ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**HOÀNG ĐỨC THỌ**

**KHAI THÁC MẪU TUẦN TỰ**

**TỪ CƠ SỞ DỮ LIỆU CHUỖI**

**TRÊN HADOOP MAPREDUCE**

**LUẬN VĂN THẠC SĨ: CÔNG NGHỆ THÔNG TIN**

**TP. Hồ Chí Minh – 2017**

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**HOÀNG ĐỨC THỌ**

**KHAI THÁC MẪU TUẦN TỰ**

**TỪ CƠ SỞ DỮ LIỆU CHUỖI**

**TRÊN HADOOP MAPREDUCE**

**Chuyên ngành: Hệ Thống Thông Tin**

**Mã số chuyên ngành: 60480104**

**LUẬN VĂN THẠC SĨ: CÔNG NGHỆ THÔNG TIN**

**NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS. TS. LÊ HOÀI BẮC**

**TP. Hồ Chí Minh – 2017**

**LỜI CẢM ƠN**

Tôi xin gởi lời cảm ơn chân thành đến Ban giám hiệu trường Trường Đại Học Khoa Học Tự Nhiên, ĐHQG TP.HCM; các Thầy, Cô thuộc Khoa Công Nghệ Thông Tin; các cán bộ phòng Đào Tạo Sau Đại Học đã tạo điều kiện thuận lợi, tận tâm giảng dạy bằng tất cả sự nhiệt tình và hướng dẫn tận tình các thủ tục, hồ sơ cần thiết trong suốt thời gian tôi tham gia học tập tại đây.

Đặc biệt, tôi xin gửi lời biết ơn chân thành và sâu sắc nhất đến thầy PSG. TS. Lê Hoài Bắc đã tận tình, chỉ bảo, hướng dẫn và giúp đỡ tôi trong thời gian làm luận văn vừa qua.

Tôi xin cảm ơn gia đình, bạn bè và đồng nghiệp đã tạo điều kiện tốt nhất cho tôi học tập, hỗ trợ, động viên tôi trong thời gian thực hiện luận văn này.

Hoàng Đức Thọ

**MỤC LỤC**

[MỞ ĐẦU 1](#_Toc491803657)

[Chương 1. TỔNG QUAN 3](#_Toc491803658)

[1.1. Giới thiệu 3](#_Toc491803659)

[1.2. Một số ứng dụng trong khai thác chuỗi 4](#_Toc491803660)

[1.3. Mục tiêu và phương pháp nghiên cứu 6](#_Toc491803661)

[1.4. Đóng góp của luận văn 7](#_Toc491803662)

[Chương 2. CƠ SỞ LÝ THUYẾT 8](#_Toc491803663)

[2.1. Phát biểu bài toán khai thác CSDL chuỗi 8](#_Toc491803664)

[2.2. Các khái niệm về chuỗi dữ liệu 8](#_Toc491803665)

[2.3. Khai thác chuỗi 11](#_Toc491803666)

[2.4. Thuật toán SPAM 14](#_Toc491803667)

[2.5. Cấu trúc CMAP và thuật toán CM-SPAM 17](#_Toc491803668)

[2.6. Cấu trúc vector bit động - DBV 18](#_Toc491803669)

[2.7. Mô hình tính toán phân tán Hadoop MapReduce 20](#_Toc491803670)

[2.7.1. Mô hình MapReduce 20](#_Toc491803671)

[2.7.2. Nền tảng Hadoop 21](#_Toc491803672)

[2.8. Thuật toán SPAMC 23](#_Toc491803673)

[Chương 3. KỸ THUẬT XỬ LÝ SONG SONG TRONG KHAI THÁC MẪU TUẦN TỰ 27](#_Toc491803674)

[3.1. Giới thiệu 27](#_Toc491803675)

[3.2. Biểu diễn dữ liệu 27](#_Toc491803676)

[3.3. Thuật toán DSPDBV 28](#_Toc491803677)

[3.3.1. Phương pháp khai thác 29](#_Toc491803678)

[3.3.2. Khai thác phân tán sử dụng MapReduce 31](#_Toc491803679)

[3.3.3. Phân tán quá trình chuyển đổi dữ liệu 32](#_Toc491803680)

[3.3.4. Phân tán quá trình khai thác mẫu tuần tự phổ biến 35](#_Toc491803681)

[3.4. Đánh giá 39](#_Toc491803682)

[Chương 4. KẾT QUẢ THỰC NGHIỆM 41](#_Toc491803683)

[4.1. Tập dữ liệu 41](#_Toc491803684)

[4.2. Phương pháp thực nghiệm 41](#_Toc491803685)

[4.3. Kiểm tra tính chính xác 42](#_Toc491803686)

[4.4. So sánh với SPAMC 43](#_Toc491803687)

[4.5. Tính mở rộng (Scalability and Extensibility) 44](#_Toc491803688)

[4.6. Kết luận 45](#_Toc491803689)

[Chương 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 46](#_Toc491803690)

[5.1. Kết luận 46](#_Toc491803691)

[5.2. Hướng phát triển 47](#_Toc491803692)

[DANH MỤC CÔNG TRÌNH CỦA TÁC GIẢ 48](#_Toc491803693)

[TÀI LIỆU THAM KHẢO 49](#_Toc491803694)

**DANH MỤC CÁC BẢNG, BIỂU MẪU**

[Bảng 2‑1 Ví dụ CSDL tuần tự SDB 9](#_Toc491803645)

[Bảng 2‑2 Cấu trúc CMAP của SDB trong CM-SPAM với minSup = 50% 18](#_Toc491803646)

[Bảng 2‑3 Ví dụ bảng vector bit 16 byte 18](#_Toc491803647)

[Bảng 3‑1 Thuật toán DSPDBV 31](#_Toc491803648)

[Bảng 3‑2 Thuật toán DistributedDBVConversion 33](#_Toc491803649)

[Bảng 3‑3 Thuật toán DistributedSequentialPatternMining 36](#_Toc491803650)

[Bảng 3‑4 Thuật toán DBVPatternExtension 38](#_Toc491803651)

[Bảng 3‑5 Thuật toán OutputDBVPattern 39](#_Toc491803652)

[Bảng 4‑1 Đặc điểm các bộ dữ liệu thực 41](#_Toc491803653)

[Bảng 4‑2 Mô tả tham số bộ dữ liệu tổng hợp 41](#_Toc491803654)

[Bảng 4‑3 Thông tin cấu hình máy trong Hadoop Cluster 42](#_Toc491803655)

[Bảng 4‑4 Thống kê mẫu tuần tự khai thác trên bộ dữ thực 43](#_Toc491803656)

**DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ**

[Hình 1‑1 Ví dụ một mẫu chuỗi DNA 4](#_Toc491803623)

[Hình 1‑2 Ví dụ một mẫu chuỗi Protein 5](#_Toc491803624)

[Hình 1‑3 Ví dụ chuỗi truy cập trang web 5](#_Toc491803625)

[Hình 1‑4 Ví dụ thông tin mua hàng của khách hàng 6](#_Toc491803626)

[Hình 2‑1 Cấu trúc dữ liệu bảng bit dạng dọc của SDB trong SPAM 14](#_Toc491803627)

[Hình 2‑2 Minh họa cây thứ tự từ điển trong SPAM 15](#_Toc491803628)

[Hình 2‑3 Minh hoạ bước S-Step (a) và bước I-Step (b) trong SPAM 16](#_Toc491803629)

[Hình 2‑4 Chuyển đổi bảng vector bit trong Bảng 2-3 sang DBV 19](#_Toc491803630)

[Hình 2‑5 Biểu diễn dữ liệu dùng DBV 20](#_Toc491803631)

[Hình 2‑6 Mô hình thực thi MapReduce của Google 21](#_Toc491803632)

[Hình 2‑7 Kiến trúc Hadoop 22](#_Toc491803633)

[Hình 2‑8 Ví dụ minh họa thực thi Hadoop MapReduce 23](#_Toc491803634)

[Hình 2‑9 Quá trình khai thác của thuật toán SPAMC 24](#_Toc491803635)

[Hình 2‑10 Minh họa DHT của SDB với minSup = 50% 25](#_Toc491803636)

[Hình 3‑1 Cấu trúc DBVItem của các sự kiện có trong SDB 28](#_Toc491803637)

[Hình 3‑2 Minh họa mở rộng chuỗi của DBVPattern 28](#_Toc491803638)

[Hình 3‑3 Minh họa mở rộng tập sự kiện của DBVPattern 28](#_Toc491803639)

[Hình 3‑4 Quá trình khai thác của thuật toán DSPDBV 29](#_Toc491803640)

[Hình 3‑5 Kết quả chuyển đổi dữ liệu CSDL SDB 35](#_Toc491803641)

[Hình 4‑1 So sánh thời gian thực thi giữa DSPDBV và SPAMC 44](#_Toc491803642)

[Hình 4‑2 Tác động của tập dữ liệu đối với hiệu quả khai thác 44](#_Toc491803643)

[Hình 4‑3 Tác động của số lượng máy đối với hiệu quả khai thác 45](#_Toc491803644)

**DANH MỤC CÁC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **STT** | **Từ viết tắt** | **Ý nghĩa** |
| 01 | CSDL | Cơ sở dữ liệu |
| 02 | SDB | Sequence Database - Cơ sở dữ liệu chuỗi |
| 03 | SID | Sequence Identify - Định danh chuỗi |
| 04 | DBV | Dynamic Bit Vector - Vector bit động |
| 05 | CMAP | Co-occurrence Map - Ánh xạ đồng xuất hiện |
| 06 | HDFS | Hadoop Distributed File System |
| 07 | DBVPattern | Biểu diễn mẫu tuần tự theo vector bit động |
| 08 | DBVItem | Biểu diễn sự kiện theo vector bit động |
| 09 | DHT | Distributed Hash Table |
| 10 | TID | Transaction Identify - Định danh giao dịch |

# MỞ ĐẦU

Trong thời đại công nghệ phát triển vượt bậc và đóng vai trò quan trọng trong cuộc sống hiện nay, khối lượng dữ liệu thu thập được từ các ứng dụng ngày càng nhiều. Tất cả thông tin liên quan đến cuộc sống hằng ngày của con người đều có khả năng được thể hiện và lưu trữ, thông qua các ứng dụng máy tính, dưới dạng dữ liệu. Vấn đề là làm thế nào khai thác được các dữ liệu này một cách có hiệu quả nhằm tìm kiếm những thông tin quý giá, những tri thức cần thiết tiềm ẩn trong khối lượng dữ liệu khổng lồ đó. Do đó, lĩnh vực khai thác dữ liệu trở thành một trong những lĩnh vực nghiên cứu đóng vai trò quan trọng trong thực tiễn cuộc sống và trong khoa học.

Trong kinh tế, ta có thể tìm ra những hành vi phổ biến của một đối tượng hoặc nhóm các đối tượng từ đó xác định được mối liên hệ giữa chúng. Ví dụ như trong siêu thị: có thể phân tích thói quen mua sắm của khách hàng. Trong các lĩnh vực địa chất, khí hậu: có thể giúp dự báo, phát hiện các thảm hoạ thiên nhiên. Trong lĩnh vực thương mại điện tử: giúp khai thác hành vi sử dụng Web, được sử dụng trong việc tổ chức lại cấu trúc trang Web cho phù hợp với ngữ cảnh sử dụng của người dùng. Trong lĩnh vực sinh học, y học: những mẫu phổ biến này có thể được dùng để phát hiện các bất thường của cấu trúc Protein, DNA. Hay trong lĩnh vực phần mềm: giúp dự báo những đoạn mã lệnh không nhất quán hay bất thường, cũng như trong lĩnh vực phân tích cụm từ trong xử lý tài liệu văn bản.

Vấn đề khai thác mẫu tuần tự từ cơ sở dữ liệu (CSDL) chuỗi đã có nhiều nhóm tác giả nghiên cứu. Cho đến nay, nhiều phương pháp tiếp tục được đề xuất nhằm tăng cường hiệu quả khai thác dữ liệu dạng này. Tuy nhiên, điểm hạn chế của các phương pháp hiện nay là chỉ mới dừng lại ở việc khai thác cục bộ, xử lý trên máy tính đơn và dữ liệu thực thi có kích thước tương đối nhỏ. Điều này không còn phù hợp với bối cảnh hiện tại. Do đó, xử lý song song là vấn đề cần thiết trong bài toán khai thác chuỗi phổ biến khi mà kích thước các tập dữ liệu ngày càng tăng, việc khai thác cục bộ hay xử lý trên máy tính đơn bộc lộ nhiều hạn chế.

Hadoop là một khung làm việc (framework) dùng để chạy những ứng dụng trên một cụm (cluster) máy tính lớn, được xây dựng trên những phần cứng thông thường. Hadoop MapReduce cho phép xử lý phân tán (distributed processing) các tập dữ liệu lớn trên các cụm máy tính (clusters of computers) thông qua mô hình lập trình đơn giản. Nghiên cứu mô hình Hadoop MapReduce nhằm áp dụng vào bài toán khai thác mẫu tuần tự từ CSDL chuỗi là vấn đề mang tính cần thiết trong bối cảnh hiện nay.

Xuất phát từ tình hình thực tế, luận văn tập trung nghiên cứu giải pháp cho bài toán khai thác mẫu tuần tự từ CSDL chuỗi trên mô hình Hadoop MapReduce. Cụ thể, luận văn sẽ tập trung vào vấn đề song song hoá quá trình khai thác mẫu tuần tự từ CSDL chuỗi dựa trên mô hình Hadoop MapReduce. Mục tiêu là thuật toán khai thác có khả năng thực thi trên dữ liệu có kích thước lớn với việc tối ưu hiệu quả thời gian.

Ngoài phần mở đầu, danh mục tài liệu tham khảo và phụ lục, bố cục của luận văn gồm có 05 chương:

* Chương 1: Giới thiệu tổng quan về bài toán khai thác cơ sở dữ liệu chuỗi; hướng tiếp cận và đóng góp của luận văn.
* Chương 2: Trình bày cơ sở lý thuyết và các công trình nghiên cứu đã công bố liên quan đến luận văn.
* Chương 3: Trình bày chi tiết phương pháp đề xuất cho bài toán khai thác chuỗi từ cơ sở dữ liệu chuỗi trên mô hình Hadoop MapReduce.
* Chương 4: Trình bày kết quả thực nghiệm của phương pháp đề xuất
* Chương 5: Trình bày kết luận và hướng phát triển của luận văn.

# TỔNG QUAN

## Giới thiệu

Việc khai thác luật tuần tự (sequential rule mining) từ các mẫu tuần tự (frequent sequence hay sequential pattern) trong CSDL chuỗi (Sequence Database) đóng một vai trò quan trọng trong thực tiễn cuộc sống và trong khoa học.

Trong kinh tế, ta có thể tìm ra những hành vi phổ biến của một đối tượng hoặc nhóm các đối tượng từ đó xác định mối liên hệ giữa chúng. Ví dụ như trong siêu thị, có thể phân tích thói quen mua sắm của khách hàng [1]. Trong các lĩnh vực địa chất, khí hậu: có thể giúp dự báo, phát hiện các thảm hoạ thiên nhiên [2]. Trong lĩnh vực thương mại điện tử: giúp khai thác hành vi sử dụng Web được sử dụng trong việc tổ chức lại cấu trúc trang Web cho phù hợp với ngữ cảnh sử dụng của người dùng [3]. Trong lĩnh vực sinh học, y học: những mẫu phổ biến này có thể được dùng để phát hiện các bất thường của cấu trúc Protein, DNA [4]. Hay trong lĩnh vực phần mềm: giúp dự báo những đoạn mã lệnh không nhất quán hay bất thường [5], cũng như trong lĩnh vực phân tích cụm từ trong xử lý tài liệu văn bản [6].

Chính vì những ứng dụng rất thiết thực của việc khai thác CSDL chuỗi, nhiều nhà khoa học đã sớm tập trung vào nghiên cứu và có những kết quả rất khả quan.

Một cách tổng quát, CSDL chuỗi thể hiện các sự kiện xảy ra theo thứ tự thời gian. Việc khai thác luật tuần tự từ các mẫu tuần tự trong CSDL chuỗi giúp hỗ trợ các tổ chức dự đoán các xu hướng, các biến đổi của dữ liệu nhằm tìm ra cách thức tổ chức, quản lý công việc tốt hơn. Khai thác CSDL chuỗi thường được chia làm hai giai đoạn chính:

(1) Giai đoạn khai thác mẫu tuần tự (sequential pattern mining): Là quá trình tìm kiếm các chuỗi phổ biến trong CSDL chuỗi. Bài toán cho trước một CSDL chuỗi và giá trị ngưỡng hỗ trợ nhỏ nhất. Yêu cầu của bài toán là tìm ra tất cả các chuỗi con có số lần xuất hiện (độ hỗ trợ) lớn hơn hoặc bằng giá trị ngưỡng hỗ trợ nhỏ nhất cho trước.

(2) Giai đoạn sinh luật từ các mẫu trong giai đoạn đầu (sequential rule mining): Là quá trình tìm ra mối quan hệ giữa các sự kiện trong CSDL chuỗi. Mối quan hệ giữa các sự kiện này được biểu thị bởi một luật có dạng X 🡪 Y, nghĩa là với một sự kiện X xảy ra trước thì sẽ có một sự kiện Y xảy ra ngay sau đó.

## Một số ứng dụng trong khai thác chuỗi

Khai thác mẫu tuần tự là lĩnh vực nghiên cứu quan trọng và có nhiều ứng dụng trong các lĩnh vực khác nhau [1], [2], [3], [4], [5], [6], [7], [8]:

**Trong lĩnh vực y tế, chăm sóc sức khỏe:** các mẫu tuần tự được khai thác từ hồ sơ bệnh án, triệu chứng, biểu hiện quan sát được; thông tin sức khỏe, các hoạt động hằng ngày của bệnh nhân. Thông tin từ các mẫu tuần tự khai thác từ mối quan hệ giữa triệu chứng và bệnh giúp bác sĩ chuẩn đoán bệnh sớm cũng như có những phương án y tế dự phòng. Mẫu tuần tự từ quá trình sử dụng thuốc đối với từng loại bệnh trên từng bệnh nhân giúp các nhà chuyên môn có những đánh giá chính xác về tác dụng thuốc cũng như liều lượng sử dụng đối với từng bệnh nhận khác nhau. Khai thác mẫu tuần tự từ hoạt động hằng ngày của bệnh nhân giúp phát hiện sớm các hành vi không phù hợp với hành vi bình thường, đưa ra các dự đoán về những vấn đề sức khỏe bất thường. Ngoài ra, kết quả khai thác có thể được sử dụng để dự đoán lượng bệnh nhân trong phân loại bệnh. Từ đó, đề xuất các quy trình phù hợp để đảm bảo rằng bệnh nhân được chăm sóc thích hợp ở đúng nơi và vào đúng thời điểm. Đồng thời, các hãng bảo hiểm chăm sóc sức khoẻ có thể phát hiện gian lận và lạm dụng của khách hàng.

**Trong sinh học:** Chuỗi dữ liệu sinh học giúp chúng ta hiểu rõ cấu trúc cũng như chức năng của các loại tế bào khác nhau, có nhiều lợi ích trong việc chuẩn đoán và chữa bệnh. Ba loại chuỗi sinh học phổ biến là chuỗi deoxyribonucleic acid (DNA), chuỗi amino acid (hay còn gọi là Peptide hoặc Protein) và ribonucleic acid (RNA). Hình 1.1 và 1.2 minh họa một mẫu chuỗi DNA và chuỗi protein.

|  |
| --- |
| GAATTCTCTGTAACACTAAGCTCTCTTCCTCAAAACCAGAGGTAGATAGAATGTGTAATAATTTACAGAATTTCTAGACTTCAACGATCTGATTTTTTAAATTTATTTTTATTTTTTCAGGTTGAGACTGAGCTAAAGTTAATCTGTGGC |

Hình ‑ Ví dụ một mẫu chuỗi DNA

Các ứng dụng khai thác chuỗi dữ liệu sinh học bao gồm tìm ra gen, suy luận về chức năng của protein, chẩn đoán bệnh, tiên lượng bệnh, điều trị bệnh, tái tạo mạng protein và tái tạo mạng tương tác. Bằng cách tìm ra chuỗi DNA có khoảng lệch, các chuỗi không xác định có thể được tìm thấy và phân loại vào họ DNA tương ứng và điều này có thể được sử dụng để nghiên cứu sâu hơn về phân tích sinh học.

|  |
| --- |
| SSQIRQNYSTEVEAAVNRLVNLYLRASYTYLSLGFYFDRDDVALEGVCHFFRELAEEKREGAERLLKMQNQRGGRALFQDLQKPSQDEWGTTPDAMKAAIVLEKSLNQALLDLHALGSAQADPHLCDFLESHFLDEEVKLIKKMGDHLTNIQRLVGSQAGLGEYLFERLTLKHD |

Hình ‑ Ví dụ một mẫu chuỗi Protein

**Trong giáo dục:** một số ứng dụng được đề xuất thông qua việc khai thác mã nguồn cũng như thông tin làm việc nhóm của sinh viên. Khai thác mẫu tuần tự từ chuỗi tuần từ được xây dựng từ các lệnh gọi hàm và lệnh điều kiển để cài đặt yêu cầu chức năng một hành vi cụ thể giúp dự báo những đoạn mã lệnh không nhất quán hay bất thường. Mẫu tuần tự từ khai thác từ dữ liệu làm việc nhóm của sinh viên có thể giúp xây dựng công cụ hỗ trợ sinh viên nhận biết sớm và hướng giải quyết vấn đề.

**Trong kinh tế, thương mại điện tử:** Khai thác mẫu tuần tự cũng có nhiều ứng dụng đáng kể trong lĩnh vực thương mại, bao gồm các số liệu thống kê đơn giản, chẳng hạn như tần suất truy cập trang, cũng như các dạng phân tích phức tạp hơn, chẳng hạn như tìm kiếm các đường đi chung thông qua một trang Web. Mẫu tuần tự giúp khai thác hành vi sử dụng trang web của người dùng, được sử dụng trong việc tổ chức lại cấu trúc trang cho phù hợp với ngữ cảnh sử dụng của người dùng, ứng dụng trong hệ thống hỗ trợ tư vấn người dùng.

|  |
| --- |
| <100,a>, <100,b>, <200,a>, <300,b>, <200,b>, <400,a>, <100,a>, <400,b>, <300,a>, <100,c>, <200,c>, <400,a>, <400,e> |

Hình ‑ Ví dụ chuỗi truy cập trang web

Khai thác thông tin hóa đơn của khách hàng trong một khoảng thời gian giúp phát hiện ra các mẫu thông tin hữu ích như thói quen mua sắm, sở thích khách hàng, cách bố trí sản phẩm trên kệ sao cho phù hợp với nhu cầu khách hàng, dự đoán nhu cầu mua sắm…

|  |
| --- |
| <223100, 05/26/06, 10am,CentralStation, {WholeMealBread,AppleJuice}>,  <225101, 05/26/06, 11am,CentralStation, {Burger,Pepsi,Banana}>,  <223100, 05/26/06, 4pm,WalMart, {Milk,Cereal,Vegetable}>,  <223100, 05/27/06, 10am,CentralStation, {WholeMealBread,AppleJuice}>,  <225101, 05/27/06, 12noon, CentralStation, {Burger,Coke,Apple}> |

Hình ‑ Ví dụ thông tin mua hàng của khách hàng

**Trong khai thác văn bản:** Dữ liệu văn bản là các chuỗi bao gồm các từ, cụm từ được sắp xếp theo thứ tự tạo thành câu có ý nghĩa. Việc khai thác mẫu tuần tự từ chuỗi văn bản cho phép chúng ta phân loại văn bản, xác định nội dung, cung cấp thông tin hữu ích trong việc phân tích phản hồi của người dùng về sản phẩm, dịch vụ của doanh nghiệp.

**Trong viễn thông:** khai thác mẫu tuần tự từ dữ liệu di chuyển của người dùng di động, giúp dự đoán vị trí sắp đi đến của người dùng sử dụng trong các ứng dụng dựa trên thông tin vị trí như quảng cáo trực tuyến; dự đoán hành vi khách hàng.

## Mục tiêu và phương pháp nghiên cứu

Mục đích nghiên cứu của luận văn là nghiên cứu các ưu và khuyết điểm của các phương pháp khai thác mẫu tuần tự từ CSDL hiện tại, từ đó đề xuất thuật toán hiệu quả để khai thác chúng. Nghiên cứu mô hình Hadoop MapReduce [9] nhằm áp dụng vào bài toán khai thác song song mẫu tuần tự từ CSDL chuỗi. Cụ thể, luận văn sẽ tập trung vào vấn đề song song hoá quá trình khai thác mẫu tuần tự từ CSDL chuỗi dựa trên mô hình Hadoop MapReduce. Mục tiêu là thuật toán khai thác có khả năng thực thi trên dữ liệu có kích thước lớn với việc tối ưu hiệu quả thời gian và không gian khai thác.

* + Nghiên cứu các cách tiếp cận của việc khai thác dữ liệu chuỗi, cùng với việc tổ chức cấu trúc dữ liệu và các kỹ thuật tối ưu trong quá trình khai thác.
  + Áp dụng và đề xuất cải tiến kỹ thuật khai thác nhằm nâng cao hơn nữa hiệu suất của quá trình khai thác. Kết hợp với kỹ thuật lập trình song song và chia nhỏ dữ liệu áp dụng trên mô hình Hadoop MapReduce.

## Đóng góp của luận văn

Nhằm giải quyết vấn đề thời gian và không gian khai thác, luận văn trình bày các cách tiếp cận và đề xuất phương pháp khai thác giúp giải quyết hiệu quả bài toán khai thác tập mẫu tuần tự từ CSDL chuỗi. Trong đó, nội dung bao gồm hai giai đoạn:

* + Giai đoạn 1: Khai thác mẫu tuần tự
  + Giai đoạn 2: Song song hóa quá trình khai thác mẫu tuần tự

Cụ thể, đối với giai đoạn 1 của luận văn, dựa trên phương pháp khai thác mẫu tuần tự gọi là SPAM [10] được đề xuất bởi Jay Ayres (2002), và phương pháp cải tiến do Fournier-Viger đề xuất năm 2014 sử dụng bảng ánh xạ đồng xuất hiện (CMAP) với tên gọi CM-SPAM [11], luận văn giữ nguyên phương pháp phát sinh mẫu ứng viên nhưng cải tiến các thức tổ chức dữ liệu từ bảng bit dạng dọc bằng cách sử dụng vector bit động được đề xuất trong thuật toán CloFS-DBV [12] bởi Tran và cộng sự (2015) giúp tối ưu không gian lưu trữ và thời gian thai thác.

Giai đoạn 2, luận văn đề xuất phương pháp khai thác mẫu tuần tự song song dựa trên mô hình xử lý phân tán Hadoop MapReduce. Mô hình Hadoop MapReduce cho phép thục hiện phân tán quá trình xử lý các tập dữ liệu lớn trên các cụm máy tính. CSDL chuỗi ban đầu được chia thành các tập cơ sở dữ liệu chuỗi nhỏ, thuật toán tiến hành khai các song song tập mẫu ứng viên từ các CSDL đã được phân cắt. Cuối cùng, thuận toán tổng hợp tất cả mẫu ứng viên và tính tổng độ hỗ trợ của mẫu ứng viên, tìm ra các mẫu tuần tự dựa trên điều kiện ngưỡng hỗ trợ tối thiểu cho trước.

Kết quả nghiên cứu của luận văn đã được chấp nhận đăng trong kỷ yếu hội nghị quốc tế chuyên ngành của tạp chí IEEE (The 9th International Conference on Knowledge and Systems Engineering - KSE 2017 - được tổ chức tại Huế vào ngày 19-21 tháng 10 năm 2017).

# CƠ SỞ LÝ THUYẾT

## Phát biểu bài toán khai thác CSDL chuỗi

Khai thác CSDL chuỗi thường được chia làm hai giai đoạn chính: (1) Giai đoạn khai thác mẫu tuần tự và (2) Giai đoạn sinh luật từ các mẫu của Giai đoạn đầu. Tuy nhiên, các thách thức của bài toán khai thác trên CSDL chuỗi là thời gian khai thác và tập luật kết quả được khai thác. Đối với thời gian, bài toán khai thác mẫu tuần tự là bài toán có độ phức tạp cao do phải tìm tất cả các hoán vị có thể có của các sự kiện trong CSDL chuỗi [13]. Do đó, tập luật được sinh từ các mẫu thường rất lớn, trong đó sẽ có tồn tại nhiều luật dư thừa.

Do vậy vấn đề chính là nghiên cứu cách thức làm giảm thời gian khai thác và không gian lưu trữ trong quá trình khai thác, cùng với việc loại bỏ luật dư thừa một cách có hiệu quả là vấn đề cần được quan tâm và phát triển.

Khai thác tập mẫu tuần tự trên dữ liệu chuỗi đã được đề cập và nghiên cứu rộng rãi kể từ khi Agrawal và Srikant đề xuất thuật toán AprioriAll [14] vào năm 1995. Thuật toán này sử dụng tính chất Apriori: “Tất cả các tập con khác rỗng của một tập sự kiện phổ biến đều phổ biến”. Nghĩa là, trong quá trình phát sinh mẫu, nếu xuất hiện một mẫu không phổ biến thì không cần phát sinh tiếp mẫu cha cho mẫu ứng viên đó. Điều này giúp cho việc loại trừ sớm ứng viên, nhằm làm giảm không gian và kiểm tra các ứng viên chắc chắn không là mẫu tuần tự phổ biến. Giai đoạn này có rất nhiều thuật toán được đề xuất theo nhiều cách tiếp cận khác nhau nhằm tăng hiệu quả khai thác.

## Các khái niệm về chuỗi dữ liệu

Cho tập *I = {i1, i2, …, in}* gồm *n* phần tử còn gọi là các sự kiện (item). Một tập sự kiện *itemset* là tập không có thứ tự khác rỗng, gồm các sự kiện. Mỗi itemset được biễu diễn trong cặp dấu ngoặc nhọn. Các dấu ngoặc nhọn được bỏ ra để đơn giản hóa ký hiệu cho các tập sự kiện với chỉ một sự kiện đơn. Ví dụ, {A, B, C} biễu diễn 1 tập sự kiện gồm 3 sự kiện là {A}, {B} và {C}. Một chuỗi *sequence*, ký hiệu *S = (e1, e2, …, em)*, là một tập có thứ tự các tập sự kiện, với mỗi *ei* (1 ≤ i ≤ m) là một tập sự kiện. Các sự kiện trong tập sự kiện được sắp xếp theo thứ tự từ điển, ký hiệu .

**CSDL chuỗi (Sequence Database)**: CSDL chuỗi, ký hiệu *SDB*, là danh sách các chuỗi, được biễu diễn dưới dạng *SDB = (s1, s2, …, s|SDB|)*, trong đó |SDB| là số lượng chuỗi trong SDB, và *si* (1 ≤ i ≤ |SDB|) là chuỗithứ i trong SDB.

|  |  |
| --- | --- |
| **SID** | **Sequence** |
| 1 | ({A},{A,B},{A,B,C}) |
| 2 | ({A,B},{A,C},{B,C}) |
| 3 | ({C},{A,C},{A,B}) |
| 4 | ({D},{A,B,D}) |

Bảng ‑ Ví dụ CSDL tuần tự SDB

Ví dụ: Bảng 2-1 mô tả CSDL chuỗi SDB có 4 chuỗi, |SDB| = 4, và 4 sự kiện phân biệt I = {A, B, C, D}. Các chuỗi có định danh lần lượt là 1, 2, 3, 4 trong cột CID, thông tin chuỗi được thể hiện trong cột Sequence. Chuỗi ({A},{A,B},{A,B,C}) gồm có 3 tập sự kiện. Tập sự kiện thứ nhất là {A}. Tập sự kiện thứ hai là {A,B} có 2 sự kiện A và B. Cuối cùng là tập sự kiện {A,B,C} có 3 sự kiện là A, B và C.

**Kích thước chuỗi (size of sequence)**: số tập sự kiện (itemset) có trong chuỗi S, ký hiệu |S|.

**Độ hỗ trợ (support)**: độ hỗ trợ của chuỗi S trong CSDL tuần tự được định nghĩa là tổng số chuỗi trong CSDL có chứa S, ký hiệu sup(S).

**Độ dài chuỗi (length of sequence)**: số sự kiện (item) có trong chuỗi S, ký hiệu l. Chuỗi có độ dài k được ký hiệu là k-sequence.

**Chuỗi con tuần tự**: một chuỗi SA=(*a1*,*a2*,…,*an*) được gọi là chuỗi con của chuỗi SB=(*b1*,*b2*,…,*bm*) nếu và chỉ nếu tồn tại dãy số 1 ≤ i1 < i2 < … < in <= m sao cho *a1*⊆*bi1*, *a2*⊆*bi2*,…, *an*⊆*bin*. Khi đó ta nói chuỗi SA là chuỗi con của SB hay chuỗi SB chứa chuỗi SA.

* + SA là **subsequence** của SB
  + SB là **supersequence** của SA

**Tiền tố (prefix)**: một chuỗi SA=(*a1*,*a2*,…,*an*) được gọi là tiền tố của chuỗi SB=(*b1*,*b2*,…,*bm*) nếu và chỉ nếu với mọi n < m và a1=b1,a2=b2,…,an=bn.

**Mẫu tuần tự (sequential pattern):** Cho trước một ngưỡng hỗ trợ tối thiểu, ký hiệu minSup, được xác định bởi người dùng. Trong đó, 0 < minSup ≤ 1. Một chuỗi S được xem là chuỗi phổ biến khi và chỉ khi độ hỗ trợ của S lớn hơn bằng ngưỡng hỗ trợ tối thiểu, sup(S) ≥ minSup. Khi đó, S được gọi là mẫu tuần tự.

**Mở rộng mẫu tuần tự từ mẫu tuần tự có độ dài 1 (extending a pattern from a 1-pattern):** Cho trước hai mẫu tuần tự α và β có độ dài 1. {tα.pα} và {tβ.pβ} lần lượt là thông tin giao dịch và vị trị của các mẫu tuần tự α và β trong SDB. Có hai hình thức mở rộng mẫu tuần tự:

* + Mở rộng tập sự kiện (itemset extension):   
    *({α, β}){tβ.pβ}, if (βα), (tα = tβ) and (pα = pβ))*
  + Mở rộng chuỗi (sequence extension):   
    *({α},{β}){tβ.pβ}, if (tα = tβ) and (pα< pβ))*

**Chuỗi con tuần tự (subsequence of sequence):** Cho trước chuỗi S. Một chuỗi con của S, ký hiệu subi,j(S) (i < j), là một dãy con liên tục từ vị trí i đến vị trí j trong chuỗi S.

**Mở rộng mẫu tuần tự từ mẫu tuần tự có độ dài k (extending a pattern from a k-pattern):** Cho trước hai mẫu tuần tự α và β có độ dài k (k > 1). u và v lần lượt là các chuỗi con tuần tự của α và β, trong đó, u=subk,k(α), and v = subk,k(β). {tα.pα} và {tβ.pβ} lần lượt là thông tin giao dịch và vị trị của các mẫu tuần tự α và β trong SDB. Có hai hình thức mở rộng mẫu tuần tự:

* + Mở rộng tập sự kiện (itemset extension):   
    *(sub1,k-1(α){u, v}){tβ.pβ},  
    if v u, tα=tβ, pα=pβ, and sub1,k-1(α)= sub1,k-1(β))*
  + Mở rộng chuỗi (sequence extension):   
    *({α}, {v}){tβ.pβ},   
    if tα=tβ, pα<pβ, and sub1,k-1(α)=sub1,k-1(β))*

Vấn đề của khai thác chuỗi phổ biến là tìm ra tất cả các mẫu tuần tự có trong CSDL tuần tự.

**Định dạng dữ liệu dạng ngang**: Là định dạng dữ liệu mà mỗi mẫu tin (entry) là một chuỗi, gắn liền với mỗi chuỗi là định danh của chuỗi (giao dịch, SID). Hình 2.1 mô tả định dạng dữ liệu dạng ngang.

**Định dạng dữ liệu dạng dọc**: Là định dạng dữ liệu mà mỗi mẫu tin chứa thông tin của một sự kiện (item). Thông tin của sự kiện bao gồm định danh chuỗi mà sự kiện ấy xảy ra và vị trí hay tập sự kiện, trong chuỗi đó, có chứa sự kiện đang xét.

Về tính hiệu quả khi tính độ hỗ trợ, dữ liệu được tổ chức theo dạng dọc có tính hiệu quả cao hơn do dễ dàng lấy được ngay các đối tượng ứng với sự kiện mà không phải duyệt lại toàn bộ CSDL. Tuy nhiên, CSDL ban đầu thường được tổ chức theo dạng ngang, do đó, nếu muốn tổ chức theo dạng dọc phải thực hiện bước tiền xử lý dữ liệu để chuyển đổi.

**Tính chất Apriori**: Hầu hết các thuật toán trong khai thác dữ liệu đều dựa trên tính chất Apriori.

“All nonempty subsets of a frequent itemset must also be frequent”

“Tất cả các tập con khác rỗng của một itemset phổ biến đều phổ biến”

* + “If an itemset is frequent, then all of its subsets must also be frequent”: nếu một itemset phổ biến thì tất cả các tập con cũng đều phổ biến.
  + “If an item set is infrequent then all its supersets must also be infrequent”: nếu một itemset không phổ biến thì tất cả các tập cha chứa itemset cũng không phổ biến.

## Khai thác chuỗi

Kể từ khi Agrawal và Srikant đề xuất thuật toán AprioriAll [14] cho bài toán khai thác chuỗi, đã có nhiều thuật toán khai thác chuỗi khác nhau được công bố. Ban đầu là các thuật toán thực hiện khai thác tuần tự, sau đó, khi tập dữ liệu ngày càng tăng, việc khai thác tuần tự không còn hiệu quả, một số công trình nghiên cứu đề xuất các thuật toán thực hiện song song quá trình khai thác mẫu tuần tự. Các thuận toán khai thác thực hiện tuần tự, dựa vào đặc trưng của thuật toán, có thể phân loại các thuật toán như sau: kỹ thuật định dạng dữ liệu dạng ngang (Horizontal Database Format Algorithms), tiêu biểu như AprioriAll [14], GSP [15]. Một số thuật toán sử dụng kỹ thuật định dạng dữ liệu dạng dọc (Vertical Database Format Algorithms), tiêu biểu như SPAM [10], SPADE [16], LAPIN-SPAM [17]. Một số thuật toán sử dụng kỹ thuật phát triển mẫu (Pattern Growth Algorithms), tiêu biểu như FreeSpan [18], PrefixSpan [19].

Các thuật toán chiếu cơ sở dữ liệu theo chiều dọc hiệu quả hơn các thuật khác do chỉ duyệt CSDL 1 hoặc 2 lần và trong quá trình khai thác chỉ sử dụng các bảng định dạng dọc để tạo ra các mẫu ứng viên và tính độ hỗ trợ bằng các phép giao logic, phép xử lý bit AND…. Thuật toán SPADE biểu diễn dữ liệu dưới dạng cấu trúc IDList. Để giảm dung lượng bộ nhớ, thuật toán SPAM, bitSPACE [20], LAPIN-SPAM sử dụng cấu trúc vector bit để biểu diễn dữ liệu theo định dạng dọc. Theo đó, giá trị 1 tương ứng với sự xuất hiện của sự kiện trong giao dịch, ngược lại, giá trị 0 là sự kiện không xuất hiện trong giao dịch.

Điểm hạn chế của cấu trúc vector bit là sử dụng cố định kích thước bảng bit cho tất cả mẫu tuần tự. Trong quá trình mở rộng mẫu tuần tự, thông tin xuất hiện mẫu tuần tự ít dần, đồng nghĩa với việc xuất hiện nhiều giá trị 0 trong vector bit. Vo và cộng sự đề xuất cấu trúc vector bit động trong thuật toán khai thác tập sự kiện phổ biến DBV-Miner [21] nhằm khắc phục hạn chế này. Để khai thác chuỗi đóng, Thai và cộng sự đề xuất thuật toán CloFS-DBV [12], đã đề xuất một cấu trúc dữ liệu bằng cách kết hợp vector bit động và thông tin vị trí xuất hiện của các giao dịch. Ý tưởng chính của cấu trúc vector bit động là tìm cách loại bỏ các giá trị 0 dư thừa ở đầu và cuối mỗi vector bit.

Nhằm giúp loại bỏ sớm mẫu ứng viên dư thừa, Philippe Fournier-Viger và cộng sự đề xuất cấu trúc dữ liệu CMAP [11] và thuật toán CM-SPAM [11] năm 2014. Dữ liệu được lưu trữ trong CMAP là ánh xạ mỗi sự kiện k thuộc *I* vào tập các sự kiện mà các sự kiện này có thể kết hợp với sự kiện k theo 2 hình thức mở rộng chuỗi hoặc mở rộng tập sự kiện để tạo ra mẫu ứng viên. Với minSup cho trước, CMAP được xem như bảng tham chiếu các mẫu phổ biến có độ dài 2 (2-sequence) thỏa điều kiện minSup.

Với xu hướng các tập dữ liệu ngày càng tăng và phức tạp dẫn đến không gian khai thác rất lớn, vấn đề khai thác hết tất cả các mẫu tuần tự tốn nhiều tài nguyên và thời gian. Do đó, thực hiện khai thác song song là giải pháp lý tưởng cho quá trình khai thác mẫu tuần tự. Một số thuật toán thực hiện khai thác song song được đề xuất như pSPADE [22]: dựa trên hệ thống chia sẽ bộ nhớ (share-memory based), tree-projection-based [23]: dựa trên hệ thống phân tán bộ nhớ (distributed-memory based). PIB-PRISM [24], pDBV-SPM [25]: dựa trên vi xử lý đa lõi (multi-core processor). Mặc dù các phương pháp này đạt hiệu suất và khả năng mở rộng nhưng vẫn mới chỉ khai thác trên máy đơn. Do đó, những thuật toán này sẽ gặp các hạn chế như giới hạn tài nguyên (limited resource), chi phí giao tiếp (communication cost), khai thác không cân bằng tải (load unbalancing).

Mô hình lập trình phân tán MapReduce [26], được đề xuất bởi Google, cung cấp nền tảng cho phép phát triển và thực hiện công việc xử lý dữ liệu quy mô lớn. Hadoop MapReduce [9], mã nguồn mở miễn phí phát triển dựa trên nền tảng Google MapReduce, cho phép phân tán xử lý lượng dữ liệu lớn trên cụm máy tính với độ tinh cậy và tính sẵn sàng cao. Nhiều thuật toán khác nhau được đề xuất nhằm mục đích khai thác song song trên nền tảng phân tán dựa trên mô hình MapReduce như DPSP [27], PTDS [28], BIDE-MR [29], SPAMC [30]. Hang và cộng sự đề xuất thuật toán DPSP [27] khai thác mẫu tuần tự trên CSDL tăng trưởng. Trong [28], Wang và cộng sự đề xuất thuật toán PTDS khai thác mẫu tuần tự, theo đó, thuật toán chia dữ liệu ban đầu thành các tập dữ liệu nhỏ và thực hiện khai thác dựa trên ý tưởng thuật toán PrefixSpan. Điểm hạn chế của PTDS là phát sinh toàn bộ tập mẫu ứng viên có trong từng tập dữ liệu con. Do đó, sẽ không hiệu quả đối với dữ liệu có chuỗi dài. Để khai thác mẫu tuần tự đóng, lấy ý tưởng từ thuật toán BIDE, Yu và cộng sự đề xuất thuật toán BIDE-MR [29]. Thuật toán SPAMC [30] được phát triển dựa trên thuật toán SPAM, thực hiện khai thác mẫu tuần tự từ CSDL chuỗi. Thay vì thực hiện khai thác trên cây từ điển hoàn chỉnh, thuật toán phân chia thành các cây con, và sử dụng lặp đi lặp lại mô hình MapReduce trên các cây con để tìm ra các mẫu phổ biến. Tuy nhiên, thuật toán SPAMC không hiệu quả đối với tập dữ liệu có số lượng sự kiện lớn, dữ liệu chuyển từ hàm Mapper đến Reducer lớn ảnh hưởng đến thời gian thực thi.

## Thuật toán SPAM

SPAM [10] là từ viết tắt của Sequential PAttern Mining using A Bitmap Representation, đây là thuật toán khai thác mẫu tuần tự dựa trên cấu trúc dữ liệu tổ chức theo dạng dọc. Thuật toán SPAM kế thừa các ý tưởng của các thuật toán GSP, SPADE và FreeSpan. SPAM sử dụng cấu trúc dữ liệu được thiết kế vừa với bộ nhớ chính và sử dụng cấu trúc dữ liệu cây từ điển để lưu các chuỗi và duyệt cây theo chiều sâu. Thuật toán sử dụng cấu trúc dữ liệu bảng bit dạng dọc để tăng hiệu quả trong quá trình tính độ hỗ trợ của mẫu ứng viên. Trong suốt quá trình khai thác, SPAM chỉ thực hiện đọc CSDL ban đầu 1 lần duy nhất để chuyển đổi sang bảng bit dạng dọc, sau đó, tiến hành thực hiện khai thác dựa trên thông tin trong bảng bit này. Hình 2-1 mô tả minh họa tổ chức cấu trúc dữ liệu bảng bit dạng dọc của SDB trong SPAM.



Hình ‑ Cấu trúc dữ liệu bảng bit dạng dọc của SDB trong SPAM

SPAM là kỹ thuật khai thác mẫu tuần tự đầu tiên sử dụng cấu trúc dữ liệu cây từ điển để lưu các chuỗi và duyệt cây theo chiều sâu. Cụ thể, các chuỗi được sắp xếp trên cây thứ tự từ điển, ứng với mỗi nút trên cây là một mẫu ứng viên và nút gốc tại mức 0 là rỗng. Theo đó, mỗi mẫu ứng viên có thể được phát sinh thông qua hai hình thức: (1) mở rộng chuỗi: mẫu ứng viên được tạo ra bằng cách thêm tập sự kiện chỉ có một sự kiện vào cuối mẫu tuần tự; (2) mở rộng tập sự kiện: mẫu ứng viên được tạo ra bằng cách thêm sự kiện vào vị trí cuối của tập sự kiện cuối của mẫu tuần tự. Hình 2-2 minh họa cấu trúc cây thứ tự từ điển được sử dụng trong SPAM.



Hình ‑ Minh họa cây thứ tự từ điển trong SPAM

Các sự kiện thuộc *I* thỏa minSup sẽ được thêm vào cây. Bắt đầu từ nút gốc, mỗi nút trên cây được mở rộng lần lượt theo mở rộng chuỗi (bước S-Step) và mở rộng tập sự kiện (bước I-Step). Thuật toán duyệt các chuỗi trên cây theo chiều sâu và kiểm tra độ hỗ trợ của mỗi chuỗi ứng viên theo ngưỡng minSup một cách đệ quy. Thông tin của các mẫu ứng viên được biểu diễn dưới dạng bảng bit dạng dọc có kích thước cố định. Nếu sự kiện i xuất hiện trong tập sự kiện j của chuỗi trong CSDL thì bit tương ứng của tập sự kiện j trong chuỗi được gán giá trị 1, ngược lại gán giá trị 0. Độ hỗ trợ của mẫu tuần tự được tính toán hiệu quả bằng cách đếm các phần bảng bit không chứa tất cả bit 0.



Hình ‑ Minh hoạ bước S-Step (a) và bước I-Step (b) trong SPAM

Ví dụ, Hình 2-3 minh họa cho các bước S-Step và I-Step trong thuật toán. Với ngưỡng hỗ trợ minSup = 50%, ta có được A, B và C là các sự kiện phổ biến. Xét quá trình mở rộng mẫu tuần tự ({A}) bởi sự kiện B. Đối với bước S-Step, trước tiên thực hiện biến đổi bảng bit của mẫu tuần tự ({A}) sau cho tất cả các bit đứng sau vị trí bit 1 đầu tiên được gán giá trị 1 (giả sử vị trí xuất hiện bit 1 đầu tiên là x), các vị trí bit nhỏ hơn hoặc bằng vị trí bit 1 đầu tiên x sẽ được gán giá trị 0. Sau đó thực hiện phép toán AND giữa bảng bit đã chuyển đổi của mẫu tuần tự ({A}) và bảng bit của sự kiện {B}. Bảng bit thu được chính là bảng bit của mẫu ứng viên ({A},{B}). Độ hỗ trợ của ({A},{B}) là 3 nên mẫu tuần tự này được giữ lại. Đối với bước I-Step, chỉ cần thực hiện phép AND giữa hai bảng bit của mẫu tuần tự ({A}) và sự kiện {B}. Quá trình duyệt cây theo chiều sâu được thực hiện cho đến khi không phát sinh được mẫu tuần tự mới.

Ưu điểm của thuật toán là chỉ cần đọc CSDL 1 lần duy nhất để chuyển đổi sang cấu trúc dữ liệu bảng bit dạng dọc, chi phí phát sinh mẫu ứng viên thấp bằng cách thực hiện phép toán logic, kiểm tra độ hỗ trợ của mẫu ứng viên thực hiện nhanh chóng dựa trên bảng bit.

Khuyết điểm của thuật toán SPAM là sử dụng bộ nhớ không hiệu quả khi sử dụng bảng bit có kích thước cố định cho toàn bộ chuỗi. Với CSDL có kích thước chuỗi dài, tần suất xuất hiện của các chuỗi xuất hiện nhiều, thuật toán cần rất nhiều bộ nhớ để lưu trữ cấu trúc bảng bit và phải thực hiện nhiều phép toán logic khi phát sinh mẫu ứng viên, do đó không đạt hiệu quả sử dụng bộ nhớ. Hơn nữa, khi số lượng chuỗi cũng như số sự kiện phân biệt tăng, không gian cần thiết cho việc biểu diễn cấu trúc bảng bit tăng đáng kể, vấn đề khai thác mẫu tuần tự trên máy đơn của thuật toán gặp nhiều khó khăn. Ngoài ra, trong quá trình phát sinh mẫu ứng viên, thuật toán không hiệu quả khi phát sinh nhiều mẫu ứng viên không tồn tại trong CSDL.

## Cấu trúc CMAP và thuật toán CM-SPAM

Cấu trúc dữ liệu CMAP [11] (Co-occurrence MAP) là một cấu trúc ánh xạ đồng xuất hiện, do Philippe Fournier-Viger đề xuất năm 2014, dùng làm bảng tra trước khi thực hiện phép kết phát sinh mẫu ứng viên nhằm cắt giảm sớm số lượng ứng viên dư thừa phát sinh trong quá trình khai thác mẫu tuần tự.

Dữ liệu được lưu trữ trong CMAP là ánh xạ mỗi sự kiện k thuộc *I* vào tập các sự kiện mà các sự kiện này có thể kết hợp với sự kiện k để tạo ra mẫu phổ biến. Với minSup cho trước, CMAP được xem như bảng tham chiếu các mẫu phổ biến có độ dài bằng 2 (2-sequence) thỏa điều kiện minSup.

Thuật toán CM-SPAM [11] là sự cải tiến thuận toán SPAM bằng cách kết hợp cấu trúc CMAP vào quá trình phát sinh mẫu ứng viên. Việc kết hợp CMAP vào thuật toán SMAP sẽ giúp loại bỏ sớm các mẫu dư thừa, tăng tốc độ khai thác của thuật toán. Tương ứng với mỗi hình thức phát sinh mẫu ứng viên sẽ có cấu trúc CMAP tương ứng. Trong đó, CMAPS sử dụng trong S-Step và CMAPI sử dụng trong I-Step.

Xét trong quá trình phát sinh ứng viên, với mỗi mẫu tuần tự A và sự kiện k là sự kiện ứng viên cho quá trình mở rộng A. Nếu tồn tại sự kiện j thuộc A và k không thuộc tập mở rộng phổ biến của j trong CMAPS thì mẫu ứng viên có được từ bước mở rộng chuỗi S-Step của A với k sẽ là mẫu không tuần tự. Do đó, thay vì thực hiện phép kết mẫu tuần tự A và sự kiện k, thuật toán sẽ loại bỏ k khỏi tập Sn trong S-Step của A. Tương với I-Step, k được xét trong tập mở rộng phổ biến của j trong CMAPI.

Ví dụ như Bảng 2-2 mô tả cấu trúc CMAP cho CSDL tuần tự mô tả trong Bảng 2-1 với minSup = 50%. Dựa vào cấu trúc CMAP của sự kiện A, ta biết được các mẫu ({A},{A}), ({A},{B}), ({A},{C}), ({A,B}), ({A,C}) là các mẫu tuần tự. Đối với sự kiện C, sự kiện A được loại khỏi tập sự kiện mở rộng của sự kiện C trong quá trình mở rộng chuỗi S-Step.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sự kiện** | **Tập mở rộng trong S-Step** | **Tập mở rộng trong I-Step** | **Tập 2-sequence** |
| A | {A, B, C} | {B,C} | ({A},{A}),({A},{B}),({A},{C})  ({A,B}),({A,C}) |
| B | {A, B, C} | {C} | ({B},{A}),({B},{B}),({B},{C})  ({B,C}) |
| C | {B, C} |  | ({C},{B}),({C},{C}) |

Bảng ‑ Cấu trúc CMAP của SDB trong CM-SPAM với minSup = 50%

Ưu điểm của thuật toán CM-SPAM là ngoài việc kế thừa các ưu điểm của thuật toán SPAM, thuật toán loại bỏ sớm các mẫu ứng ứng viên dư thừa và tránh việc tốn chi phí cho thao tác kết mẫu dư thừa bằng các bổ sung lệnh kiểm tra thông qua cấu trúc CMAP.

## Cấu trúc vector bit động - DBV

Các thuật toán khai thác mẫu tuần tự dựa trên định dạng dữ liệu dạng dọc đã được chứng minh có hiệu quả hơn các thuật toán sử dụng định dạng dữ liệu dạng ngang. Các giá trị 1, 0 được dùng để đánh dấu sự xuất hiện của sự kiện trên mỗi giao dịch trong CSDL chuỗi. Trong đó:

* + Giá trị 1 cho biết sự kiện có xuất hiện trong giao dịch.
  + Ngược lại, giá trị 0 khi sự kiện không xuất hiện trong giao dịch.

Giả sử có 16 giao dịch trong CSDL tuần tự, sự kiện i xuất hiện trong các giao dịch 6, 7 và 11. Một cấu trúc vector bit cho sự kiện i cần 16 byte như sau:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Vector bit | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Bảng ‑ Ví dụ bảng vector bit 16 byte

Điểm hạn chế của cấu trúc vector bit là việc sử dụng vector có kích thước cố định. Việc khai thác mẫu tuần tự chủ yếu quan tâm đến các giá trị 1. Quá trình mở rộng mẫu ứng viên (sử dụng phép toán logic AND) sẽ xuất hiện nhiều giá trị 0 trong cấu trúc vector bit làm tốn nhiều bộ nhớ và thời gian xử lý.

Ý tưởng chính của cấu trúc vector bit động (Dynamic Bit Vector - DBV) [21] là tìm cách loại bỏ các giá trị 0 dư thừa ở đầu và cuối mỗi vector bit. Xét vector bit trong Bảng 2-3, giá trị 1 xuất hiện đầu tiên ở vị trí thứ 6 và vị trí cuối cùng xuất hiện giá trị 1 là 11. DBV chỉ lưu trữ thông tin vector bit từ vị trí xuất hiện lần đầu tiên và cuối cùng của giá trị 1. DBV gồm có 2 thành phần:

* + Vị trí bắt đầu (start): vị trí xuất hiện đầu tiên của giá trị 1.
  + Cấu trúc vector bit (vector bit): chuỗi bit từ vị trí xuất hiện đầu tiên đến vị trí cuối cùng xuất hiện giá trị 1. Việc chuyển vector bit sang giá trị số giúp gia tăng tốc độ xử lý khi áp dụng kỹ thuật khởi tạo trước bảng tham chiếu.

Với dữ liệu trong Bảng 2-3, thay vì sử dụng cấu trúc vector bit gồm 16 byte để lưu trữ, DBV chỉ cần 7 byte để lưu trữ thông tin xuất hiện của item tại mỗi giao dịch trong CSDL chuỗi. Theo đó, DBV có vị trí bắt đầu (start) là 6 và cấu trúc vector bit là 110001, được biểu diễn là {6, 11001}. Kết quả chuyển đổi vector bit trong Bảng 2-3 sang cấu trúc DBV được mô tả trong Hình 2-4.



Hình ‑ Chuyển đổi bảng vector bit trong Bảng 2-3 sang DBV

Ví dụ, với SDB trong Bảng 2-1, sự kiện A xuất hiện trong các giao dịch 1, 2, 3 và 4. Sự kiện B xuất hiện trong các giao dịch 1, 2, 3 và 4. Sự kiện C xuất hiện trong các giao dịch 1, 2 và 3. Sự kiện D xuất hiện trong giao dịch 4. Hình 2-5 mô tả kết quả chuyển đổi các sự kiện trong SDB sang DBV như sau:



Hình ‑ Biểu diễn dữ liệu dùng DBV

## Mô hình tính toán phân tán Hadoop MapReduce

### Mô hình MapReduce

Theo định nghĩa của Google [26], “MapReduce là mô hình lập trình và thực thi song song quá trình xử lý và phát sinh các tập dữ liệu lớn”. Đối với các tập dữ liệu lớn, vấn đề xử lý trên máy đơn gặp nhiều khó khăn, kém hiệu quả. Thay vào đó, MapReduce chia tập dữ liệu thành các thành phần nhỏ để xử lý, mỗi phần nhỏ có kích thước trong khoảng 16MB đến 64MB. Sau đó, xử lý các phần nhỏ song song và độc lập trên máy tính trong hệ thống phân tán. Các máy này bao gồm một máy master và các máy worker được điều kiển bởi máy master. Cuối cùng, tổng hợp tất cả kết quả đã xử lý được để có được kết quả cuối cùng. Hình 2-6 mô tả mô hình thực thi MapReduce của Google.



Hình ‑ Mô hình thực thi MapReduce của Google

MapReduce định nghĩa dữ liệu (cấu trúc và không cấu trúc) dưới dạng cặp khóa-giá trị <key, value>. Lấy ý tưởng từ lập trình hàm, MapReduce có hai tác vụ cơ bản là Map và Reduce. Đầu tiên, theo mô hình MapReduce, tác vụ Map nhằm ánh xạ các dữ liệu đầu vào thành một tập hợp các cặp khóa-giá trị rồi sau đó sử dụng tác vụ Reduce để tập hợp các giá trị có khóa giống nhau lại. Dữ liệu đầu vào (input) và đầu ra (output) của mô hình MapReduce có thể được mô tả như sau:

* + Mapper: <key input, value input> to list <key map, value map>
  + Reducer: <key map, list(values)> to list <key reducer, value reducer>

### Nền tảng Hadoop

Apache Hadoop (Hadoop) [9] là một mã nguồn mở miễn phí, được phát triển trên nền tảng Java, cung cấp một nền tảng phân tán trên các cụm máy tính giúp lưu trữ và quản lý các dữ liệu lớn. Điểm mạnh của nền tảng Hadoop là cung cấp độ tin cậy và tính sẵn sàng cao. Các thành phần chính của nền tảng Hadoop bao gồm:

* + Hadoop Distributed File System (HDFS) là hệ thống tập tin ảo. HDFS sẽ chia nhỏ dữ liệu ra thành nhiều phần. Dữ liệu được quản lý một cách có hệ thống. Ý tưởng được sử dụng ở đây là “di chuyển tính toán vào trong dữ liệu” thay vì “di chuyển dữ liệu để tính toán” như cách thông thường. Các đoạn nhỏ của tập tin sẽ được nhân rộng và lưu trữ trên nhiều máy chủ khác (thường là 3) để tăng sức chịu lỗi và tính sẵn sàng cao.
  + Hadoop MapReduce là mô hình xử lý tính toán phân tán có thể xử lý nhanh một lượng dữ liệu lớn. Các dữ liệu này được chia nhỏ và đặt tại các máy tính phân tán trong cụm máy tính. Các máy tính này sẽ hoạt động song song độc lập với nhau. Điều này giúp rút ngắn thời gian xử lý toàn bộ dữ liệu.



Hình ‑ Kiến trúc Hadoop

Hình 2-7 mô tả kiến trúc Hadoop. Các nhà phát triển Hadoop đã giữ nguyên cơ chế MapReduce của Google để cài đặt thành bộ máy thực thi Hadoop MapReduce. Đây là một nền tảng làm việc cho phép dễ dàng phát triển và triển khai các ứng dụng MapReduce. Hadoop MapReduce có các ưu điểm:

* + Xử lý tốt bài toán với lượng dữ liệu lớn có các tác vụ phân tích và tính toán phức tạp.
  + Xử lý song song trên các máy phân tán một cách chính xác và hiệu quả
  + Có thể cài đặt mô hình MapReduce trên nhiều ngôn ngữ khác nhau với các thư viện tương ứng.



Hình ‑ Ví dụ minh họa thực thi Hadoop MapReduce

Ví dụ, hình 2-8 minh họa bài toán đếm từ trên Hadoop MapReduce. Đầu tiên, dữ liệu ban đầu được chia thành 4 tập con. Tiếp đến, khởi tạo các hàm Mapper ứng với mỗi tập con. Hàm Mapper nhận dữ liệu đầu vào là đoạn văn bản, thực thi xử lý và trả về kết quả là các cặp khóa-giá trị với khóa là từ có trong văn bản và giá trị là số lần xuất hiện của từ. Sau khi các hàm Mapper thực thi xong, các hàm Shuffer được gọi với nhiệm vụ sắp xếp các từ và tổng hợp dữ liệu đầu vào cho hàm Reducer từ các kết quả đầu ra của hàm Mapper. Sau đó, các hàm Reducer nhận giá trị đầu vào là cặp khóa-giá trị với khóa là từ và giá trị là danh sách các số lần xuất hiện của từ trên các tập con và tổng hợp các giá trị số lần xuất hiện của từ. Kết quả cuối cùng nhận được là danh sách các từ và số lần xuất hiện của từ.

## Thuật toán SPAMC

Xuất phát từ thuật toán SPAM, C.Chen và cộng sự đề xuất thuật toán xử lý song song khai thác chuỗi trên mô hình lập trình phân tán MapReduce, đặt tên là SPAMC [30]. Theo đó, tác giả thiết kế mô hình khai thác song song có thể làm tăng khả năng mở rộng quá trình khai thác chuỗi bằng cách thay vì thực hiện khai thác trên cây thứ tự từ điển hoàn chỉnh, thuật toán phân chia thành các cây con, và gọi lặp lại mô hình MapReduce trên các cây con để tìm ra các mẫu tuần tự. Hình 2-10 mô tả quá trình khai thác của thuật toán. Thuật toán được chia thành hai giai đoạn chính: (1) Scanning Phase: chuyển đổi CSDL và (2) Mining Phase: khai thác mẫu tuần tự dựa trên dữ liệu đã chuyển đổi. Trong đó, Scanning Phase được thực hiện một lần duy nhất và Mining Phase được gọi lại nhiều lần cho đến khi không tìm được mẫu tuần tự mới.



Hình ‑ Quá trình khai thác của thuật toán SPAMC

Scanning Phase: Trong giai đoạn này, tương tự thuật toán SPAM, SPAMC chỉ thực hiện đọc CSDL ban đầu 1 lần duy nhất để chuyển đổi CSDL ban đầu sang cấu trúc bảng bit dọc. Đầu tiên, thuật toán chia nhỏ CSDL thành các tập đọc và tiến hành chuyển đổi song song trên các tập dữ liệu con. Việc chia nhỏ CSDL giúp tăng hiệu quả quá trình chuyển đổi cũng như tránh được vấn đề tràn bộ nhớ đối với CSDL có kích thước lớn, không thể xử lý được trên 1 máy đơn. Tiếp theo, thực hiện song song quá trình chuyển đổi dữ liệu bằng cách thực thi 1 tiến trình MapReduce. Với mỗi tập dữ liệu con sẽ khởi tạo 1 hàm Mapper để đọc dữ liệu và chuyển đổi dữ liệu. Hàm Reducer tính tổng độ hỗ trợ của mỗi sự kiện trên tất cả tập dữ liệu con, loại bỏ các sự kiện không phổ biến. Các sự kiện phổ biến được lưu trữ lại theo cấu trúc bảng băm phân tán (Distriducted Hash Table - DHT), được sử dụng là dữ liệu đầu vào của giai đoạn khai thác, Mining Phase. Hình 2-11 mô tả thông tin DHT của SDB với ngưỡng hỗ trợ minSup = 50%. Với minSup = 50%, sự kiện D có độ hỗ trợ 25% nên là sự kiện không phổ biến. Do đó, DHT không lưu trữ sự kiện D.



Hình ‑ Minh họa DHT của SDB với minSup = 50%

Mining Phase: ý tưởng chính của quá trình khai thác tập mẫu tuần tự của thuật toán là thay vì thực hiện phát sinh tất cả mẫu ứng viên trên cây toàn bộ cây thứ tự từ điển, thuật toán tiến hành khai thác song song trên từng cây con có cùng cấp. Chiến lược khai thác theo chiều sâu trên từng cây con với cấp độ được xác định trước được áp dụng.

Theo đó, hàm Mapper phát sinh tập mẫu ứng viên trên cây con theo hai hình thức mở rộng chuỗi và mở rộng tập sự kiện. Hàm Reducer tính tổng độ hỗ trợ mẫu ứng viên và loại bỏ các mẫu ứng viên không thỏa điều kiện minSup. Chiến lược khai thác này giúp tăng hiệu quả thực thi các hàm Mapper do khắc phục được tình trạng các hàm Mapper thực thi xong trước phải đợi 1 hàm Mapper thực thi chưa xong do các cây con của cây có cấp bậc không giống nhau. Qua thực nghiệm, tác giả nhận thấy rằng trong quá trình khai thác, thực thi tiến trình MapReduce trên cây con có cấp bằng 2 đạt hiệu quả tốt nhất và cũng giúp loại bỏ sớm mẫu ứng viên dư thừa cho các bước tiếp theo.

Ưu điểm của thuật toán là tận dụng được các ưu điểm của thuật toán SPAM, có thể khai thác hiệu quả mẫu tuần tự trên CSDL có kích thước lớn. Chiến lược khai thác song song trên các cây con cùng cấp giúp thuật toán loại bỏ sớm mẫu ứng viên dư thừa và đạt hiệu suất cân bằng tải.

Tuy nhiên, thuật toán còn hạn chế khi khai thác trên CSDL có số lượng sự kiện phân biệt lớn. Thêm vào đó, trong quá trình khai thác, thuật toán vẫn giữ nguyên chiến lược phát sinh mẫu ứng viên giống thuật toán SPAM gốc nên dẫn đến phát sính nhiều mẫu ứng viên dư thừa, không tồn tại trong CSDL. Đồng thời, tốn nhiều chi phí trong quá trình chuyển dữ liệu từ các hàm Mapper đến các hàm Reducer trong giai đoạn khai thác khi thực hiện chuyển toàn bộ thông tin bảng bit của mẫu ứng viên từ hàm Mapper đến hàm Reducer.

# KỸ THUẬT XỬ LÝ SONG SONG TRONG KHAI THÁC MẪU TUẦN TỰ

## Giới thiệu

Điểm hạn chế của các phương pháp khai thác chuỗi trên CSDL chuỗi hiện nay là chỉ mới dừng lại ở việc khai thác cục bộ, xử lý trên máy tính đơn và dữ liệu thực thi có kích thước tương đối nhỏ. Phân tán dữ liệu và tính toán song song là hướng nghiên cứu cần thiết để tăng cường tính hiệu quả cho bài toán nhằm đáp ứng yêu cầu ngày càng tăng của kích thước dữ liệu.

Trong chương này, luận văn trình bày thuật toán đề xuất DSPDBV. DSPDBV là thuật toán khai thác mẫu tuần tự từ CSDL chuỗi trên mô hình MapReduce. Việc thuật toán thực thi trên mô hình Hadoop MapReduce sẽ giúp cho quá trình khai thác mẫu tuần tự được xử lý song song, giúp tăng hiệu quả thời gian khai thác, dễ mở rộng và khai thác được các dữ liệu có kích thước lớn.

## Biểu diễn dữ liệu

Cấu trúc DBVItem được sử dụng để biểu diễn thông tin vị trí xuất hiện của mỗi sự kiện trong CSDL chuỗi. Mỗi DBVItem gồm có 2 thành phần: (1) Item: thông tin sự kiện và (2) BlockInfo: thông tin vị trí xuất hiện của sự kiện. Trong đó, cấu trúc BlockInfo gồm có (a) DBV: biểu diễn vị trí xuất hiện của sự kiện trên các giao dịch và (b) PostList: danh sách vị trí xuất hiện của mỗi giao dịch trong chuỗi. Độ hỗ trợ của một DBVItem được tính bằng cách đếm số bit khác 0 có trong thông tin DBV của BlockInfo.

Ví dụ, xét sự kiện C trong CSDL SDB. Chuỗi C xuất hiện trong các giao dịch {1, 2, 3}. Trong giao dịch thứ 1, sự kiện C xuất hiện tại vị trí 3; ta có thông tin vị trí của C trên giao dịch thứ nhất là {3}. Trong giao dịch thứ 2, sự kiện C xuất hiện tại các vị trí 2 và 3; ta có danh sách vị trí {2,3}. Trong giao dịch thứ 3, sự kiện C xuất hiện tại các vị trí 1 và 2; tương tự ta có {1,2}. Tổng cộng, sự kiện C xuất hiện trên 3 giao dịch 1, 2 và 3. Như vậy, ta có 1 DBV {1:111} và độ hỗ trợ của C là 3. Hình 3-1 mô tả chi tiết thông tin cấu trúc DBVItem của các sự kiện có trong CSDL SDB.

Hình ‑ Cấu trúc DBVItem của các sự kiện có trong SDB

Tương tự như DBVItem, cấu trúc DBVPattern được dùng để biểu diễn thông tin vị trí xuất hiện của chuỗi. Mỗi DBVPattern cũng gồm 2 thành phần: (1) Pattern: thông tin chuỗi và (2) BlockInfo: thông tin vị trí xuất hiện của sự kiện cuối cùng trong chuỗi.

Các mẫu ứng viên được phát sinh bằng cách mở rộng mẫu phổ biến DBVPattern từ các sự kiện phổ biến DBVItem thông qua hai hình thức: mở rộng chuỗi (sequence extension) và mở rộng tập sự kiện (itemset extension). Cấu trúc DBVTree, dạng cấu trúc cây tiền tố, được sử dụng để lưu trữ các mẫu ứng viên. Trên cây, mỗi nút là một DBVPattern. Các hình 3-2 và 3-3 minh họa các hình thức mở rộng mẫu phổ biến.

Hình ‑ Minh họa mở rộng chuỗi của DBVPattern

Hình ‑ Minh họa mở rộng tập sự kiện của DBVPattern

## Thuật toán DSPDBV

Cho trước cơ sở dữ liệu tuần tự SDB và chuỗi S (S ∈ D), giả sử SDB được chia thành n tập dữ liệu (partition), SDB = p1 + p2 + … + pn.

Ta có, sup(S/SDB) = sup(S/p1) + sup(S/p2) + … + sup(S/pn). Trong đó, sup(S/SDB) là độ phổ biến của chuỗi S đối với SDB; các sup(S/pi) (1 ≤ i ≤ n) lần lượt là các độ phổ biến của chuỗi S trên các tập dữ liệu pi.

Nhận xét 1: Cho ngưỡng hỗ trợ minSup. Chuỗi S được gọi là phổ biến nếu sup(S/p1) + sup(S/p2) + … + sup(S/pn) >= minSup.

Nhận xét 2: Giả sử chuỗi S là mẫu phổ biến bất kỳ của SDB, nếu S tồn tại trong pi (1 ≤ i ≤ n) thì S cũng được xem là mẫu phổ biến trên pi.

Nhận xét 3: Cho 2 chuỗi SA và SB, trong đó |SA| < |SB|. Với ngưỡng hỗ trợ đủ nhỏ, thời gian phát sinh tập mẫu ứng viên trong SA sẽ ít hơn đối với SB do SA có ít mẫu ứng viên hơn SB. Ví dụ, giả sử SA = ({A,B}) và SB = ({A,B,C}). Với minSup = 50%, tập mẫu ứng viên từ SA bao gồm 3 mẫu ({A}), ({B}) và ({A,B}). Trong khi đó, với SB, ta có 6 mẫu ứng viên bao gồm ({A}), ({B}), ({C}), ({A,B}), ({A,C}) và ({B,C})

### Phương pháp khai thác

Thay vì thực hiện khai thác mẫu tuần tự trên CSDL SDB, thuật toán phân chia SDB được chia thành các tập dữ liệu nhỏ (partition), tiến hành khai thác song song trên các tập dữ liệu nhỏ, sau đó tổng hợp các kết quả từ các tập dữ liệu để cho ra kết quả cuối cùng là tập các mẫu tuần tự của SDB. Quá trình thực hiện song song được thực hiện dựa trên mô hình phân tán MapReduce.



Hình ‑ Quá trình khai thác của thuật toán DSPDBV

Hình 3-4 mô tả quá trình khai thác song song mẫu tuần tự. Thuật toán sử dụng cây tiền tố để lưu trữ các ứng viên trong quá trình khai thác. Mỗi tập dữ liệu sẽ có kích thước cây khác nhau, kích thước cây càng lớn càng ảnh hưởng đến bộ nhớ và thời gian thực thi. Do đó, thuật toán tiến hành khai thác lần lược từng cấp. Quá trình này cũng giúp thuật toán sớm loại bỏ các mẫu ứng viên dư thừa. Thuật toán được mô tả chi tiết gồm 6 bước như sau:

(1) Cắt cơ sở dữ liệu tuần tự SDB thành n tập dữ liệu con (n partitions).

(2) Ứng với mỗi tập dữ liệu, chuyển đổi các chuỗi thành dữ liệu dạng cấu trúc vector bit động (lưu trữ dưới dạng các DBVItem), đồng thời tìm các chuỗi có độ dài 2 (2-sequence).

(3) Tính tổng độ hỗ trợ của các DBVItem và 2-sequence. Lưu trữ các DBVItem và 2-sequence thỏa minSup. Các DBVItem thỏa minSup sẽ được lưu trữ dưới dạng cấu trúc vector bit động tương ứng với tập dữ liệu ban đầu (listDBVItem) để thực hiện bước kế tiếp.

(4) Ứng với mỗi tập mẫu phổ biến, khai thác cây con tiền tố. Các mẫu ứng viên (DBVPattern) được phát sinh bằng cách mở rộng DBVPattern với DBVItem trên cùng tập phân chia. Trong quá trình phát sinh ứng viên, sử dụng cấu trúc CMAP (được xây dựng từ tập 2-sequence) để loại bỏ sớm ứng viên dư thừa.

(5) Tính tổng độ hỗ trợ của mỗi DBVPattern. Lưu trữ các mẫu tuần tự thỏa điều kiện minSup. Các DBVPattern thỏa minSup sẽ được lưu trữ dưới dạng cấu trúc vector bit động tương ứng với tập dữ liệu ban đầu (listDBVPattern) để thực hiện bước kế tiếp.

(6) Nếu còn tồn tại cấu trúc vector bit động cho bước tiếp theo, quay lại thực hiện bước (4).

Theo nhận xét 3, trong quá trình chia cắt CSDL tuần tự ban đầu, thuật toán phân chia các chuỗi dựa theo chiều dài của chuỗi. Việc phân chia này giúp quá trình thực hiện khai thác đạt hiệu quả và tải cân bằng tốt do tận dụng được thời gian trong khi 1 hàm Mapper khai thác trên tập chuỗi dài thì các hàm Mapper khác thực hiện nhanh chóng trên các tập chuỗi ngắn.

### Khai thác phân tán sử dụng MapReduce

Mô hình lập trình MapReduce được dùng để xử lý dữ liệu lớn dựa trên lý thuyết của mô hình tính toán song song và mô hình xử lý dữ liệu phân tán trên cụm máy tính. Thuật toán sử dụng mô hình MapReduce để thực hiện song song và phân toán quá trình khai thác mẫu tuần tự. Trong đó, hàm Mapper thực hiện quá trình chuyển đổi dữ liệu, phát sinh mẫu ứng viên. Hàm Reducer thực hiện tính tổng độ hỗ trợ của mẫu ứng viên. Các mẫu ứng viên thỏa điều kiện minSup sẽ được giữ lại. Quá trình thực hiện được chia thành 2 giai đoạn:

(i) Giai đoạn chuyển đổi dữ liệu (DBV Conversion Phase): được thực hiện trong 1 hàm Mapper và 1 hàm Reducer. Trong đó, hàm Mapper sẽ thực hiện bước 2 và hàm Reducer thực hiện bước 3 của thuật toán.

(ii) Giai đoạn khai thác mẫu phổ biến (Sequential Pattern Mining Phase): cũng được thực hiện trong 1 hàm Mapper và 1 hàm Reducer tương ứng với bước 4 và bước 5. Trong quá trình khai thác, giai đoạn này được gọi lại liên tục, tương ứng với việc lặp lại quá trình thực thi MapReduce.

|  |  |
| --- | --- |
| **Algorithm 1: DSPDBV** | |
| **Input:** | |
|  | a sequence database *SDB*, a support threshold *minSup*  and the maximum depth of child prefix tree *depth* |
| **Output:** | |
|  | complete set of frequent sequential patterns |
| 1. | let isExists; |
| 2. | let partitions = split SDB into n partitions; |
| 3. | **call** DistributedDBVConversion (partitions, minSup); |
| 4. | let isExists = check if exist output for listDBVItems; |
| 5. | **while** (isExists) |
| 6. | **call** DistributedSequentialPatternMining (listDBVPattens, minSup, depth); |
| 7. | let isExists = check if exist output for listDBVPatterns; |

Bảng ‑ Thuật toán DSPDBV

MapReduce làm việc trên các dữ liệu được định nghĩa theo cặp khóa-giá trị <key, value>. Do đó, trong quá trình thực thi trên MapReduce, thuận toán tiến hành chuyển đổi các cấu trúc dữ liệu thành cấu trúc <key, value> như sau: key: sự kiện hoặc mẫu tuần tự; value: BlockInfo hoặc độ hỗ trợ. Chi tiết sẽ được mô tả trong các nội dung tiếp theo.

Bảng 3-1 mô tả mã giả thuật toán DSPDBV. Đầu tiên, thuật toán chia CSDL SDB ban đầu thành các tập dữ liệu (dòng 2). Sau đó, thuật toán thực hiện đọc các tập dữ liệu một lần để chuyển đổi chuỗi sang cấu trúc DBVItem và tìm các chuỗi 2-sequence (dòng 3). Dòng 5-7 thực hiện khai thác mẫu phổ biến bằng cách khai thác lần lượt các cây tiền tố con cùng cấp, mẫu ứng viên được phát sinh cách mở rộng nút trên cây con. Quá trình này được lặp lại cho đến khi không còn mẫu ứng viên cho bước tiếp theo được tìm thấy.

### Phân tán quá trình chuyển đổi dữ liệu

CSDL sau khi được chia cắt thành các tập dữ liệu sẽ được lưu trữ và phân tán trong HDFS. Trong giai đoạn chuyển đổi dữ liệu (DBV Converting Phase), quá trình chuyển đổi dữ liệu được thực hiện song song bằng cách thực hiện một quá trình MapReduce (MapReduce job). Một hàm Mapper được gọi tương ứng với một tập dữ liệu con (partition). Bảng 3-2 mô tả thuật toán DistributedDBVConversion. Giả sử CSDL SDB được chia thành n partition, sẽ có n hàm Mapper được gọi. Mỗi hàm Mapper thực hiện đọc dữ liệu một lần để xây dựng các DBVItem của mỗi sự kiện có trong dữ liệu đầu vào. Đồng thời, hàm Mapper cũng tìm tập 2-sequence để xây dựng CMAP cho giai đoạn sau của thuật toán (dòng 1). Có 2 kiểu cặp khóa-giá trị <key, value> được kết xuất (output) đến hàm Reducer, các cặp khóa-giá trị (pairs) có cùng khóa sẽ được gửi đến cùng hàm Reducer. Nếu dữ liệu là các DBVItem thì cặp khóa-giá trị là <item, blockinfo>. Theo đó, khóa là sự kiện thuộc DBVItem và giá trị là thông tin blockinfo của sự kiện (dòng 2-3). Cặp khóa-giá trị thứ hai là <pattern, support> nếu dữ liệu là chuỗi 2-sequence, với khóa là chuỗi 2-sequence và giá trị là độ hỗ trợ (support) của chuỗi 2-sequence (dòng 4-5).

|  |  |
| --- | --- |
| **Algorithm 2: DistributedDBVConversion** | |
| **Mapper Side** | |
| **Input:** a partition of sequence database *p* | |
| 1. | scan partition data only one time to  a. contruct DBVItem for each item, put into DBVItems<item,blockInfo>  b. find 2-sequences and its support, put into sequences<pattern,support> |
| 2. | **for each** (dbv **in** DBVItems) |
| 3. | output <dbv.item, dbv.blockInfo> |
| 4. | **for each** (item **in** sequences) |
| 5. | output <item.pattern, item.support> |
| **Reducer Side** | |
| **Input:** mapper output pairs *<key, values>* and a support threshold *minSup* | |
| **Output:** complete set of frequent items, 2-sequences and set of DBVItems divide by partition (listDBVItem) | |
| 1. | let support; |
| 2. | let blockInfos<BlockInfo>; |
| 3. | **if** (key is pattern) // pattern is 2-sequence |
| 4. | **for each** (value **in** values) |
| 5. | let support = sum of sup(value); |
| 6. | **if** (support >= minSup) |
| 7. | output <key, support>; |
| 8. | **else if** (key is item) |
| 9. | **for each** (value **in** values) |
| 10. | let support = sum of sup(value); |
| 11. | add value to blockInfos; |
| 12. | **if** (support >= minSup) |
| 13. | output <key, support>; |
| 14. | **for each** (blockInfo **in** blockInfos) |
| 15. | output <key, blockInfo> divide by partition; |

Bảng ‑ Thuật toán DistributedDBVConversion

Hàm Reducer tính tổng độ hỗ trợ của các sự kiện hoặc chuỗi 2-sequence. Đối với các khóa (key) là chuỗi 2-sequence, tổng độ hỗ trợ được tính bằng cách cộng tất cả giá trị và những chuỗi 2-sequence không thỏa điều kiện minSup sẽ bị loại bỏ (dòng 8-12).

Đối với các khóa là sự kiện, tính tổng độ hỗ trợ dựa trên các DBV trong BlockInfo. Các sự kiện thỏa điều kiện minSup sẽ được giữ lại, đồng thời, thông tin DBVItem của sự kiện sẽ được lưu trữ lại và phân chia tương ứng với dữ liệu đầu vào (các tập dữ liệu listDBVItem) để phục vụ cho giai đoạn khai thác mẫu tuần tự (dòng 13-20).

Ví dụ với CSDL SDB được mô tả trong bảng 2-1, giả sử ngưỡng hỗ trợ là 50% và SDB được chia thành 2 tập dữ liệu. Tập thứ nhất gồm các chuỗi có định danh 1 và 2. Tập dữ liệu thứ hai gồm các chuỗi có định danh 3 và 4. Như vậy, tương ứng với tập dữ liệu, sẽ có 2 hàm Mapper được gọi, hàm Mapper 1 cho tập dữ liệu thứ nhất và hàm Mapper 2 cho tập dữ liệu thứ 2.

Ví dụ, xét sự kiện A. Trong hàm Mapper 1, sự kiện A xuất hiện tại các giao dịch có vị trị 1, 2, 3 trên chuỗi định danh 1 và xuất hiện trên các giao dịch có vị trí 1 và 2 trên chuỗi định danh 2. Như vậy ta có DBVItem cho sự kiện A là <A, ({1:11},({1,2,3},{1,2}))> và sự kiện A có độ hỗ trợ là 2. Bên cạnh đó, các 2-sequence bắt đầu bằng sự kiện A gồm có <({A},{A}), 2>, <({A},{B}), 2>, <({A},{C}), 2>, <({A,B}), 2>, <({A,C}), 2>. Tương tự, trong hàm Mapper 2, DBVItem cho sự kiện A là <A, ({1:11},({2,3},{2}))>, độ hỗ trợ của A là 2 và tập 2-sequence gồm có <({A},{A}), 1>, <({A},{B}), 1>, <({A,B}), 2>, <({A,C}), 1>, <({A,D}), 1>. Tại hàm Reducer, độ hỗ trợ của các chuỗi 2-sequence lần lượt là <({A},{A}), 3>, <({A},{B}), 3>, <({A},{C}), 2>, <({A,B}), 4>, <({A,C}), 3>, <({A,D}), 1>. Với minSup = 50%, mẫu tuần tự ({A,D}) bị loại bỏ do có độ hỗ trợ chỉ là 25%. Sự kiện A có 2 blockInfo là ({1::11},({1,2,3},{1,2})) và ({1:11},({2,3},{2})). Sự kiện A được giữ lại do có độ hỗ trợ là 100%, đồng thời, ({1::11},({1,2,3},{1,2})) được ghi vào tập listDBVItem thứ 1 và ({1:11},({2,3},{2})) được ghi vào tập listDBVItem thứ 2. Hình 3-5 mô tả kết quả dữ liệu của quá trình chuyển đổi dữ liệu của CSDL SDB.



Hình ‑ Kết quả chuyển đổi dữ liệu CSDL SDB

### Phân tán quá trình khai thác mẫu tuần tự phổ biến

Trong quá trình khai thác mẫu tuần tự phổ biến, việc khai thác toàn bộ mẫu ứng viên trên cây tiền tố DBVTree đối với tập dữ liệu lớn không hiệu quả do không gian tìm rộng, đòi hỏi nhiều tài nguyên. Bên cạnh đó, các nhánh trên cây độc lập với nhau nên quá trình phát trinh mẫu ứng viên cũng có thể được thực hiện một cách độc lập. Do đó, thuật toán được thiết kế để thực hiện khai thác theo từng cấp của cây giúp quá trình khai thác linh động và yêu cầu ít tài nguyên hơn. Các mẫu ứng viên trên cây con được khai thác song song một cách độc lập bằng cách thực hiện các quá trình MapReduce. Các nhánh trên cây có độ cao khác nhau, việc khai thác song song các cây con có độ cao vừa đủ giúp các tiến trình khai thác đạt cân bằng tải tốt hơn. Với độ cao cây con cho trước, thuật toán thực hiện lặp đi lặp lại tiến trình MapReduce để khai thác toàn bộ mẫu ứng viên của cây tiền tố DBVTree. Ngoài ra, thuận toán cũng được thiết kế để tối ưu dữ liệu chuyển từ hàm Mapper sang hàm Reducer.

Phát sinh mẫu ứng viên: các mẫu ứng viên được phát sinh song song bởi các hàm Mapper. Thuật toán sử dụng cấu trúc cây tiền tố, với các nút được sắp xếp theo thứ tự từ điển, giúp quá trình phát sinh mẫu ứng viên đạt hiệu quả cao. Mỗi nút trên cây là một mẫu ứng viên, được biểu diễn theo cấu trúc DBVPattern. Dữ liệu được chia thành các tập dữ con, nhờ đó, mỗi nút trên cây được khai thác song song độc lập trong các hàm Mapper khác nhau giúp cải thiện thời gian thực thi. Quá trình mở rộng cây con được thực hiện gồm 3 bước chính: (1) xây dựng tập DBVItem phổ biến, (2) xây dụng cấu trúc CMAP, (3) mở rộng cây con. Bảng 3-3 mô tả thuật toán cho quá trình phát sinh mẫu ứng viên (dòng 1-4). Theo đó, đối với mỗi cây con, thuật toán đọc một lần tập listDBVItem tương ứng với tập listDBVPattern chứa nút gốc để xây dựng tập sự kiện phổ biến phục vụ cho quá trình mở rộng cây (dòng 1).

|  |  |
| --- | --- |
| **Algorithm 3: DistributedSequentialPatternMining** | |
| **Mapper Side** | |
| **Input:** a list of root nodes of child prefix tree *roots*, a support threshold *minSup*  and the maximum depth of child prefix tree *depth* | |
| 1. | scan listDBVItem of particular partition only one time to  a. load list of DBVItems into DBV<item,blockInfo>  b. extract list item frequent items into items<item> |
| 2. | scan 2-sequence data only one time to contruct CMAP |
| 3. | **for each** (root **in** roots) |
| 4. | **call** DBVPattenExtension(root, DBV, CMAP, items, items, 1, depth); |
| **Reducer Side** | |
| **Input:** mapper output pairs *<key, values>*, a support threshold *minSup*  and the maximum depth of child prefix tree *depth* | |
| **Output:** complete set of frequent sequential patterns and set of DBVPatterns divide by partition (listDBVPattern) | |
| 5. | let support; |
| 6. | let blockInfos<BlockInfo>; |
| 7. | **for each** (value **in** values) |
| 8. | let support = sum of sup(value); |
| 9. | **if** (key is node at lowest depth in child prefix tree) |
| 10. | add value to blockInfos |
| 11. | **if** (support >= minSup) |
| 12. | output <key, support>; |
| 13. | **if** (key is node at lowest depth in child prefix tree) |
| 14. | **for each** (blockInfo **in** blockInfos) |
| 15. | output <key, blockInfo> divide by partition; |

Bảng ‑ Thuật toán DistributedSequentialPatternMining

Ví dụ, nút gốc R xuất hiện trong tập listDBVPattern thứ i, khi đó, hàm Mapper sẽ lấy tập sự kiện phổ biến từ listDBVItem thứ i. Tiếp theo, thuật toán sử dụng cấu trúc CMAP, được xây dựng từ tập 2-sequence phổ biến, để giúp loại bỏ sớm các mẫu ứng viên dư thừa (dòng 2). Dòng 3-4, thực hiện mở rộng cây con để phát sinh mẫu ứng viên.

Tìm mẫu phổ biến: Hàm Reducer tính tổng độ hỗ trợ của các mẫu ứng viên. Bên cạnh đó, nếu mẫu ứng viên là nút ở cấp cuối cùng của cây, thông tin BlockInfo của mẫu ứng viên sẽ được giữ lại (dòng 7-10). Các mẫu ứng viên thỏa điều kiện minSup sẽ được giữ lại (dòng 10-11), đồng thời, thông tin DBVPatten của mẫu ứng viên sẽ được lưu trữ lại và phân chia tương ứng với dữ liệu đầu vào (các tập dữ liệu listDBVPattern) để thực hiện tiếp giai đoạn khai thác mẫu tuần tự (dòng 13-15).

Bảng 3-4 mô tả thuật toán DBVPatternExtension được gọi trong hàm Mapper thực hiện quá trình phát sinh mẫu ứng viên. Mẫu ứng viên được phát sinh bằng cách mở rộng mẫu tuần tự của mỗi DBVPattern. Có hai hình thức mở rộng mẫu tuần tự: mở rộng chuỗi (dòng 2-12) và mở rộng tập sự kiện (dòng 15-22). Trước khi thực hiện thực hiện phép mở rộng, loại bỏ sớm mẫu dư thừa dựa trên cấu trúc CMAP (dòng 6 cho bước mở rộng chuỗi và dòng 16 cho bước mở rộng tập sự kiện). Các mẫu ứng viên có độ hỗ trợ lớn hơn 0 sẽ được giữ lại và chuyển đến hàm Reduce để kiểm tra tổng độ hỗ trợ (dòng 11 và dòng 21).

Bảng 3-5 mô tả thuật toán OutputDBVPattern được gọi trong thuật toán DBVPatternExtension để thực hiện việc chuyển kết quả từ hàm Mapper đến hàm Reducer. Thuật toán được thiết kế để tối ưu lượng dữ liệu trong quá trình chuyển dữ liệu. Theo đó, thay vì chuyển toàn bộ thông tin BlockInfo của mỗi nút đến hàm Reducer, thuật toán chỉ chuyển thông tin BlockInfo đến hàm Reducer nếu nút đang xét là nút ở cấp cuối cùng của cây (dòng 1-2). Ngược lại, chỉ thông tin độ hỗ trợ của nút được chuyển đi (dòng 3-4). Cách làm này giúp giảm số lượng dữ liệu chuyển tiếp giữa hàm Mapper và hàm Reducer, hạn chế tình trạng thắc cổ chai (bottloneck), hiệu suất xử lý của hàm Reducer cũng tăng lên do xử lý ít dữ liệu hơn.

|  |  |
| --- | --- |
| **Algorithm 4: DBVPattenExtension** | |
| **Input:** a root node *root*, a set of frequent items *DBV*, a co-occurrence map *CMAP*, list of extendable items *sItems*, *iItems* and depth information *depth*, *maxDepth* | |
| 1. | **if** (depth > maxDepth) |
| 2. | return; |
| 3. | let sTemp<item> = NULL; |
| 4. | let sNode<DBVPattern> = NULL; |
| 5. | **for each** (item **in** sItem) // extendable item for sequence extension |
| 6. | **if** (item is not pruned by CMAP-sequence-extension) |
| 7. | **if** (sup(let node = SequenceExtension(root, DBV(item))) > 0) |
| 8. | add item to sTemp; |
| 9. | add node to sNode; |
| 10. | **for each** (node **in** sNode) |
| 11. | **call** OutputDBVPatten(node, depth == maxDepth) |
| 12. | **call** DBVPattenExtension (node, DBV, CMAP, sTemp, iItem, depth + 1, maxDepth); |
| 13. | let iTemp<item> = NULL; |
| 14. | let iNode<DBVPattern> = NULL; |
| 15. | **for each** (item **in** iItem) |
| 16. | **if** (item is not pruned by CMAP-itemset-extension) |
| 17. | **if** (sup(let node = ItemsetExtension(root, DBV(item))) > 0) |
| 18. | add item to iTemp; |
| 19. | add node to iNode; |
| 20. | **for each** (node **in** iNode) |
| 21. | **call** OutputDBVPatten(node, depth == maxDepth) |
| 22. | **call** DBVPattenExtension (node, DBV, CMAP, sTemp, iTemp, depth + 1, maxDepth); |

Bảng ‑ Thuật toán DBVPatternExtension

|  |  |
| --- | --- |
| **Algorithm 5: OutputDBVPatten** | |
| **Input:** a node of prefix tree *node* and flag to check if node is at lowest depth *flag* | |
| 1. | **if** (flag) |
| 2. | output <node.pattern, node.BlockInfo>; |
| 3. | **Else** |
| 4. | output <node.pattern, sup(node)>; |

Bảng ‑ Thuật toán OutputDBVPattern

Ví dụ, giả sử cấp cây con depth là 2. Tập 2-sequence phổ biến của SDB gồm có ({A},{A}), ({A},{B}), ({A},{C}), ({A,B}), ({A,C}), ({B},{A}), ({B},{B}), ({B},{C}), ({B,C}), ({C},{B}) và ({C},{C}). Xét quá trình mở rộng nút B trên tập listDBVItem1. Tập listDBVItem1 có các sự kiện phổ biến A, B và C, do đó, B có thể được mở rộng bởi A, B và C. Đối với bước mở rộng chuỗi, ta có các mẫu ứng viên ({B},{A}), ({B},{B}) và ({B},{C}). Do các mẫu ứng viên này là các nút có cấp là 1 nên cặp khóa-giá trị gồm có <({B},{A}), 2>, <({B},{B}), 2> và <({B},{C}), 2>. Tiếp tục mở rộng mẫu tuần tự ({B},{B}). Đối với bước mở rộng chuỗi, các mẫu tuần tự ({B},{B},{A}), ({B},{B},{B}) và ({B},{B},{C}) bị loại bỏ do có độ hỗ trợ bằng 0. Đối với bước mở rộng tập sự kiện, các sự kiện A và B bị loại bỏ nhanh chóng bởi CMAP, thực hiện mở rộng với sự kiện C ta tìm được ({B},{B,C}). Do ({B},{B,C}) là nút có cấp bằng với giá trị depth nên cặp khóa-giá trị gửi đến hàm Reduce sẽ là <({B},{B,C}), BlockInfo>.

## Đánh giá

Thuật toán DSPDBV đạt hiệu quả cao khi thực hiện khai song song mẫu tuần tự trên cụm máy tính. Thuật toán áp dụng khai thác theo chiều sâu, thực hiện song song trên các tập dữ liệu và các cây con cùng mức. Việc khai thác trên cây con thay vì toàn bộ cây cùng lúc giúp tăng hiệu suất cân bằng tải giữa các tiến trình. Bên cạnh đó, việc sử dụng cấu trúc vector bit động giúp giảm chi phí bộ nhớ cũng như thời gian tính toán. Đồng thời, thuật toán phát hiện và loại bỏ sớm mẫu dư thừa bằng cách sử dụng cấu trúc ánh xạ đồng xuất hiện đối với các sự kiện phổ biến. Chi phí của thao tác kiểm tra thấp hơn so với thao tác kết phát sinh mẫu ứng viên, cho phép thuật toán chạy nhanh hơn

# KẾT QUẢ THỰC NGHIỆM

## Tập dữ liệu

Trong quá trình thực nghiệm, luận văn sử dụng 2 nhóm CSDL chuỗi. Nhóm CSDL chuỗi thứ nhất là 3 bộ dữ liệu thực tế, gồm có: BMSWebView1, BMSWebView2 và Kosarak10k. Các bộ dữ liệu này được cung cấp bởi Philippe Fournier-Viger và có thể tải về tại địa chỉ website <http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php>. Bảng 4-1 mô tả đặc điểm các bộ dữ liệu thực.

|  |  |  |  |
| --- | --- | --- | --- |
| **Bộ dữ liệu** | **Số chuỗi** | **Số sự kiện phân biệt** | **Loại dữ liệu** |
| BMSWebView1 | 59601 | 497 | web click stream |
| BMSWebView2 | 77512 | 3340 | web click stream |
| Kosarak10k | 10000 | 10094 | web click stream |

Bảng ‑ Đặc điểm các bộ dữ liệu thực

Nhóm CSDL chuỗi thứ hai là các bộ dữ liệu tổng hợp được phát sinh từ công cụ được cung cấp bởi IBM (IBM synthetic data generator). Các bộ dữ liệu tổng hợp có đặc điểm được mô tả trong bảng 4-2.

|  |  |
| --- | --- |
| **Tham số** | **Ý nghĩa** |
| D | Số lượng chuỗi |
| C | Số lượng giao dịch trung bình trong chuỗi |
| T | Số lượng sự kiện trung bình trong tập sự kiện |
| N | Số lượng sự kiện phân biệt có trong CSDL |

Bảng ‑ Mô tả tham số bộ dữ liệu tổng hợp

## Phương pháp thực nghiệm

Toàn bộ mã lệnh của các thuật toán sử dụng trong quá trình thực nghiệm đều được viết bằng ngôn ngữ JAVA. Quá trình thực nghiệm bao gồm ba giai đoạn. Giai đoạn thứ nhất thực hiện kiểm tra tính chính xác của thuật toán đề xuất. Giai đoạn thứ hai thực nghiệm so sánh thời gian thực thi của thuật toán DSPDBV với SPAMC. Giai đoạn thứ ba thực nghiệm kiểm tra tính mở rộng DSPDBV.

Trong giai đoạn thứ nhất, thực thi lần lượt các thuật toán PrefixSpan, SPAM, SPAMC và DSPDBV trên 3 bộ dữ liệu thực BMSWebView1, BMSWebView2 và Kosarak10k với các ngưỡng hỗ trợ khác nhau. Việc thực nghiệm được tiến hành trên trên máy tính Intel Core i5 cài đặt hệ điều hành Window 7 Ultimate, JDK 1.7, Hadoop 2.3. Với mỗi lần thực nghiệm, ghi nhận lại số lượng mẫu tuần tự khai thác được. Sau đó, tiến hành so sánh kết quả của các thuật toán về số lượng mẫu tuần tự khai thác được.

Trong giai đoạn thứ hai, thực hiện lần lượt các thuật toán SPAMC và DSPDBV trên các bộ dữ liệu tổng hợp với các tham số và ngưỡng hỗ trợ khác nhau. Các thuật toán thực thi trên môi trường Hadoop Cluster gồm 4 máy: 1 máy master và 3 máy slave. Các máy đều sử dụng hệ điều hành Ubuntu Server 13.04 (phiên bản 64bit) được cài đặt Hadoop 2.7.3, Open JDK 1.7. Tất cả các máy giao tiếp với nhau thông qua giao thức SSH. Thông tin chi tiết cấu hình các máy được mô tả trong bảng 4-3. Với mỗi lần thực nghiệm, ghi nhận lại thời gian thực thi thuật toán. Mỗi thực nghiệm được tiến hành 3 lần trên mỗi bộ dữ liệu có cùng bộ tham số và ngưỡng hỗ trợ. Thời gian thực thi của thuật toán được tính là thời gian trung bình của 3 lần thực thi. Sau đó, tiến hành so sánh thời gian thực thi các thuật toán.

|  |  |  |
| --- | --- | --- |
| **Máy/Thành phần** | **Master** | **Slaves** |
| CPU | Intel(R) Xeon(R)  CPU E3-1220 V2 @ 3.10GHz | Intel(R) Xeon(R)  CPU E3-1230 V2 @ 3.30GHz |
| Bộ nhớ (Memory) | 10GB | 8GB |
| Ổ cứng (Storage) | 500GB, 7200RPM | 500GB, 7200RPM |
| Mạng (Network) | 2 X Ethernet 1000MB | 2 X Ethernet 1000MB |

Bảng ‑ Thông tin cấu hình máy trong Hadoop Cluster

Giai đoạn thứ ba thực hiện ghi nhận kết quả tương tự giai đoạn thứ hai. Trong đó, thực hiện thực thi thuật toán DPSDBV với các ngưỡng hỗ trợ và số lượng máy trong Hadoop Cluster khác nhau.

## Kiểm tra tính chính xác

Bảng 4-4 thống kê số lượng mẫu tuần tự khai thác được khi thực thi các thuật toán PrefixSpan, SPAM, SPAMC và DSPDBV trên các bộ dữ liệu thực BMSWebView1, BMSWebView2 và Kosarak10k với các ngưỡng hỗ trợ lần lượt là 0.4%, 0.5%, 0.6% và 0.7%. Quan sát bảng kết quả ta thấy rằng tất cả các thuật toán đều có cùng số lượng mẫu tuần tự khai thác được. Kết quả thực nghiệm đã chứng minh thuật toán đề xuất thực thi chính xác khi so sánh với các thuật toán đã công bố.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Bộ dữ liệu** | **Ngưỡng hỗ trợ (%)** | **Số lượng mẫu tuần tự khai thác được** | | | |
| **PrefixSpan** | **SPAM** | **SPAMC** | **DSPDBV** |
| BMSWebView1 | 0.4 | 286 | 286 | 286 | 286 |
|  | 0.5 | 201 | 201 | 201 | 201 |
|  | 0.6 | 162 | 162 | 162 | 162 |
|  | 0.7 | 133 | 133 | 133 | 133 |
| BMSWebView2 | 0.4 | 676 | 676 | 676 | 676 |
|  | 0.5 | 408 | 408 | 408 | 408 |
|  | 0.6 | 257 | 257 | 257 | 257 |
|  | 0.7 | 187 | 187 | 187 | 187 |
| Kosarak10k | 0.4 | 2800 | 2800 | 2800 | 2800 |
|  | 0.5 | 1716 | 1716 | 1716 | 1716 |
|  | 0.6 | 1178 | 1178 | 1178 | 1178 |
|  | 0.7 | 825 | 825 | 825 | 825 |

Bảng ‑ Thống kê mẫu tuần tự khai thác trên bộ dữ thực

## So sánh với SPAMC

Thực hiện so sánh thời gian thực thi trên tập dữ liệu D500kC5T5 với số lượng sự kiện phân biệt N lần lượt là 500, 600, 700, 800, 900, 1000 và ngưỡng hỗ trợ minSup = 0.1% trên cụm máy tính có 4 máy. Khi N tăng, thuật toán DSPDBV có thời gian thực thi giảm đều. Trong khi đó, thuật toán SPAMC có thời gian thực thi giảm dần khi N tăng từ 500 lên 800, sau đó, có xu hướng tăng khi N tăng lên 900, 1000. Hình 4-1 mô tả so sánh thời gian thực thi giữa 2 thuật toán DSPDBV và SPAMC. Ví dụ, với tập dữ liệu có 1000 sự kiện phân biệt, DSPDBV thực thi nhanh hơn 5 lần so với SPAMC.

Hình ‑ So sánh thời gian thực thi giữa DSPDBV và SPAMC

## Tính mở rộng (Scalability and Extensibility)

Để kiểm tra khả năng mở rộng của thuật toán, thực hiện kiểm tra thuật toán lần lượt trên các tập dữ liệu có kích thước tăng dần với các ngưỡng hỗ trợ khác nhau trên cụm máy tính có số lượng máy khác nhau. Đầu tiên, thực nghiệm trên tập dữ liệu C5T5N1000 với số lượng chuỗi D tăng từ 1M đến 5M (mỗi bước tăng là 1M) Các tập dữ liệu này được sao chép lại từ tập D500kC5T5N1000 (Ví dụ: tập dữ liệu D1000kC5T5N1000 sẽ gồm 2 tập dữ liệu D500kC5T5N1000). Ngưỡng hỗ trợ minSup thay đổi tăng từ 0.1% đến 0.5% (mỗi bước tăng là 0.1%). Thuật toán thực thi trên cụm máy tính gồm có 4 máy. Hình 4-2 cho thấy, đối với tất cả tập dữ liệu, khi ngưỡng hỗ trợ tăng dần thì thuật toán đạt hiệu suất tốt hơn.

Hình ‑ Tác động của tập dữ liệu đối với hiệu quả khai thác

Tiếp theo, thực hiện thuật toán trên cụm máy tính có số lượng máy khác nhau để kiểm tra sự ảnh hưởng của số lượng máy khi thực hiện khai thác song song. Thực nghiệm được thực nghiệm với ngưỡng hỗ trợ minSup là 0.2% trên tập dữ liệu C5T5N1000 có số lượng chuỗi D tăng từ 1M lên 5M (mỗi bước tăng là 1M – 1 triệu chuỗi) với số lượng máy (node) lần lượt là 2, 3 và 4. Hình 4-3 cho thấy, khi số lượng máy tăng, hiệu quả khai thác của thuật toán cũng tăng theo. Do đó, với hệ thống có càng nhiều máy, thời gian khai thác của thuật toán càng giảm.

Hình ‑ Tác động của số lượng máy đối với hiệu quả khai thác

## Kết luận

Từ các kết quả thực nghiệm cho thấy rằng, thuật toán đề xuất cho kết quả chính xác trên các bộ dữ liệu BMSWebView1, BMSWebView2 và Kosarak10k. Thuật toán DSPDBV có thời gian thực thi tốt hơn thuật toán SPAMC trên bộ dữ liệu tổng hợp có số lượng sự kiện phân biệt lớn. Đồng thời, thuật toán đề xuất cũng đạt tính mở rộng cao trên tiêu chí ngưỡng hỗ trợ và số lượng máy trong Hadoop Cluster.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

Luận văn đã nghiên cứu tổng quan về bài toán khai thác chuỗi trên cơ sở dữ liệu chuỗi bao gồm các khái niệm cơ bản, cơ sở lý thuyết của bài toán và các công trình nghiên cứu đã có của các tác giả trong và ngoài nước. Phân tích các ưu và khuyết điểm của các kỹ thuật, từ đó, đề xuất kỹ thuật hiệu quả để khai khác chuỗi trên cơ sở dữ liệu chuỗi có kích thước lớn.

Luận văn đã trình bày ý tưởng và nội dung một phương pháp mới khai thác song song chuỗi tuần tự từ CSDL chuỗi trên môi trường lập trình phân tán Hadoop MapReduce. Thuật toán DSPDBV chỉ đọc CSDL một lần duy nhất cho toàn bộ quá trình khai thác. CSDL chuỗi ban đầu được chia thành các tập con. Sau đó, chuyển đổi dữ liệu chuỗi ban đầu sang cấu trúc vector bit động. Cuối cùng, thực hiện khai mẫu tuần tự trên dữ liệu đã chuyển đổi.

Toàn bộ quá trình chuyển đổi dữ liệu và khai thác mẫu tuần tự đều được thực hiện song song bằng cách thực thi các tiến trình MapReduce. Để tăng hiệu quà khai thác, thuật toán DSPDBV thực hiện khai thác lần lượt trên các cây con cùng mức, sử dụng cấu trúc đồng ánh xạ giúp loại bỏ sớm mẫu ứng viên dư thừa, giảm chi phí trong quá trình chuyển dữ liệu từ các hàm Mapper đến các hàm Reducer.

Các kết quả thực nghiệm trên các bộ dự liệu phát sinh tổng hợp cho thấy phương pháp đề xuất hiệu quả trên CSDL chuỗi có kích thước và số lượng sự kiện phân biệt lớn. Đồng thời, phương pháp đề xuất cũng đạt tính mở rộng cao. Tuy nhiên, luận văn cũng còn hạn chế khi chưa thực nghiệm trên các bộ dữ liệu thực có kích thước lớn. Số lượng máy thực nghiệm còn ít.

Kết quả nghiên cứu của luận văn đã được chấp nhận đăng trong kỷ yếu hội nghị quốc tế chuyên ngành của tạp chí IEEE (The 9th International Conference on Knowledge and Systems Engineering - KSE 2017 - được tổ chức tại Huế vào ngày 19-21 tháng 10 năm 2017).

## Hướng phát triển

Tiếp tục nghiên cứu cải tiến thuật toán DSPDBV. Những vấn đề cần nghiên cứu bao gồm kỹ thuật lưu trữ giúp tăng hiệu quả truy xuất dữ liệu giữa các hàm Mapper và Reducer, kỹ thuật giúp loại bỏ sớm mẫu dư thừa, thực nghiệm trên môi trường có số lượng máy nhiều hơn và trên các bộ dữ liệu thực.

Phát triển thuật toán khai thác song song chuỗi tuần tự đóng sử dụng cấu trúc dữ liệu đã được trình bày trong luận văn này trên môi trường lập trình phân tán MapReduce đối với dữ liệu lớn.

# DANH MỤC CÔNG TRÌNH CỦA TÁC GIẢ

1. *Distributed Algorithm for Sequential Pattern Mining on a Large Sequence Dataset*. **Tho Hoang, Bac Le, Minh-Thai Tran.** 2017, KSE2017. In 9th International Conference on Knowledge and Systems Engineering. IEEE (accepted).

# TÀI LIỆU THAM KHẢO

1. *A Data Mining Approach to consumer behavior.* **Watada, J. and Yamashiro, K.** 2006, ICICIC'06. First International Conference on (Vol. 2). IEEE, pp. 652-655.

2. *An early warning system for coastal earthquakes.* **Cervone, G., et al.** 2006, Advances in space research, 37(4), pp. 636-642.

3. *Mining of users access behavior for frequent sequential pattern from web logs .* **Vijaylakshmi, S., Mohan, V. and Suresh Raja, S.** 2010, International Journal of Database Management System (IJDM), 2.

4. *Applying Sequential Rules to Protein Localization Prediction.* **Baralis, E., Chiusano, S. and Dutto, R.** 2008, Computer and Mathematics with Applications 55 (5), pp. 867–878.

5. *Mining temporal API Rules from Imperfect Traces.* **Yang, J., et al.** 2006, Proceedings of International Conference on Software Engineering, pp. 282–291.

6. *Discovering Trends in Text Databases.* **Lent, B., Agrawal, R. and Srikant, R.** 1997, In KDD (Vol. 97), pp. 227-230.

7. **Dong, G., Pei, J.** *Sequence Data Mining.* New York : Springer Science+Business Media, LLC, 2007.

8. *Applications of pattern discovery using sequential data mining.* **Gupta, M., & Han, J.** s.l. : IGI Global, 2011, Pattern Discovery Using Sequence Data Mining: Applications and Studies, pp. 1-23.

9. *"Apache Hadoop".* **The Apache Software Foundation.**

10. *Sequential PAttern Mining using A Bitmap Representation.* **Ayres. J, Gehrke. J, Yiu. T, Flannick. J.** 2002. in SIGKDD ’02 Edmonton, Alberta, Canada.

11. *Fast Vertical Mining of Sequential Patterns Using Co-occurrence Infomation.* **Fournier-Viger, P., Gomariz, A., Campos, M., Thomas, R.** LNAI 8443, 2014, PAKDD 2014, pp. 40-52.

12. *Combination of dynamic bit vectors and transaction information for mining frequent closed sequences efficiently.* **Tran, M., Le, B. and Vo, B.** 2015, Engineering Applications of Artificial Intelligence, 38, pp. 183-189.

13. **Zaki, M. J. and Meira Jr, W.** *Data mining and analysis: fundamental concepts and algorithms.* s.l. : Cambridge University Press, 2014.

14. *Mining sequential patterns.* **Agrawal, R. and Srikant, R.** 1995, Proc. of IEEE International Conference on Data Engineering, pp. 3-14.

15. *Mining sequential patterns: Generalizations and performance improvements.* **Srikant, R. and Agrawal, R. &.** 1996, Springer Berlin Heidelberg, pp. 1-17.

16. *SPADE: An efficient algorithm for mining frequent sequences.* **Zaki, M.** 2001, Machine Learning 42(1-2), pp. 31–60.

17. *LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern.* **Yang, Z., Kitsuregawa, M.** 2005. The International Conference on Data Engineering Workshops. p. 1222.

18. *FreeSpan: Frequent Pattern-Projected Sequential Pattern.* **J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.-C. Hsu.** 2000. ACM SIGKDD Int’l Conf. Knowledge Discovery. pp. 355-359.

19. *Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth.* **Pei, J., et al.** 2001, In 2013 IEEE 29th International Conference on Data Engineering (ICDE). IEEE Computer Society., pp. 0215-0215.

20. *bitSPADE: A Lattice-based Sequential Pattern Mining Algorithm Using Bitmap Representation.* **S. Aseervatham, A. Osmani, E. Vienne.** 2006. The International Conference on Data Mining. pp. 792–797.

21. *DBV-Miner: a dynamic bit-vector approach for fast mining frequent closed itemsets.* **Vo, B., Hong, T.P., Le, B.** 2012, Expert Systems with Applications, 39, pp. 7196–7206.

22. *Parallel Sequence Mining on Shared-Memory Machines.* **Zaki, M. J.** March 2001, Journal of Parallel and Distributed Computing, Vol. 61, pp. 401-426.

23. *Parallel tree-projection-based sequence mining algorithms.* **Guralnik, V., Karypis, G.** 4, 2004, Parallel Computing, Vol. 30, pp. 443–472.

24. *Using multi-core processors for mining frequent sequential patterns.* **Huynh. B, Vo. B.** 2015, ICIC Express Letters 9(11), pp. 3071-3079.

25. *An efficient method for mining frequent sequential patterns using multi-Core processors.* **Huynh. B, Vo. B, Snasel. V.** 2017, Applied Intelligence, Vol. 46, pp. 703-716.

26. *MapReduce: Simplified Data Processing on Large Clusters.* **J. Dean and S. Ghemawat.** 2008, Communications of the ACM 51 (1), pp. 107-113.

27. *DPSP: Distributed Progressive Sequential Pattern Mining on the Cloud.* **J. W. Huang, S. C. Lin, and M. S. Chen.** 2010. Advances in Knowledge Discovery and Data Mining. pp. 27-34.

28. *Parallel Sequential Pattern Mining by Transaction Decomposition.* **Wang. X, Wang. J, Wang. T, Li. H, Yang. D.** 2010. Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2010). pp. 1746–1750.

29. *BIDE-Based Parallel Mining of Frequent Closed Sequences with MapReduce.* **Yu. D, Wu. W, Zheng. S, Zhu. Z.** Berlin : Springer, Berlin, Heidelberg, 2012. Algorithms and Architectures for Parallel Processing (ICA3PP 2012). Vol. 7440, pp. 177–186.

30. *Highly scalable sequential pattern mining based on MapReduce model on the cloud.* **Chen CC, Tseng CY, Chen MS.** 2013, IEEE international congress on big data (BigData Congress’13), pp. 310–317.