM*, **jourvol** 18, **number** 9, 509–517, 1975, ISSN: 0001-0782. DOI: 10.1145/361002.361007. **url**: https://doi.org/10.1145/361002.361007.

[31] S. M. Omohundro, *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[32] K. L. Clarkson, "Nearest-neighbor searching and metric space dimensions," *Nearest-neighbor methods for learning and vision: theory and practice*, **pages** 15–59, 2006.

[33] Apache Software Foundation. "Unified engine for large-scale data analytics." Accessed: 2023-04-10. (2023), **url**: https://spark.apache.org/.

[34] J. S. Damji, B. Wenig **and** T. Das, *Learning Spark: Lightning-Fast Data Analytics*. O'Reilly Media, 2020.

[35] T. Phan **and** P. Do, "A novel framework to enhance the performance of training distributed deep neural networks," *Intelligent Data Analysis*, **jourvol** 27, **number** 3, **pages** 753–768, 2023, ISSN: 1571-4128. DOI: 10.3233/IDA-226710.

[36] M. Aly, M. E. Munich **and** P. Perona, "Distributed Kd-Trees for Retrieval from Very Large Image Collections," **in***Proceedings of the British Machine Vision Conference (BMVC)* 2011. **url**: https://api.semanticscholar.org/CorpusID:881126.

[37] D. Wehr **and** R. Radkowski, "Parallel kd-Tree Construction on the GPU with an Adaptive Split and Sort Strategy," *International Journal of Parallel Programming*, **jourvol** 46, **number** 6, **pages** 1139–1156, 2018, ISSN: 1573-7640. DOI: 10.1007/s10766-018-0571-0. **url**: https://doi.org/10.1007/s10766-018-0571-0.

[38] A. Chakravorty, W. S. Cleveland **and** P. J. Wolfe, *Scalable k-d trees for distributed data*, 2022. arXiv: 2201.08288 [cs.DS].

[39] G. Blokdyk, *Hadoop Distributed File System Third Edition*, 3rd. Emereo Publishing, 2018, ISBN: 9780655494171.

[40] M. Elkawkagy **and** H. Elbeh, "High performance hadoop distributed file system," *International Journal of Networked and Distributed Computing*, **jourvol** 8, **number** 3, **pages** 119–123, 2020, ISSN: 22117946. DOI: 10.2991/ijndc.k.200515.007.

[41] R. Rajak, "Cluster Computing: Emerging Technologies, Benefits, Architecture, Tools and Applications," *International Journal of Advanced Science and Technology*, **jourvol** 29, **number** 04, **pages** 4008 –, 2020. **url**: http://sersc.org/journals/index.php/IJAST/article/view/24569.

[42] A. Mishra, "Amazon S3," **in***Machine Learning in the AWS Cloud* John Wiley & Sons, Ltd, 2019, **chapter** 9, **pages** 181–200, ISBN: 9781119556749. DOI: 10.1002/9781119556749.ch9.

[43] A. Lakshman **and** P. Malik, "Cassandra - A decentralized structured storage system," *Operating Systems Review (ACM)*, **jourvol** 44, **number** 2, **pages** 35–40, 2010, ISSN: 01635980. DOI: 10.1145/1773912.1773922.

[44] K. Bohora, A. Bothe, D. Sheth **and** R. Chopade, "Backup and Recovery Mechanisms of Cassandra Database: A Review," *The Journal of Digital Forensics, Security and Law*, **jourvol** 15, **page** 5, 2021. DOI: 10.15394/jdfsl.2021.1613.

[45] P. Zezula, G. Amato, V. Dohnal **and** M. Batko, *Similarity Search: The Metric Space Approach*. Springer US, 2006. DOI: 10.1007/0-387-29151-2.

[46] P. Zezula, G. Amato, V. Dohnal **and** M. Batko, *Similarity Search: The Metric Space Approach*, 1st. Springer Publishing Company, Incorporated, 2010, ISBN: 1441939725.

[47] T. Pellissier Tanon, G. Weikum **and** F. Suchanek, "YAGO 4: A Reason-able Knowledge Base," **in***The Semantic Web* A. Harth **andothers**, **editors**, Cham: Springer International Publishing, 2020, **pages** 583–596, ISBN: 978-3-030-49461-2.

[48] N. Reimers **and** I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," **in***Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing* Association for Computational Linguistics, **november** 2019. **url**: https://arxiv.org/abs/1908.10084.

[49] N. Reimers **and** I. Gurevych, "Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation," **in***Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing* Association for Computational Linguistics, **november** 2020. **url**: https://arxiv.org/abs/2004.09813.

[50] M. Zaib, W. E. Zhang, Q. Z. Sheng, A. Mahmood **and** Y. Zhang, "Conversational question answering: a survey," *Knowledge and Information Systems*, **jourvol** 64, **number** 12, **pages** 3151–3195, 2022, ISSN: 0219-3116. DOI: 10.1007/s10115-022-01744-y. **url**: https://doi.org/10.1007/s10115-022-01744-y.

[51] L. Zhang **andothers**, "A survey on complex factual question answering," *AI Open*, **jourvol** 4, **pages** 1–12, 2023, ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2022.12.003. **url**: https://www.sciencedirect.com/science/article/pii/S2666651022000249.

[52] V. Mavi, A. Jangra **and** A. Jatowt, "A Survey on Multi-hop Question Answering and Generation," *CoRR*, **jourvol** abs/2204.0, 2022. arXiv: 2204.09140.

[53] L. S. Zettlemoyer **and** M. Collins, "Learning context-dependent mappings from sentences to logical form," **in***Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP* Suntec, Singapore: Association for Computational Linguistics, 2009, **page** 976. DOI: 10.3115/1690219.1690283. **url**: https://aclanthology.org/P09-1110.

[54] C. Liang, J. Berant, Q. Le, K. D. Forbus **and** N. Lao, "Neural symbolic machines: Learning semantic parsers on freebase with weak supervision," **in***ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)* **volume** 1, Association for Computational Linguistics (ACL), 2017, **pages** 23–33, ISBN: 9781945626753. DOI: 10.18653/v1/P17-1003. arXiv: 1611.00020.

[55] J. Cheng, S. Reddy, V. Saraswat **and** M. Lapata, "Learning an executable neural semantic parser," *Computational Linguistics*, **jourvol** 45, **number** 1, **pages** 59–94, 2019, ISSN: 15309312. DOI: 10.1162/coli_a_00342. arXiv: 1711.05066. **url**: https://doi.org/10.1162/coli%5C_a%5C_00342.

[56] Y. Su **and** X. Yan, "Cross-domain semantic parsing via paraphrasing," **in***EMNLP 2017 - Conference on Empirical Methods in Natural Language Processing, Proceedings* 2017, **pages** 1235–1246, ISBN: 9781945626838. DOI: 10.18653/v1/d17-1127. arXiv: 1704.05974.

[57] X. Wu **and** X. Zhu, "Ontology Reasoning for Multi-hop Question Answering," *ACM Transactions on Intelligent Systems and Technology (TIST)*, **jourvol** 12, **number** 3, **pages** 1–24, 2021. DOI: 10.1145/3428569.

[58] G. Xiao **and** J. Corman, "Ontology-Mediated SPARQL Query Answering over Knowledge Graphs," *Big Data Research*, **jourvol** 23, **page** 100 177, 2021, ISSN: 22145796. DOI: 10.1016/j.bdr.2020.100177.

[59] Y. Sun, L. Zhang, G. Cheng **and** Y. Qu, "SPARQA: Skeleton-based semantic parsing for complex questions over knowledge bases," **in***AAAI 2020 - 34th AAAI Conference on Artificial Intelligence* 2020, **pages** 8952–8959, ISBN: 9781577358350. DOI: 10.1609/aaai.v34i05.6426. arXiv: 2003.13956.

[60] A. Ait-Mlouk **and** L. Jiang, "KBot: A Knowledge Graph Based ChatBot for Natural Language Understanding over Linked Data," *IEEE Access*, **jourvol** 8, **pages** 149 220–149 230, 2020, ISSN: 21693536. DOI: 10.1109/ACCESS.2020.3016142.

[61] F. Sovrano, M. Palmirani **and** F. Vitali, "Legal knowledge extraction for knowledge graph based question-answering," **in***Frontiers in Artificial Intelligence and Applications* **volume** 334, 2020, **pages** 143–153, ISBN: 9781643681504. DOI: 10.3233/FAIA200858.

[62] Y. Lan **and** J. Jiang, "Query Graph Generation for Answering Multi-hop Complex Questions from Knowledge Bases," **in***ACL* 2020, **pages** 969–974. DOI: 10.18653/v1/2020.acl-main.91.

[63] H. Ren **andothers**, "LEGO: Latent Execution-Guided Reasoning for Multi-Hop Question Answering on Knowledge Graphs," **in***International Conference on Machine Learning* 2021, **pages** 8959–8970. **url**: http://github.com/snap-stanford/lego..

[64] L. Dong, F. Wei, M. Zhou **and** K. Xu, "Question answering over freebase with multi-column convolutional neural networks," **in***ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference* **volume** 1, 2015, **pages** 260–269, ISBN: 9781941643723. DOI: 10.3115/v1/p15-1026.

[65] M. Yu, W. Yin, K. S. Hasan, C. dos Santos, B. Xiang **and** B. Zhou, "Improved neural relation detection for knowledge base question answering," **in***ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)* **volume** 1, 2017, **pages** 571–581, ISBN: 9781945626753. DOI: 10.18653/v1/P17-1053. arXiv: 1704.06194.

[66] Z. Xu, H. T. Zheng, Z. Fu **and** W. Wang, "Enhancing Question Understanding and Representation for Knowledge Base Relation Detection," *Proceedings - IEEE International Conference on Data Mining, ICDM*, **jourvol** 2018-Novem, **pages** 1362–1367, 2018, ISSN: 15504786. DOI: 10.1109/ICDM.2018.00186.

[67] A. Bordes, J. Weston **and** N. Usunier, "Open question answering with weakly supervised embedding models," **in***Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **volume** 8724 LNAI, 2014, **pages** 165–180, ISBN: 9783662448472. DOI: 10.1007/978-3-662-44848-9_11. arXiv: 1404.4326.

[68] M. C. Yang, N. Duan, M. Zhou **and** H. C. Rim, "Joint relational embeddings for knowledge-based question answering," **in***EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* 2014, **pages** 645–650, ISBN: 9781937284961. DOI: 10.3115/v1/d14-1071.

[69] F. Becattini **and** T. Uricchio, "Memory Networks," **in***Proceedings of the 30th ACM International Conference on Multimedia* **jourser** MM '22, New York, NY, USA: Association for Computing Machinery, 2022, **pages** 7380–7382, ISBN: 9781450392037. DOI: 10.1145/3503161.3546972. **url**: https://doi.org/10.1145/3503161.3546972.