TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

PHAN QUỐC VINH
ĐẶNG THANH TÙNG

THIẾT KẾ VÀ CÀI ĐẶT HỆ THỐNG PHÂN TÍCH DỮ LIỆU THỜI GIAN THỰC

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

TP. HCM, 2015

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

PHAN QUỐC VINH - 1112389
ĐẶNG THANH TÙNG - 1112378

THIẾT KẾ VÀ CÀI ĐẶT HỆ THỐNG PHÂN TÍCH DỮ LIỆU THỜI GIAN THỰC

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

GIÁO VIÊN HƯỚNG DẪN
ThS. NGUYỄN HOÀNG ANH

KHÓA 2014 – 2015

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

TP. Hồ Chí Minh, ngày ….. tháng …… năm 2015

Giáo viên hướng dẫn

[Ký tên và ghi rõ họ tên]

NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

Khóa luận đáp ứng yêu cầu của Khóa luận cử nhân CNTT.

TP. Hồ Chí Minh, ngày ….. tháng …… năm 2015

Giáo viên phản biện

[Ký tên và ghi rõ họ tên]

LỜI CẢM ƠN

Chúng em xin chân thành cám ơn Khoa Công nghệ thông tin, trường Đại học Khoa học Tự nhiên TP.HCM đã tạo điều kiện cho chúng em thực hiện đề tài tốt nghiệp này. Xin chân thành cảm ơn các thầy cô trong khoa đã tận tình dạy dỗ, truyền đạt cho chúng em những kiến thức quý báu trong suốt 4 năm học vừa qua.

Xin cảm ơn Thầy Nguyễn Hoàng Anh, người đã tận tình hướng dẫn, chỉ bảo chúng em trong suốt thời gian thực hiện đề tài. Trong thời gian làm việc với Thầy, những kiến thức bổ ích cùng với tinh thần làm việc, thái độ nghiên cứu khoa học nghiêm túc của Thầy là những bài học vô cùng quý giá đối với chúng em.

Xin chân thành cảm ơn gia đình và bạn bè đã quan tâm, động viên và giúp đỡ chúng em rất nhiều trong quá trình thực hiện đề tài.

Mặc dù đã cố gắng hoàn thiện khóa luận với tất cả nỗ lực của bản thân, nhưng chắc chắn không thể tránh khỏi những thiếu sót. Kính mong nhận được sự góp ý của quý thầy cô để khóa luận được hoàn chỉnh hơn.

Xin cám ơn!

TP.HCM, ngày … tháng … năm 2015

Nhóm thực hiện đề tài

Phan Quốc Vinh – Đặng Thanh Tùng

Khoa Công Nghệ Thông Tin

Bộ môn Công Nghệ Phần Mềm

ĐỀ CƯƠNG CHI TIẾT

Tên đề tài: Thiết kế và cài đặt hệ thống phân tích dữ liệu thời gian thực

Giáo viên hướng dẫn: ThS. Nguyễn Hoàng Anh

Thời gian thực hiện: Từ ngày 04/10/2014 đến 01/07/2015

Sinh viên thực hiện:

- 1112389 – Phan Quốc Vinh
- 1112378 – Đặng Thanh Tùng

Loại đề tài:

- Nghiên cứu lý thuyết
- Xây dựng và triển khai hệ thống thực nghiệm

Nội dung đề tài:

Nội dung và yêu cầu đề tài

- Khảo sát, phân tích, và đánh giá một số frameworks trong việc khai thác dữ liệu thời gian thực. Từ đó, đề xuất hệ thống phân tích dữ liệu thời gian thực.
- Nghiên cứu và tìm hiểu một số thuật toán mới về Data Mining để cài đặt tích hợp vào bên trong hệ thống đã đề xuất
- Triển khai thử nghiệm hệ thống phân tích dữ liệu thời gian thực, và thực hiện một số thí nghiệm kiểm tra hiệu năng hoạt động của hệ thống

Phương pháp thực hiện

- Tìm hiểu một số hệ thống phân tích dữ liệu thời gian thực
- Chọn lọc những thuật toán mới và được đánh giá cao để tích hợp vào hệ thống
- Triển khai cài đặt thử nghiệm một số frameworks khác nhau
- Thực hiện báo cáo tiến độ hằng tuần với giáo viên hướng dẫn

Kết quả khóa luận:

- Xây dựng thành công hệ thống phân tích dữ liệu thời gian thực và tích hợp hai thuật toán khai thác loại dữ liệu này.
- Đưa ra một bộ thư viện giúp việc tích hợp nhiều thuật toán khác về khai thác mẫu phổ biến dựa trên kiến trúc đã thiết kế.
- Thí nghiệm đánh giá hiệu năng hoạt động của hệ thống, và viết báo cáo.

Kế hoạch thực hiện:

Giai đoạn 1: Tìm hiểu tổng quan về hệ thống khai thác dữ liệu thời gian thực, và một số khái niệm, thuật toán cơ sở trong Data Mining (04/10/2014 – 01/12/2015)

- 04/10/2014 – 01/12/2014: Liên hệ giáo viên hướng dẫn, nhận và tìm hiểu đề tài
    - Tuần 1, 2, 3 : Tìm hiểu về hệ khai thác dữ liệu thời gian thực
    - Tuần 4, 5, 6 : Tìm hiểu Hadoop, hệ thống HDFS và cơ chế MapReduce
- 01/12/2014 – 01/01/2015: Tìm hiểu một số loại dữ liệu và Data Mining
    - Tuần 1, 2 : Tìm hiểu về Data Stream, và hệ thống khai thác nó
    - Tuần 3, 4: Tìm hiểu một số thuật toán cơ bản và hướng tiếp cận bài toán khai thác mẫu phổ biến trên Data Stream

Giai đoạn 2: Khảo sát một số công nghệ cụ thể trong khai thác Data Stream, và một số thuật toán khai thác loại dữ liệu này (01/01/2015 – 01/03/2015)

- Tuần 1, 2, 3: Tìm hiểu một số frameworks hổ trợ việc phân tích trên Date Stream, như là Apache Samoa, Apache Kafka, Apache Storm
- Tuần 4 ,5, 6: Tìm hiểu hệ thống Storm cluster, và một số thuật toán nền Lossy Counting, MFI-TransSW, MFI-TimeSW, Space Saving
- Tuần 7,8: Cài đặt thử nghiệm Storm trên Single node, cài đặt một số thuật toán nền cho khai thác mẫu phổ biến

Giai đoạn 3: Phân tích đánh giá, và lựa chọn công nghệ cho hệ thống khai thác Data Stream. Cùng với đó, tìm hiểu một số thuật toán mới trong việc khai thác Data Stream (01/03/2015 – 15/04/2015)

- Tuần 1, 2: Tìm hiệu thuật toán TwMinSwap và cài đặt, tìm hiểu Twitter Streaming API
- Tuần 3, 4: Thiết kế mẫu kiến trúc hệ thống khai thác Data Stream
- Tuần 5, 6: Tìm hiểu paper thuật toán Skip LC-SS và cài đặt thành phần cải tiến, phát thảo sơ về hệ thống tổng quan

Giai đoạn 4: Triển khai xây dựng hệ thống, và tích hợp thuật toán (15/04/2015 – 15/05/2015)

- Tuần 1, 2: Xây dựng hệ thống hoàn chình
- Tuần 3, 4: Tích hợp TwMinSwap, Skip LC-SS vào hệ thống, và nâng cấp toàn bộ hệ thống để tăng hiệu năng xử lý và lưu trữ

Giai đoạn 5: Tiến hành kiểm tra thực nghiệm, và viết báo cáo khóa luận (15/05/2015 – 30/06/2015)

- Tuần 1, 3: Chạy thực nghiệm trên hệ thống hoàn chỉnh
- Tuần 3, 4, 5 ,6: Tiến hành viết báo cáo khóa luận

Xác nhận của GVHD ThS. Nguyễn Hoàng Anh	Ngày tháng năm 2015 Nhóm sinh viên thực hiện Phan Quốc Vinh Đặng Thanh Tùng

✪✪✪

MỤC LỤC

DANH MỤC THUẬT NGỮ VÀ CÁC TỪ VIẾT TẮT

STT	Thuật ngữ	Giải thích
1	Data stream	Dòng dữ liệu, liên tục, không biết độ dài của dòng.
2	Bursty data stream	Hiện tượng khác thường của dòng dữ liệu chỉ xảy ra tại một vài thời điểm cụ thể. Dữ liệu tại những thời điểm này khác thường, không kiểm soát được. Nguyên nhân của hiện tượng này đến từ yếu tố bên ngoài như động đất, tấn công DOS…
3	Item	Khái niệm trừu tượng dùng trong khai thác dữ liệu. Item ám chỉ những cá thể đơn lẻ như là đơn hàng trong tập các giao dịch, từ vựng trong đoạn văn bản…
3	Itemset	Khái niệm trừu tượng dùng trong khai thác dữ liệu. Itemset ám chỉ một tập các cá thể cùng tính chất như tập đơn hàng có các món hàng như là bàn, ghế, xe, tivi ….
4	Space Saving (SS)	Thuật toán khai thác item phổ biến trên data stream có thể chạy tốt dưới sự ràng buộc tài nguyên của hệ thống. Thuật toán này là một trong hai thuật toán nền của thuật toán
5	Lossy Counting (LC)	Thuật toán khai thác item/itemset phổ biến trên Data Stream. Thuật toán này là một trong hai thuật toán nền của thuật toán khai thác itemset phổ biến *Skip LC-SS*
6	AJAX	Kỹ thuật cho phép trang web tải dữ liệu bằng JavaScript bất đồng bộ không cần tải lại toàn bộ trang
7	Streaming System	Hệ thống xử lý dòng dữ liệu thời gian thực
8	Batch System	Hệ thống xử lý dữ liệu tĩnh theo lô
9	Queue	Danh sách hàng đợi
10	Batch Processing	Việc xử lý dữ liệu theo lô, thường được tới việc xử lý dự liệu tĩnh với dung lượng rất lớn

11	Edge Server	Những server thực hiện việc giao tiếp trực tiếp với người dùng cuối
12	Metadata	Dữ liệu định nghĩa cho một thành phần nào đó trong hệ thống. Thường dùng để chứa thông tin cấu hình, thông tin thực thi của ứng dụng
13	MapReduce	Mô hình xử lý song song dữ liệu lớn thông qua việc phân tán xử lý cho nhiều nodes trong cluster
14	Fault-tolerant	Khả năng giải quyết lỗi của hệ thống khi gặp sự cố. Thường lỗi thường rơi vào 2 loại. Lỗi đo mất kết nối giữa các server, hoặc là do mất gói tin trong qua trình truyền dữ liệu

DANH MỤC CÁC HÌNH

DANH MỤC CÁC BẢNG

TÓM TẮT KHÓA LUẬN

Lượng dữ liệu được truyền tải hằng ngày hàng giờ trên Internet đã và đang tăng nhanh cả về lưu lượng lẫn tốc độ, nhu cầu cho việc khai khoáng thông tin từ Data Stream cũng được quan tâm nhiều hơn không chỉ trong cộng đồng khoa học mà còn trong môi trường công nghiệp.

Mục tiêu chính của khóa luận này nhằm đề xuất một mẫu kiến trúc hệ thống để khai thác thông tin từ Data Stream. Mẫu kiến trúc này dựa trên nền tảng kiến thức tổng quan về Streaming System được trình bày trong tài liệu [1]*, và được lựa chọn và tích hợp thêm một số thành phần nhằm tăng hiệu năng cho toàn hệ thống.*

Thêm vào đó nhằm giải quyết 2 bài toán về Top-k mẫu phổ biến gần (recent frequent patterns) và Bursty trên Data Stream, khóa luận đã vận dụng 2 papers mới [2] [3] *để giải quyết hai bài toán trên, và tích hợp chúng vào hệ thống đã thiết kế. Bên cạnh đó, một bộ thư viện giúp việc tích hợp một số thuật toán mới về khai thác mẫu phổ biến trên Data Stream vào bên trong hệ thống, đã được chúng tôi xây dựng nhằm giải quyết một số vấn đề trong thực tế. Những đóng góp này chỉ ra một phương pháp khả quan trong việc khai thác dữ liệu lớn trực tiếp từ Data Stream, và cũng mở ra một hướng tiếp cận dễ dàng hơn cho những người mới bắt đầu.*

Nội dung của khóa luận bao gồm 8 chương:

Chương 1: Giới thiệu

Chương 1 sẽ giới thiệu khóa luận tổng quan về khóa luận, bài toán đặt ra và các giải pháp để giải quyết vấn đề. Ý nghĩa khoa học và thực tiễn của việc nghiên cứu.

Chương 2: Tổng quan về hệ thống phân tích Data Stream

Chương 2 trình bày về khái niệm Data Stream và đặc trưng dữ liệu này. Chương này còn giới thiệu mô hình kiến trúc tổng quan về một Streaming System, và những đặc trưng của một Streaming System.

Chương 3: Kiến trúc hệ thống khai thác Data Stream

Chương 3 trình bày về mô hình thiết kế Streaming System mà nhóm để xuất. Hơn thế nữa, chương này còn giới thiệu mô hình tích hợp một số thuật toán hiện đại về khai thác mẫu phổ biến trên Data Stream, và tích hợp thử nghiệm trên 2 thuật toán nhóm đã tìm hiểu.

Chương 4: Tổng quan khai thác mẫu phổ biến trên Data Stream

Chương 4 trình bày về một số khái niệm cơ bản về việc khai thác mẫu phổ biến trên loại dữ liệu Data Stream, như là hướng tiếp cận bài toàn, mô hình dữ liệu, một số thuật toán nền. Mục tiêu chương này nhằm làm tiền đề cho 2 thuật toán nhóm sẽ trình bày trong những chương sau.

Chương 5: Phương pháp khai thác Frequent Item trên Data Stream

Chương 5 trình bày thuật toán về khai thác items được tham khảo từ paper [2] *để giải quyết bài toán Top-k, và việc vận dụng vào hệ thống.*

Chương 6: Phương pháp khai thác Frequent Itemset trên Data Stream

Chương 6 trình bày một số thuật toán về khai thác itemset phổ biến trên Data Stream, và việc cải tiến chúng để giải quyết bài toán Bursty trên Data Stream

Chương 7: Thực nghiệm

Chương 7 trình bày mô hình triển khai thực tế dựa trên kiến trúc hệ thống đã thiết kế, và hai thí nghiêm để đo hiệu năng của toàn hệ thống sau khi đã tích hợp 2 thuật toán khai thác Data Stream.

Chương 8: Kết quả và hướng phát triển

CHƯƠNG 1: GIỚI THIỆU

Trong chương này, chúng tôi sẽ giới thiệu tổng quan cho toàn khóa luận, ý nghĩa thực tiễn và khoa học của việc nghiên cứu này. Một số vấn đề và thách thức đặt ra trong việc xây dựng một hệ thống khai thác Data Stream[1], và một số giải pháp được trình bày để giải quyết những vấn đề trên. Phần cuối chương sẽ giới thiệu tổng quan về bố cục của luận văn.

1.1. Động lực nghiên cứu

❖ ***Ý nghĩa thực tiễn***

Việc khai thác thông tin hữu ích trực tiếp từ các Data Stream ngày càng được sự quan tâm nhiều. Việc khai thác thông tin từ Data Stream trở nên cấp thiết, bởi vì khả năng cung cấp nhanh nhất những xu hướng, các thay đổi trong cộng đồng mạng. Nhiều frameworks đã ra đời nhằm giúp việc khai thác thông tin trên Data Stream được dễ dàng và hiệu quả hơn. Tuy nhiên, chúng ta cũng cần một mô hình kiến trúc hệ thống vững chắc và liên kết chặt chẽ các thành phần với nhau, để hình thành lên một hệ thống mạnh mẽ trong xử lý và linh động trong việc nâng cấp và triển khai.

❖ ***Ý nghĩa khoa học***

Trong thực tế, có nhiều loại dữ liệu khác nhau như dữ liệu *Text, Graph, Spatial,…*, tuy nhiên nổi bật và đang được sự quan tâm nhiều trong cộng động khoa học trong những năm gần đây là loại dữ liệu ***Data Stream*** [4]. Trong đó việc khai thác tập mẫu phổ biến (***frequent patterns mining***) trên Data Stream thì nổi bật hơn về lượng ứng dụng rộng rãi và được cộng đồng khoa học chú ý nhiều [4]. Có nhiều thuật toán và phương pháp đã được đề xuất nhằm giải quyết các vấn đề trong việc khai thác loại dữ liệu này, trong số đó việc khai thác *item phổ biến* và *itemset phổ biến* là hai vấn đề đang được quan tâm hơn cả.

Từ những ý nghĩa thực tiễn và ý nghĩa khoa học đó đã làm cơ sở và động lực chính để thúc đẩy các thành viên trong nhóm khảo sát, phân tích và thiết kế hệ thống khai thác Data Stream, cùng với việc xây dựng một bộ thư viện để tích hợp các thuật

[1] Data Stream: dòng dữ liệu hay dữ liệu thời gian thực. Chi tiết được trình bày trong mục 2.1

toán mới về khai thác mẫu phổ biến trên Data Stream vào hệ thống. Hai thuật toán [2] [3] là một minh chứng cho việc vận dụng bộ thư viện tích hợp này.

1.2. Phát biểu bài toán

Việc khai thác dữ liệu trên Data Stream đặt ra nhiều khó khăn và thách thức, những khó khăn này chủ yếu tập trung tại 2 thành phần là hệ thống xử lý Data Stream, và những thuật toán khai thác loại dữ liệu này [5].

1.2.1. Hệ thống phân tích Data Stream

❖ ***Vấn đề trong việc xử lý dữ liệu lớn thời gian thực***

Thời kỳ đầu, các nhà phát triển thông thường phải xây dựng một mạng lưới các *queues* [2] và *workers* [3] để khai thác Data Stream. Tuy nhiên, điều này gặp một số hạn chế sau [6]:

- *Tedious*: *việc tiêu tốn thời gian và chi phí đầu tư để phát triển hệ thống xử* lý dữ liệu real-time. Đòi hỏi nguồn nhân lực để quản trị và phát triển
- *Brittle*: *hạn chế tính logic, và khả năng chịu lỗi* của hệ thống
- *Painful to scale*: trong vài trường hợp lượng dữ liệu truyền cho một worker hay queue quá nhiều cần tới việc xử lý phân tán dữ liệu. *Việc cấu hình phân chia công việc cho những workers khác sẽ gặp nhiều khó khăn*.

Trong những năm sau đó, với việc phát minh ra công nghệ xử lý và lưu trữ dữ liệu lớn phân tán *Hadoop* [7], nó giúp việc xử lý dữ liệu lớn thông qua nhiều máy dễ dàng hơn. *Tuy nhiên, hệ thống batch system này xử lý dữ liệu với độ trễ cao vì chi phí thời gian khởi động và shutdown* [1].

Dựa trên những khảo sát và so sánh giữa nhiều framework mới hiện nay, nhóm đã chọn *Apache Storm* là thành phần xử lý dữ liệu thời gian thực và có khả năng giải quyết những vấn đề nếu trên, chi tiết được trình bày trong chương 2, và 3.

[2] Queue: là một danh sách dữ liệu hàng đợi, thông thường được lưu trữ trong Ram. Những hệ thống thời kỳ đầu thường tạo nhiều queues để chứa dữ liệu real-time được truyền vào

[3] Worker: là những ứng dụng xử lý dữ liệu, nó sẽ load dữ liệu từ queues để xử lý tuần tự

❖ ***Vấn đề trong việc giao cộng tác và tiếp giữa nhiều server với nhau***

Việc xử lý song song giữa nhiều server khác nhau trong thời kỳ đầu gặp rất nhiều vấn đề cần giải quyết, một trong số đó đã được đề cặp trong tài liệu [1]:

- *Unreliable network*: vấn đề về việc *mất kết nối giữa một hay nhiều server* bởi nhiều nguyên nhân khác nhau
- *Clock synchronization*: vấn đề về *sự bất đồng bộ thời gian* giữa nhiều server khác nhau

Vì vậy, ta cần có một hệ thống để giải quyết những vấn để được đề cập bên trên. *Apache Zookeeper* là một trong những framework nổi tiếng hiện nay với khả năng hổ trợ việc giao tiếp, và sử lý song song giữa nhiều server. Chi tiết về mô hình triển khai được trình bày trong chương 3.

❖ ***Vần đề trong việc quản lý dòng dữ liệu lớn real-time***

Data Stream là một loại dữ liệu đến từ nhiều nguồn khác nhau với cường độ cao. Vì thế việc thu thập loại dữ liệu này, và phân tán nó cho nhiều ứng dụng xử lý khác nhau đặt ra một số khó khăn [8]. Chính vì lẽ đó, ta cần một hệ thống có thể thực hiện được những việc sau

- *Thu thập dữ liệu từ nhiều nguồn dữ liệu khác nhau*
- *Phân tán dòng dữ liệu này với cường độ cao cho nhiều các ứng dụng* khác nhau trong nội bộ hệ thống
- Có *khả năng sao lưu phục hồi* dữ liệu bị mất trong quá trình xử lý

Để quản lý dòng hiệu quả, nhóm đã ứng dụng hệ thống *Apache Kafka*, và được thiết kế theo một cách logic, sao cho nó có thể hoạt động được với công suất tối đa. Chi tiết thiết kế được trình bày trong chương 3.

❖ ***Vấn đề về lưu trữ và đồng nhất dữ liệu phân tán***

Kết quả của quá trình xử lý dữ liệu phải được lưu trữ lại, để phân tán cho người dùng cuối, hoặc xử dụng cho những mục đích khai thác sau này. Tuy nhiên, trong hệ thống phân tán, dữ liệu được xử lý trên nhiều nodes khác nhau, và việc lưu trữ chúng thường được đổ về một nơi để lưu trữ. Điều này làm giảm hiệu năng của hệ thống, vì hải nguyên nhân chính như sau:

- Việc lưu trữ bên ngoài sẽ làm tốn chi *phí thời gian truyền tải dữ liệu*
- Ta cần một *cơ sở dữ liệu có khả năng đọc ghi nhanh*, và vẫn có thế lưu trữ dữ liệu bền vững.

Để có thể lưu trữ dữ liệu với tốc độ cao, và vẫn đảm bảo sự nhất quán dữ liệu, cũng như giảm chi phí truyền tải dữ liệu ra các databases bên ngoài. Nhóm đã cài đặt *RedisDB trên các server xử lý*, và áp dụng một công nghệ rất mới để đảm bảo tính nhất quán về dữ liệu giữa các databases này là *Redis Cluster*. Chi tiết thiết kế được trình bày trong chương 3.

1.2.2. Việc khai thác mẫu phổ biến trên Data Stream

Việc khai thác mẫu phổ biến trên Data Stream đặt ra 3 thách thức lớn [4]:

- Việc xử lý truy vấn trên Data Stream yêu cầu một lượng bộ nhớ rất lớn và không thể dự đoán được *(1)*.
- Số lượng mẫu phát sinh tăng theo cấp số nhân, đòi hỏi phải có một giải thuật mới để khai thác như là việc tính toán xấp xỉ. *(2)*
- Tốc độ dữ liệu truyền vào nhanh hơn tốc độ xử lý của hệ thống trong một vài trường hợp gây quá tải hệ thống *(3)*.

Để giải quyết những vấn đề trên, trong việc khai thác mẫu phổ biến trên Data Stream, chúng tôi đã vận dụng 2 thuật toán mới và được đánh giá cao [2] [3] để tích hợp vào hệ thống.

Để giải quyết *vấn đề (1) về lượng bộ nhớ tiêu thụ* khi khác thác tập phổ biến, chúng tôi ứng dụng *thuật toán* [2] *tính sấp xỉ*, chi tiết được trình bày trong chương 5.

Bài toán về *Bursty tập ứng viên*, trong trường hợp độ đài của một giao tác (*transaction*) vượt ngưỡng cho phép của hệ thống. Chúng tôi, đã khảo sát và cài đặt *thuật toán* [3] để giải quyết vấn đề *(2)*. Chi tiết được trình bày trong chương 6.

Trong vấn đề số *(3)* về việc *Bursty lượng dữ liệu đổ về hệ thống*, chúng tôi sử dụng *Apache Kafka* là nơi chứa dữ liệu tạm trong trường hợp bùng nổ dữ liệu. Chi tiết về việc thiết kế Kafka được đề cập trong chương 3.

1.3. Đóng góp của khóa luận

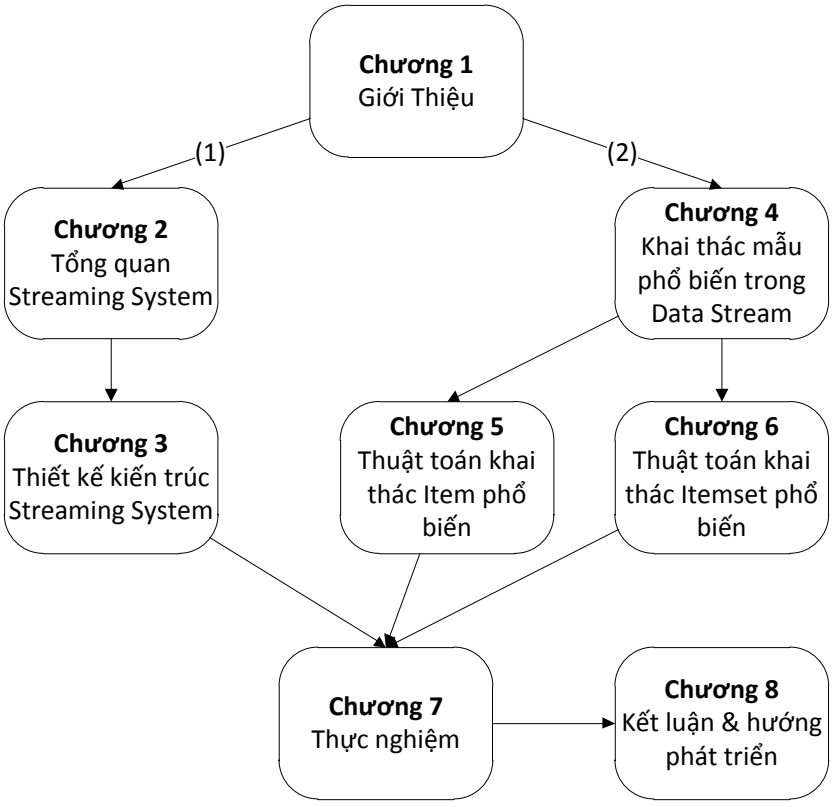
Những đóng góp của chúng tôi có thể được trình bày như sau:

- Chúng tôi đã *đề xuất một mẫu kiến trúc khai thác Data Stream*. Mẫu kiến trúc này được dựa trên bộ khung tổng quan về 5 thành phần của một Streaming System được giới thiệu trong tài liệu [1]. Tuy nhiên, tác giả không đề cập tới việc kết hợp chúng như thế nào để thành một hệ thống hoàn chỉnh và hoạt động hiệu quả. Hơn thế nữa, chúng tôi còn *tích hợp thêm một thành phần* để hệ thống hoạt động tốt nhanh hơn.
- *Nghiên cứu 2 thuật toán mới* [2] [3] về khai thác tập item phổ biến và itemset phổ biến. Những paper mới này được đánh ranking cao trên KDD[4]. Chúng tôi đã *cài đặt và tích hợp chúng vào hệ thống* đã thiết kế nhằm giải quyết hai bài toán về Top-K mẫu phổ biến và Bursty tập ứng viên.
- Chúng tôi đã *xây dựng thành công một bộ thư viện trong việc tích hợp những thuật toán mới* về khai thác mẫu phổ biến trên Data Stream vào trong hệ thống.
- Về thực nghiệm, chúng tôi đã *triển khai thành công hệ thống khai thác Data Stream dựa vào mô hình đã đề xuất* trong chương 3. Bên cạnh đó, thực nghiệm còn trình bày thử nghiệm về đo hiệu năng của hệ thống sau khi tích hợp thuật toán, thử nghiệm này nhằm *dự đoán ngưỡng chịu tải tối đa mà hệ thống vẫn hoạt động bình thường*.

[4] **KDD** (Knowledge Discovery & Data Mining): là một hội nghị danh tiếng trong lĩnh vực Data Mining, và được đánh **Rank A** trên http://www.core.edu.au/index.php/conference-rankings

1.4. Bố cục của luận văn

Khóa luận bao gồm 8 chương chính. Chương 1 là phần giới thiệu tổng quan của toàn khóa luận, về động lực nghiên cứu, vấn để gặp phải và những đóng góp thực tiễn. Chương 2 và 3 sẽ trình bày về bộ khung tổng quan và mẫu thiết kế hệ thống khai thác Data Stream. Chương 4, 5, 6 sẽ trình bày hướng tiếp cận bài toán khai thác mẫu phổ biến của khóa luận và 2 thuật toán mới trong khía cạnh này. Chương 7 sẽ trình bày về phần thực nghiệm việc tích hợp thuật toán vào trong hệ thống và đo lường hiệu năng của toàn hệ thống. Chương cuối cùng là tổng kết và hướng phát triển của khóa luận. *Hình 1-1* bên dưới mô tả sơ đồ các chương và hướng tiếp cận.



Hình 1-1: Sơ đồ kiến trúc khóa luận

Khóa luận được viết để mà ta có thể tiếp cận theo 2 hướng khác nhau, nếu mục tiêu của ta mún tìm hiểu về hệ thống khai thác Data Stream thì nên theo nhánh *(1)*. Trong đó, chúng tôi sẽ trình bày về những khái niệm cơ bản của một Streaming System trong chương 2, và chương 3 sẽ là mô hình thiết kế hệ thống mà chúng tôi đã

đề xuất. Một hướng tiếp cận khác là tìm hiểu về 2 thuật toán khai thác mẫu phổ biến hiện đại và cách tích hợp nó vào hệ thống, ta có thể theo nhánh *(2)*.

CHƯƠNG 2: TỔNG QUAN VỀ HỆ THỐNG PHÂN TÍCH DATA STREAM

Trong chương này, chúng tôi sẽ trình bày về một số khái niệm cơ bản về loại dữ liệu Data Stream như nguồn dữ liệu, những đặc trưng của dữ liệu này so với những loại dữ liệu khác, ý nghĩa và thách thức trong việc khai thác loại dữ liệu này. Chương này còn trình bày về kiến trúc tổng quan cần có của một hệ thống phân tích Data Stream, những đặc trưng của hệ thống này so với những hệ thống khác, và trình bày về 5 thành phần thiết yếu của một hệ thống phân tích Data Stream. Cuối cùng đưa ra một số lời khuyên về việc chọn frameworks thích hợp để tích hợp vào kiến trúc hệ thống phân tích Data Stream tổng quan.

2.1. Data Stream

Ngày nay, có rất nhiều loại dữ liệu khác nhau đến từ nhiều nguồn như là dữ liệu giao tác, dữ liệu network, dữ liệu web, dữ liệu siêu văn bản, …. Nhưng trong số đó phải kể tới dữ liệu Data Stream, với đặc tính nổi bật về tốc độ truyền tải và lượng thông tin hữu ích tìm ẩn bên trong dữ liệu. *Data Stream là một nguồn dữ liệu mới, là nơi mà dữ liệu được tạo ra liên tục với cường độ cao* [1].

Một số ví dụ về data stream là dòng các gói tin được gửi trên mạng, click stream, thông tin được gửi về từ các cảm biến, text stream, chuỗi các giao tác. Tiếp theo sau, chúng tôi sẽ trình bày về một số nguồn Data Stream, đặc trưng của Data Stream so với những loại dữ liệu khác, và một số vấn đề gặp phải trong việc phân tích nó.

2.1.1. Một số nguồn dữ liệu trong Data Stream

Nguồn Data Stream là nguồn sinh ra Data Stream, thường sinh ra dữ liệu liên tục và không bao giờ kết thúc. Nguồn Data Stream rất phong phú và đa dạng, nhưng ta có thể chia chúng thành 5 loại nguồn chính [1].

❖ ***Web Analytics***

Web analytics là nguồn dữ liệu real-time đến từ websites của những tổ chức hoặc công ty thông qua thương mại điện tử và quảng cáo trực tuyến. Ví dụ như việc thu

thập dữ liệu lịch sử và hiện tại từ những người dùng cuối, để gợi ý những sản phẩm tốt hơn như là Amazon. Hoặc là việc khai thác lịch sử lược view để giới thiệu phim cho người dùng như là Netflix.

❖ ***Operational Monitoring***

Operational Monitoring là hành động theo giỏi và giám xát các thiết bị từ phần cứng thông qua phần mềm. Thông thường, loại dữ liệu này được lấy từ việc theo dõi hiệu năng của những hệ thống vật lý. Ví dụ như nguồn dữ liệu đến từ việc thu thập dữ liệu nhiệt độ từ các sensors cảm biến như là tốc độ quạt, cường độ dòng điện trong hệ thống,….

❖ ***Online Advertising***

Online Advertising là những dữ liệu được thập từ những nguồn quảng cáo khác nhau. Những dữ liệu này thường được thu thập từ những giao dịch trên thị trường chứng khoán, hay số lần click vào những banner quảng cáo. Ví dụ: click stream, …

❖ ***Social Media***

Social Media là nguồn dữ liệu đền từ những mạng xã hội, và truyền thông. Nguồn dữ liệu này rất đa dạng và phong phú về loại dữ liệu như dòng dữ liệu video , dữ liệu ảnh, dữ liệu text, status trên mạng xẽ hội, các bản tin,....

❖ ***Mobile Data and Internet of Things***

Đây là một nguồn dữ liệu mới và đầy tìm năng, bởi vì lượng dữ liệu đổ về ngày càng nhiều với tốc độ tăng nhanh đáng kinh ngạc, và những dữ liệu đó có liên quan chặt chẽ tới con người hơn khi so với những dữ liệu đến từ những nguồn khác. *Những dữ liệu này đến từ những điện thoại thông minh, máy tính bảng, đồng hồ thông minh, hay từ những sensors cảm biến*. Vì thế, việc khai thác dòng dữ liệu từ nguồn này đặt ra nhiều thách thức và tìm năng trong tương lại.

2.1.2. Đặc trưng của Data Stream

Mặc dù Data Stream có rất nhiều đặc điểm nổi bật như là nguồn dữ liệu lớn, ngày càng phát triển, cấu trúc dữ liệu linh động, tốc độ truyền tải rất nhanh, đến từ rất nhiều nguồn khác nhau,.... Tuy nhiên, có 3 tính chất đặc trưng [1] nhất mà làm cho Data Stream khác biệt so với những loại dữ liệu khác.

❖ ***Always On, Always Flowing***

Data Stream thì *luôn luôn sẵn sàng* đến từ bất kỳ đâu và bất cứ lúc nào, những dữ liệu thì *luôn mới* và được tạo ra *liên tục không bao giờ kết thúc*. Trong một vài trường hợp lượng dữ liệu này đồ về rất nhiều, điều đó đặt ra nhiều thách thức cho những hệ thống quản lý và khai thác nó.

❖ ***Loosely Structured***

Data stream thường có *cấu trúc dữ liệu linh hoạt hơn so với những bộ dữ liệu khác*. Bởi vì, nó đến từ nhiều nguồn dữ liệu khác nhau, được thu thập từ nhiều nơi và được cấu thành từ nhiều thành phần dữ liệu khác nhau. Cho nên sự đa dạng về cấu trúc dữ liệu của data stream cao hơn những loại dữ liệu khác.

❖ ***High-Cardinality Storage***

Cardinality chỉ tới số lượng giá trị duy nhất mà một đơn vị dữ liệu có thể mang đi. Nói cách khác, Cardinality chỉ kích thước của một bộ dữ liệu có thể được tạo ra. *Vấn đề nhiều thành phần dữ liệu được lưu trữ trong một đối tượng dữ liệu* là một trong những đặc trưng nổi bật của data stream.

Mặc dù, đặc điểm này thì phổ biến cả trong Batch System và Streaming System, nhưng nó khó giải quyết high-cardinality hơn trong hệ thống khai thác Data Stream. Bởi vì, trong Batch System thì dữ liệu được chia thành nhiều phần nhỏ và được xử lý song song, số lần duyệt dữ liệu có thể nhiều hơn 1 lần, và thời gian xử lý thoáng hơn nhiều. Điều này đặt ra những thách thức lớn đối với những hệ thống phân tích Data Stream so với những hệ thống phân tích dữ liệu Batch.

2.1.3. Khó khăn và thách thức trong việc phân tích Data Stream

Từ những đặc điểm của Data Stream và một số nghiên cứu về hệ thống quản lý dữ liệu dòng, một số vấn đề [3] gặp phải khi triển khai xây dựng một hệ thống phân tích dữ liệu Data Stream.

❖ Việc xử lý truy vấn trên data stream *yêu cầu một lượng bộ nhớ rất lớn và không thể dự đoán được*. Vì thế cần những công nghệ xử lý truy vấn gần đúng để giải quyết việc này.

- *Tốc độ dữ liệu truyền vào nhanh hơn tốc độ xử lý của hệ thống* trong một vài trường hợp. Vì thế ta cần phải thực hiện một số phương pháp lấy mẫu dữ liệu khi cần.
- Việc phân tích trực tiếp trên stream *đòi hỏi những thuật toán nhanh và nh*ẹ để có thể đáp ứng việc hồi đáp tức thời của hệ thống
- *Đòi hỏi một hệ thống đủ mạnh và có khả năng chịu lỗi cao,* hỗ trợ những cơ chế thực hiện tính toán lại dữ liệu bị thất thoát và khắc phục sự cố hệ thống
- Hệ thống phân tích stream thường *chỉ được duyệt dữ liệu một lần duy nhất* và đưa ra một kệt quả gần đúng sau lần duyệt đó

2.2. Kiến trúc tổng quan hệ thống phân tích Data Stream

Trong mục này, chúng tôi sẽ trình bày một mô hình kiến trúc tổng quát, nó đóng vai trò như là một kiến trúc nền tảng cho việc phát triển một hệ thống khai thác Data Stream trong môi trường phân tán. Bên cạnh đó, chương này còn trình bày 3 đặc điểm quan trọng cần có của một hệ thống khai thác Data Stream. Tương ứng với từng thành phần trong mô hình kiến trúc tổng quan, chương cũng đề ra một số frameworks thích hợp cho từng thành phần. Trong mục cuối của chương, chúng tôi sẽ trình bày một số thảo luận về việc chọn những frameworks thích hợp giúp việc triển khai một hệ thống phân tích Data Stream mà chúng tôi sử dụng để triển khai hệ thống.

2.2.1. Mô hình kiến trúc tổng quan

Như đã được giới thiệu bên trên về ý nghĩa và tầm quan trọng của một mô hình kiến trúc tổng quan khai thác Data Stream. Trong mục này, chúng tôi sẽ trình bày mô hình trừu tượng của một hệ thống khai thác dòng dữ liệu [1].

Kiến trúc này được vẽ lại một cách trực quan như hình 2-1 bên dưới. *Mô hình gồm có 5 thành phần cấu thành nên, mỗi thành phần đảm nhận một nhiệm vụ khác nhau trong toàn bộ hệ thống*. Thông thường dữ liệu sẽ được truyền tuần tự qua các thành phần này với những mục đích khác nhau, nhưng trong một số trường hợp Data Stream có thể được phân nhánh và chỉ chuyền qua một vài thành phần trong mô hình, điều này phụ thuộc vào nhu cầu người dùng. Không phải tất cả hệ thống phân tích

Data Stream đều có tất cả 5 thành phần này, nhưng đây là những thành phần quan trọng và thiết yếu cần có của một Streaming System.

Hình 2-1: Kiến trúc hệ thống phân tích dòng dữ liệu

Collection	Thành phần làm nhiệm vụ thu thập dữ liệu và truyền vào Data Flow
Data Flow	Chịu trách nhiệm quản lý dòng dữ liệu truyền vào và phát ra
Processing	Làm nhiệm vụ xử lý và phân tích dữ liệu trực tuyến
Storage	Thành phần lưu trữ dữ liệu để phục vụ cho nhiều mục đích khác nhau
Delivery	Thực hiện việc phân tán và hiển thị dữ liệu sau khi phân tích

Trong giai đoạn đầu, dữ liệu được thu thập từ các thành phần khác nhau thông qua thành phần collection và được đổ trực tiếp vào data flow. Tại thành phần data flow dữ liệu được lưu trữ vào một queue trong bộ nhớ để phục vụ việc phun dữ liệu cho những thành phần khác trong hệ thống. Giai đoạn tiếp theo, dữ liệu được đổ vào thành phần processing để phục vụ cho quá trình khai thác dữ liệu trên Data Stream, sau đó kết quả của quá trình phân tích sẽ được lưu lại trong thành phần storage. Thành phần delivery sẽ thực hiện việc hiển thị và phân tán dữ liệu được lưu trữ trong storage đến những người dùng cuối. Nội dung chi tiết của từng thành phần sẽ được trình bày cụ thể trong những mục sau của chương này.

2.2.2. Đặc trưng của hệ thống phân tích Data Stream

Tất cả các thành phần của hệ thống phân tích Data stream có trong mẫu kiến trúc bên trên đều dựa trên 3 đặc điểm chính [1] là: *high availability*, *low latency*, và *horizontal scalability*. Nếu thiếu một trong những tính chất này, hệ thống phân tích Data Stream sẽ gặp những nguy cơ tìm ẩn cao và hạn chế trong khâu triển khai hệ thống.

❖ ***High Availability***

High-availability là tính sẵn sàng cao của hệ thống. Nó là một trong những đặc điểm chính của ứng dụng xử lý Data Stream khi so sánh nó với những ứng dụng batch-processing hay business-intelligence (BI*)*. Bởi vì, trong những ứng dụng Batch và BI thì việc xử lý chậm trễ vài phút hoặc vài giờ cũng không ảnh hưởng quá nhiều tới hoạt động của hệ thống, nhưng trong hệ thống xử lý Data Stream thì việc xử lý dữ liệu phải thật nhanh và kết quả phải được trả ra ngay lập tức. *Tính high-availability ở đây, không chỉ ám chỉ tới tính sẵn sàng của những dịch vụ, mà còn chỉ tới việc hệ thống luôn sẵn sàng cho việc xử lý dữ liệu.*

Tính high-availability thường không phải vấn đề lớn với thành phần Delivery trong kiến trúc bên trên, nhưng nó lại rất quan trọng với những thành phần khác như Collection, Data Flow, và Processing. Vì thế*, để đảm bảo đặc tính này thì những hệ thống phân tích Data Stream thường dựa vào 2 tính chất nổi bật của hệ thống phân tán, đó là Distribution và Replication*.

- **Distribution** được định nghĩa là việc *dùng nhiều máy server vật lý để phân tán việc xử lý cũng như thay thế cho những server bị lỗi khi cần*, nhằm đảm bảo đặc tính high-availability của hệ thống phân tích Data Stream
- **Replication** là tính *nhân rộng dữ liệu ra nhiều server vật lý khác nhau*. Ngoài việc đảm bảo hệ thống luôn sẵn sàng hoạt động, thì nguồn dữ liệu đầu vào cũng phải luôn sẵn sàng để đảm bảo hệ thống luôn có dữ liệu mới để xử lý

❖ ***Low Latency***

Trong những ứng dụng Batch-Processing, Low-latency ám chỉ tới lượng thời gian mà server hồi đáp lại cho client sau khi nhận được request từ client. Tuy nhiên trong những ứng dụng phân tích Data Stream, Low-latency còn *là lượng thời gian mà client chờ server xử lý và trả kết quả về lại*, nó bao gồm 3 khoảng thời gian:

- Thời gian các nodes của server trao đổi thông tin với nhau trong nội bộ
- Thời gian xử lý để đưa ra kết quả trả về
- Thời gian mà server hồi đáp lại người dùng cuối

Vì vậy, để giảm nhiễu, ta phải cẩn nhắc tới việc giảm thời thời gian truyền tải và xử lý tại 3 thành phần Collection, Data Flow, và Processing.

❖ ***Horizontal Scalability***

Horizontal Scalability cho biết khả năng mở rộng của hệ thống. *Đó là việc tích hợp thêm 1 số server khác vào hệ thống để giải quyết 1 số vấn đề cụ thể hoặc thêm thành phần xử lý vào hệ thống để tăng hiệu năng của toàn hệ thống*. Vấn đề đặt ra là làm sao liên kết các server này lại với nhau thành 1 khối đồng nhất và mạnh mẻ.

Quả thực vậy, khi các server trong hệ thống cộng tác làm việc với nhau, lúc này xuất hiện nhiều vấn đề phức tạp. Vì thế đã có nhiều thuật toán ra đời để cho những server này có thể hợp tác làm việc một cách dễ dàng, và quản lý các server một cách hiệu quả trong trường hợp server bị sập hoặc mất liên kết với hệ thống. Hiện tại, những những thuật toán để quản lý stream được xây dựng và đóng gói như một dịch vụ thông qua những server quản lý liên kết, gọi là Coordination Server. Một trong những hệ thống nổi tiếng làm việc này là hệ thống ZooKeeper[5].

2.2.3. Các thành phần cấu thành kiến trúc phân tích Data Stream

Như đã trình bày bên trên về kiến trúc tổng quan của một hệ thống phân tích Data Stream, trong mục này, chúng tôi sẽ mô tả chi tiết các thành phần cấu thành nên mẫu kiến trúc đó. Cùng với đó, chúng tôi sẽ giới thiệu một vài frameworks nổi tiếng và phù hợp cho việc tích hợp vào mô hình kiến trúc bên trên.

❖ ***Collection***

Thành phần Collection là *nơi thu thập dữ liệu từ nhiều nguồn khác nhau*. Trong những hệ thống lớn như Hadoop [7], thành phần này là nơi thu thập dữ liệu từ các người dùng cuối và thực hiện việc ghi Log-file vào bên trong những hệ thống lưu trữ phân tán. Tuy nhiên, *trong hệ thống phân tích Data Stream được triển khai trong khóa luận, thành phần này được tích hợp vào bên trong Data Flow*. Data Flow hiện nay làm việc như một cầu nối để truyền tãi dữ liệu giữa các ứng dụng khác nhau. Nó

[5] Apache ZooKeeper là một framework hỗ trợ việc cộng tác và giao tiếp giữa các nodes trong hệ thống phân tán. ZooKeeper sẽ được trình bày cụ thể hơn ở chapter 3.

có khả năng hút dữ liệu được lưu trữ trong hệ thống *HDFS* [6]của Hadoop và phun nó vào thành phần xử lý Processing của một ứng dụng cụ thể.

Tuy nhiên, vấn đề đặt ra là cần một định dạng dữ liệu chung để có thể truyền tải dữ liệu Client-Server và trong nội bộ hệ thống. Ngày nay, với sự xuất hiện của định dạng JavaScrip Object Notation (JSON) đã gặt hái được những thành công và giải quyết được vấn đề trước kia gặp phải. JSON hổ trợ việc xây dựng một cấu trúc dữ liệu rõ ràng và linh động, bên cạnh đó nó cũng dễ dàng cho việc mở rộng, chỉnh sữa và có nhiều thư viện hổ trợ, vì thế nó dần trở thành 1 định dạng dữ liệu Log chuẩn và được xử dụng rộng rãi.

❖ ***Data Flow***

Data Flow hay còn gọi là Data Motion System là một phần quan trọng trong hệ thống phân tích Data Stream. *Nó có 2 nhiệm vụ chính là truyền tải và thu thập dữ liệu trong môi trường phân tán*. Hệ thống này có thể được sử dụng cho nhiều dịch vụ khác nhau, mà những dịch vụ này có thể phối hợp và cấu hình lẫn nhau, nó làm việc như một hệ thống phân tán và quản lý Data Stream đến từ nhiều nguồn. Về lịch sử hình thành và phát triển, hệ thống này được chia thành 3 giai đoạn.

- Trong thời kỳ đầu, hệ thống này được xem như là một hệ thống truy vấn dữ liệu, như là ActiveMQ[7] .Tuy nhiên, nó không được thiết kế cho việc truyền tải dữ liệu lớn với tốc độ cao vì nó là một hệ thống Log-file phân tán. Ưu điểm của hệ thống này là sự tin cậy (reliability), vì nó đảm bảo giảm tới mức tối thiểu việc mất mác thông tin bằng việc ghi Log toàn bộ dữ liệu vào đĩa cứng. Mặc dù sau này khi ổ SSD ra đời, tốc độ đọc ghi có tăng đáng kể nhưng so với việc tốc độ của Ram thì vẫn còn thua xa.
- Thế hệ thứ 2 của hệ thống Data Flow nhân ra rằng việc truyền tải dữ liệu liệu cậy không phải là ưu tiên hàng đầu. Vì thế những nhà phát triển đã cài đặt thêm một số cơ chế, gọi là RPC, giúp việc truyền tải dữ liệu giữa các hệ thống được nhanh hơn. Nhưng tốc độ nhanh đòi hỏi sự mất mác dữ liệu cũng cao

[6] HDFS (Hadoop distributed file system) là nơi lưu trữ dữ liệu Log-file phân tán của Hadoop
[7] ActiveMQ: http://activemq.apache.org/

hơn, vì thế sự tin cậy cũng giảm. Một số hệ thống nổi trội tại thời kỳ này là: Scribe[8], Avro[9].

- Thế hệ thứ 3 và cũng là thế hệ mới nhất hiện giờ trong việc truyền tải và xử lý dữ liệu. Hệ thống ở giai đoạn này kết hợp sự tin cậy của thế hệ đầu tiên, và tính truyền tải dữ liệu lớn với tốc độ nhanh thông qua mô hình RPC ở thế hệ thứ 2. Nó có thể truyền tải dữ liệu real-time, và cũng hỗ trợ ghi Log-file

Apache Kafka và Apache Flume [10]là 2 frameworks nổi trội ở thế hệ thứ 3 này, vì nó đáp ứng được cả 2 tính chất truyền tải dữ liệu lớn real-time và lưu trữ bền vững dữ liệu. Tuy nhiên, về thứ tự ngữ nghĩa của dữ liệu thì bị bỏ qua ở cả 2 frameworks này. Cả 2 frameworks này đều nổi tiếng và rất mạnh mẽ trong việc quản lý và truyền tải dòng dữ liệu lớn với tốc độ cao, nhưng tùy vào mục đích sử dụng mà ta cân nhắc tới việc lựa chọn Flume hay Kafka để triển khai.

❖ ***Processing***

Trong những thập kỷ trước, với việc phát minh ra những công nghệ xử lý dữ liệu phân tán, nội bật hơn cả là cơ chế MapReduce của Hadoop. Nó cho phép lưu trữ và xử lý một lượng lớn dữ liệu cùng lúc. Tuy nhiên, những công nghệ này không hổ trợ hệ thống Data Stream, việc xử lý dữ liệu Data Stream về cơ bản khác nhiều so với việc sử lý dữ liệu theo lô như những hệ thống Batch-Processing. Bên cạnh đó, việc xử lý dữ liệu lớn Data Stream ngày càng phổ biến và nhu cầu ngày càng tăng cao. Vì thế, sự thiếu hụt khả năng xử lý dữ liệu lớn trong real-time của Hadoop bắt đầu là một vấn đề lớn trong việc xử lý dữ liệu trên hệ thống phân tán.

Thời kỳ đầu, các nhà phát triển thông thường phải xây dựng một mạng lưới các queues và workers để xử lý dữ liệu real-time. Những con workers này phải xử lý dữ liệu của một queue, cập nhật cơ sở dữ liệu, và gửi kết đã quả xử lý qua những *queues* khác để tiếp tục xử lý. Tuy nhiên, điều này gặp một số hạn chế [6] sau:

---

[8] Scribe: http://www.scribesoft.com/

[9] Avro: https://avro.apache.org/

[10] Flume : https://flume.apache.org/

- **Tedious**: việc tiêu tốn thời gian và chi phí đầu tư để phát triển hệ thống xử lý dữ liệu real-time. Đòi hỏi nguồn nhân lực để quản trị và phát triển
- **Brittle**: hạn chế tính logic, và khả năng chịu lỗi của hệ thống
- **Painful to scale**: trong vài trường hợp lượng dữ liệu truyền cho một worker hay queue quá nhiều cần tới việc xử lý phân tán dữ liệu.Việc cấu hình phân chia công việc cho những workers khác sẽ gặp nhiều khó khăn.

Hiện nay có rất nhiều frameworks hỗ trợ làm những công việc này một cách dễ dàng hơn rất nhiều, tiêu biểu trong số đó là *Apache Storm* [6], *Apache Samza*[11], và *Apache Spark*[12]. *Cả 3 frameworks này đều là những frameworks mạnh mẽ và đều hổ trợ việc xử lý dữ liệu lớn thời gian thực*. Tuy nhiên, mỗi cái có một số thế mạnh riêng như Storm và Samza thiêng về xư lý dữ liệu động (data in motion), còn Spark thiêng về việc xử lý dữ liệu tĩnh (data at rest) nhưng vẫn có cơ chế hổ trợ xử lý dữ liệu động. Vì thế, ta nên cân nhắc khi chọn và triển khai. Trong mục cuối của chương này sẽ, chúng tôi sẽ thảo luận thêm về việc chọn framework cho phù hợp với từng nhu cầu sử dụng.

❖ ***Storage***

Tầng Storage trong những hệ thống phân tích dữ liệu Data Stream thường là *tầng lưu trữ những kết quả có được sau quá trình phân tích dữ liệu nguồn, hoặc lưu lại những dữ liệu có ích để hổ trợ việc khai thác sau này*. Việc lựa chon cơ sở dữ liệu cho hệ thống phân tích Data Stream thì rất đa dạng và phong phú. Có rất nhiều cơ sở dữ liệu thỏa yêu cầu, nhưng đa số các hệ thống lưu trữ dữ liệu lớn thường dùng các loại cơ sở dữ liệu NoSQL. Trong đó nội bật hơn cả là những cơ sở dữ liệu dạng key-value trong NoSQL.

Một số cơ sỡ dữ liệu dạng NoSQL ta có thể cân nhắc tới như là *MongoDB*[13], RedisDB, *Cassandra*[14], và data warehouse. Tất cả đều phụ hợp cho hệ thống khai

[11] Apache Samze: một framework xử lý dữ liệu phân tán trên Data Stream (http://samza.apache.org/)
[12] Apache Spark: tương tự như Storm và Samza (https://spark.apache.org/)
[13] MongoDB: là cơ sở dữ liệu phân tán dạng NoSQL (https://www.mongodb.org/)
[14] Cassandra: cũng là một cơ sở dữ liệu dạng NoSQL (http://cassandra.apache.org/)

thác Data Stream, vì tính mở rộng, tốc độ lưu trữ nhanh và khả năng lưu trữ dữ liệu lớn.

❖ ***Delivery***

Tầng Delivery là *tầng trình diễn và visualizing data*[15]. Hầu hết các ứng dụng sử dụng cơ chế real-time đều thuộc loại giao diện web. Bởi vì nhiều lý do thuật lợi như dễ dàng trong việc cài đặt, gọn nhẹ trong khâu triển khai, và có rất nhiều frameworks hổ trợ cho việc này. Những ứng dụng web hiện nay hổ trợ rất mạnh việc truyển tải hoặc lấy dữ liệu real-time, như chức năng XMLHttpRequest (XHR) có trong hầu hết các trình duyệt hiện nay. Ngoài ra, kỹ thuật AJAX góp phần đơn giản hóa việc truyển tải và lấy dữ liệu giữa Client-Server, cũng như giảm kích thước các gói tin giúp việc giao tiếp được nhanh hơn. Điều đó tạo cảm giác real-time cho người dùng ứng dụng Web.

Dựa trên những phương pháp nổi bật trên, gần đây mới xuất hiện một framework mạnh mẽ trong việc trình diễn dữ liệu real-time là NodeJS [9].

2.3. Lựa chọn từng phần cho hệ thống phân tích Data Stream

Mỗi frameworks kể trên có những ưu và khuyết điểm riêng và được dùng cho những ứng dụng cụ thể, vì vậy thật khó để so sánh những frameworks này với nhau. Tuy nhiên, dựa vào nhu cầu và điều kiện, ta có thể linh động trong việc thay thế các công nghệ này lẫn nhau. Thay vì so sánh các frameworks này với nhau, khóa luận đưa ra một danh sách các frameworks tương thích tốt khi làm việc cùng nhau và dễ dàng sử dụng cho những người mới bắt đầu tìm hiểu về lĩnh vực này.

❖ ***Collection***

Trong nhiều trường hợp, thành phần này không phải là một sự lựa chọn mà ta phải cân nhắc nhiều. Trong những doanh nghiệp, thường đã tồn tại trước những hệ thống giúp làm việc này rồi, và mục tiêu của ta là việc tích hợp những server sử lý Data Stream vào trong một hệ thống lớn hơn.

[15] Virsualizing Data là công nghệ hiện thực hóa dữ liệu, giúp dữ liệu sinh động hơn và ý nghĩa hơn

Tuy nhiên, trong một vài trường hợp chúng ta có thể phải tinh chỉnh lại thành phần này để thu thập những thông tin mà chúng ta cần. Những máy chủ chạy trên Java sẽ là một sự lựa chọn tốt để làm việc này. Bởi vì, Java là một ngôn ngữ được ứng dụng rộng rãi, hiệu năng tốt, và được hổ trợ bởi nhiêu frameworks. Nó thực sự là một công cụ thích hợp để làm việc này. Trong hệ thống sẽ được trình bày ở *chương 3* thì Java là ngôn ngữ chính được dùng để triển khai cho toàn hệ thống.

❖ ***Data Flow***

Như đã để cập bên trên về 2 frameworks mạnh mẽ và nổi tiếng để thực hiện việc quản lý dòng dữ liệu, đó là Apache Flume và Apache Kafka. Cả 2 frameworks này đều tốt, và thật khó để so sánh chúng một cách cụ thể.

Tuy nhiên, Apache Flume sẽ là một sự lựa chọn tuyệt vời cho những nơi mà đã tồn tại những hệ thống chạy trước đó rồi. Bởi vì nó dễ dàng cấu hình và được hổ trợ nhiều cho việc tích hợp vào những hệ thống lớn hơn mà không gây ra xung đột. Nhưng nếu ta thêm vào một hệ thống mới hoặc xây dựng một ứng dụng mới thì Apache Kafka sẽ tốt và dễ dàng hơn. Bên cạnh đó, *Kafka còn hổ trợ rất tốt việc kết hợp với Apache Storm trong thành phần Processing*. Vì thế, Kafka sẽ là một sự lựa chọn tốt cho những người mới bắt đầu khám phá hệ thống phân tích Data Stream, và khóa luận cũng ứng dụng công nghệ này trong mô hình thiết kế hệ thống.

❖ ***Processing***

Apache Storm, Samza, và Apache Spark là 3 frameworks đại diện cho tầng xử lý của toàn hệ thống. Chúng đều là những frameworks mạnh mẽ và đầy tìm năng trong tương lại. Samza là một framework rất mới và được phát triển dựa trên *Apache YARN*[16], vì vậy nó không thực sự là một sự lựa chon tuyệt vời bởi khả năng xử lý dữ liệu động không mạnh mẽ như 2 frameworks kia. Bên cạnh đó, vì còn rất mới nên dường như không có nhiều modules hổ trợ việc tịch hợp với những frameworks khác và cộng đồng phát triển không đông, điều này gây nhiều khó khăn trong việc triển khai và phát triển.

[16] Apache YARN hay còn gọi là Hadoop 2, nó được phát triển dựa trên những thiếu sót của Hadoop trong việc quản lý nhiều nodes gây việc quá tải cho JobTracker

Apache Storm và Apache Spark là 2 frameworks trưởng thành hơn khi so với Samza. Mặc dù Spark có hổ trợ xử lý dữ liệu động nhờ vào module Spark Streaming, nhưng nó tương đối phực tạp hơn Storm trong khâu triển khai. *Apache Storm chuyên xử lý dữ liệu động trong môi trường phân tán. Bên cạnh đó việc kết hợp tuyệt vời giữa Kafka-Storm sẽ đảm bảo được việc xử lý dữ liệu một cách chính xác với hiệu năng cao*. Apache Storm là một framework gọn nhẹ, khả năng chịu lỗi cao và có một cộng đồng phát triển ngày càng đông, điều đó cho thấy được tìm năng của Storm trong tương lai không xa so với những framework khác trong việc xử lý Data Stream. Vì thế, Storm được nhóm chọn là framework chính cho toàn hệ thống để thực hiên việc xử lý dữ liệu.

❖ ***Storage***

Tùy nhu cầu người dùng mà cân nhắc tới việc sử dụng cơ sở dữ liệu nào cho phù hợp. Nếu ta cần một nơi lưu trữ dữ liệu lớn với cấu trúc dữ liệu phức tạp, có khả năng mở rộng cao mà tốc độ truy vẫn tương đối nhanh thì MongoDB sẽ là một lựa chọn tuyệt vời. Bên cạnh đó, lưu trữ dữ liệu dạng key-value như RedisDB sẽ rất hiệu quả trong trường hợp dữ liệu thường xuyên được cập nhật và cần một nơi lưu trữ và truy vấn dữ liệu với tốc độ cao. Tuy nhiên, những câu truy vấn thường đơn giản và cấu trúc dữ liệu lưu trữ đơn giản, sẽ thích hợp cho những ứng dụng ít tương tác như là dashboard.

Trong khóa luận, chúng tôi sử dụng RedisDB để làm cơ sở dữ liệu trung gian, giúp việc hiển thị dữ liệu kết quả. Bởi vì, Redis lưu trữ dữ liệu trên Ram nên tốc độ cực cao, bên cạnh đó Redis Cluster sẽ là một điểm sáng trong việc trao đổi kết quả xử lý và truy vấn dữ liệu trên các nodes trong Storm.

❖ ***Delivery***

Để làm nhiệm vụ hiển thị và visualize dữ liệu trả về trong môi trường real-time một cách mượt mà cần có một framework nhẹ và nhanh. NodeJS là một trong số đó, nó là một framework javascript nhưng vẫn có thể chạy cả ở client lẫn server, với việc xử lý và hiển thị dữ liệu cực nhanh làm cho người dùng có thể cảm nhận được sự real-time trong qua trình vận hành và sử lý dữ liệu.

Với những tính năng như lấy và hiển thị dữ liệu nhanh, dể dàng cài đặt trên bất kỳ web service nào, ngôn ngữ javascript thân thiện. NodeJS thật sử là một sự lựa chọn tuyệt vời cho những reports real-time.

2.4. Tổng kết

Trong chương này, chúng tôi đã giới thiệu về một loại dữ liệu mới là Data Stream, ý nghĩa việc khai thác nó, những thách thức và khó khăn đặt ra trong quá trình khai thác loại dữ liệu này. Mặc dù đã có rất nhiều công nghệ được đưa ra để hỗ trợ việc phân tích trên Data Stream, nhưng chúng còn rất rời rạc và chưa gắn kết chặt chẽ với nhau để trở thành một hệ thống mạnh mẽ và tìm năng trong việc khai thác dữ liệu trực tiếp từ Data Stream. Đồng tình với quan điểm này, tác giả của tài liệu [1] đã đưa ra một kiến trúc tổng quan về 5 thành phần cần có của một hệ thống khai thác Data Stream. Tuy nhiên, kiến trúc này còn rất trừu tượng và chưa chỉ ra được sự liên kết của 5 thành phần này trong việc triển khai. Dựa trên những hạn chế đó, chúng tôi đã khảo sát trên nhiều công nghệ khác nhau nhằm xây dựng lên một mô hình thiết thực cho việc triển khai Data Stream và dễ dàng trong việc tích hợp vào hệ thống lớn hơn. Chúng tôi đã nhận thấy rằng, bằng cách kết hợp 5 công nghệ mới một cách có logic, chúng ta có thể xây dựng lên một hệ thống khai thác Data Stream mạnh mẽ về hiệu năng và khả năng chịu lỗi cao khi triển khai thực tế. Hơn thế nữa, bằng việc tích hợp thêm vào hệ thống 2 thuật toán khai thác tập phổ biến được trình bày trong chương 6 và 7, chúng góp phần làm hệ thống mạnh mẽ hơn rất nhiều trong việc khai thác những thông tin tìm ẩn trong Data Stream.

Chương kế tiếp chung tôi sẽ trình bày mẫu thiết kế chi tiết cho hệ thống phân tích Data Stream. Bằng việc kết hợp các thành phần này một cách logic, hệ thống sẽ đạt được hiệu năng tối ưu hơn và dễ dàng hơn trong việc phân tích và truy vấn dữ liệu thời gian thực.

CHƯƠNG 3: KIẾN TRÚC HỆ THỐNG KHAI THÁC DATA STREAM

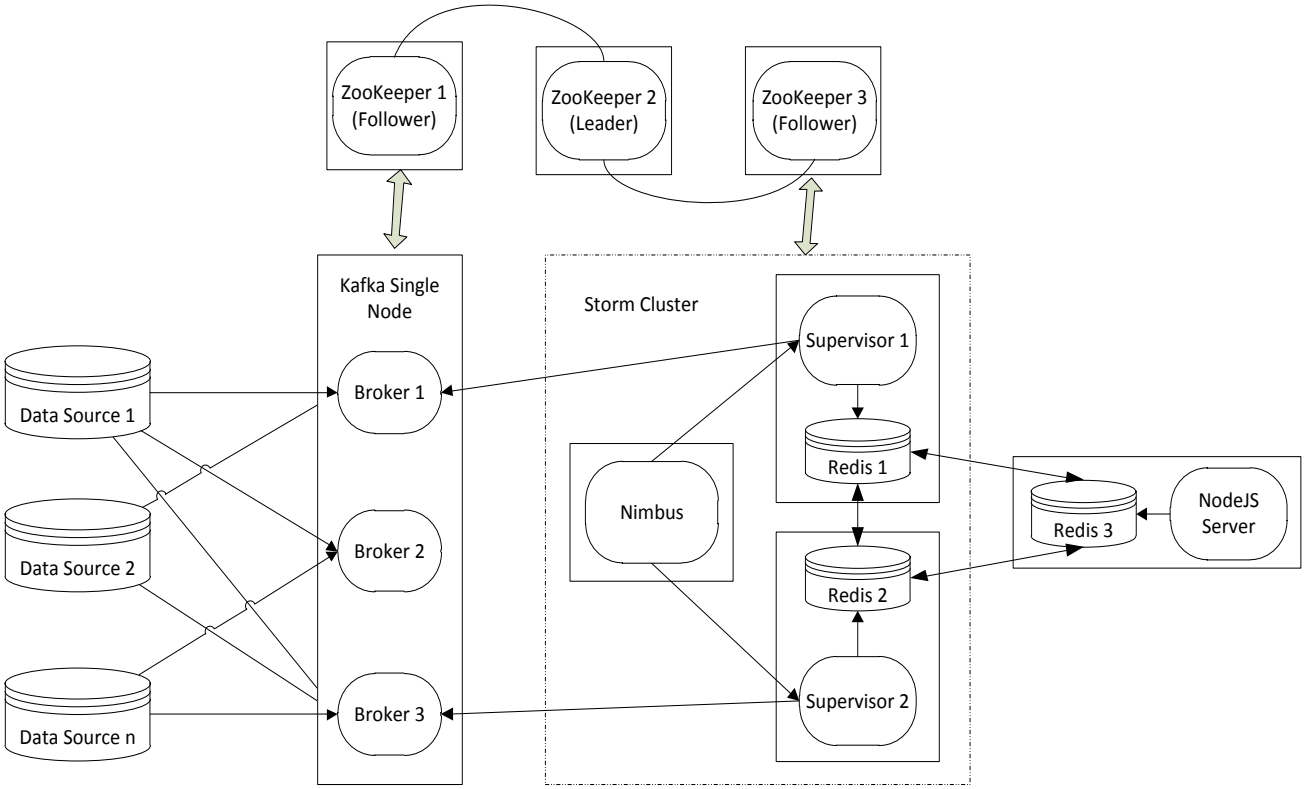
Trong chương này, chúng tôi sẽ trình bày về mô hình thiết kế hệ thống phân tích Data Stream, dựa trên nền tảng lý thuyết về một Streaming System đã được đề cập trong chương 2. Cùng với đó, chi tiết thiết kế của từng thành phần trong hệ thống cũng được chỉ ra. Mục cuối chương trình bày về bộ thư viện giúp cho việc tích hợp những thuật toán mới về khai thác mẫu phổ biến vào bên trong hệ thống.

3.1. Mô hình thiết kế hệ thống tổng quan

3.1.1. Giới thiệu mô hình

Trong mục này, chúng tôi sẽ trình bày mô hình thiết kế của toàn bộ hệ thống phân tích Data Stream. Mô hình thiết kế này sử dụng những frameworks mạnh mẽ đã được chúng tôi đề cập trong mục 2.3. *Mô hình này bao gồm 6 thành phần chính, đó là Data source, Kafka, Storm cluster, ZooKeeper, RedisDB, và NodeJS, mỗi* thành phần đảm nhận những chức năng riêng. Về môi trường, mô hình này được triển khai trên 6 server vật lý. Trong đó, ZooKeeper server đóng vai trò quan trọng nhất, nó giúp đảm bảo tính high-availability cho toàn bộ các nodes trong hệ thống.

Hình 3-1 thể hiện mô hình thiết kế của hệ thống phân tích Data Stream. Đầu tiên, dữ liệu sẽ được phát ra từ các Data sources và truyền vào Kafka server để phân luồng, lưu trữ và phun dữ liệu cho những ứng dụng nào cần, tiếp theo supervisor sẽ lấy dữ liệu từ Kafka. Tại đây, Storm thực hiện quá trình xử lý dữ liệu, những kết quả khai thác được sẽ được lưu trữ vào Redis cluster. Tiếp đó, Redis sẽ cập nhật dữ liệu vào database, và phát thông báo cho NodeJS biết dữ liệu đã được cập nhật. Cuối cùng NodeJS server sẽ thực hiện việc phân tán dữ liệu cho người dùng cuối. Các thành phần cấu thành lên mô hình trong *hình 3-1* sẽ được trình bày chi tiết trong những mục bên dưới.



Hình 3-1: Mô hình phân tích Data Stream tổng quan

3.1.2. Tính chất nổi bật của hệ thống

Hệ thống được thiết kế dựa trên những frameworks mạnh mẽ với nhiều ưu điểm nổi trội, vì vậy mô hình thiết kế này không những *thừa hưởng những ưu điểm của những frameworks cấu thành lên nó, mà còn giúp tăng hiệu năng trong quá trình phân tích Data Stream* và hiển thị kết quả cho người dùng cuối. Một số tính chất nổi bật của hệ thống được trình bày như sau:

- ❖ Khả năng xử lý dữ liệu lớn thời gian thực
- ❖ Độ trễ thấp (*low latency*)
- ❖ Khả năng mở rộng, và sử lý dữ liệu lớn (*horizontally scalable*)
- ❖ Có khả năng chịu lỗi cao khi gặp sự cố (*fault tolerant*)
- ❖ Dễ dàng trong quản lý, và triển khai
- ❖ Đảm bảo việc sử lý tất cả dữ liệu ít nhất một lần (*guaranteed data processing*)
- ❖ Truy vấn dữ liệu một các dễ dàng và nhanh chóng

3.2. Mô hình thiết kế chi tiết các thành phần trong hệ thống

3.2.1. Hệ thống quản lý việc phối hợp xử lý và cấu hình dịch vụ

Trong hầu hết các hệ thống phân tán cần phải *chia sẻ Metadata giữa các nodes với nhau*. Metadata thông thường là những loại thông tin cấu hình dịch vụ, như là: vị trí của các cơ sở liệu, hay các node master-slave thường cần lưu vết lại các hoạt động của các nodes con của nó. Vì thế, vấn đề đặt ra là làm sao để quản lý những nhu cầu này trong một hệ thống phân tán.

❖ ***Một số vấn đề phổ biến trong việc duy trì hoạt động đồng thời trong hệ thống phân tán***

Có nhiều nguyên nhân làm mất tính đồng thời trong một hệ thống phân tán. Tuy nhiên, ta có thể chia ra thành 2 nhóm vấn đề chính [1] là Unreliable Network Connection và Clock Synchronization.

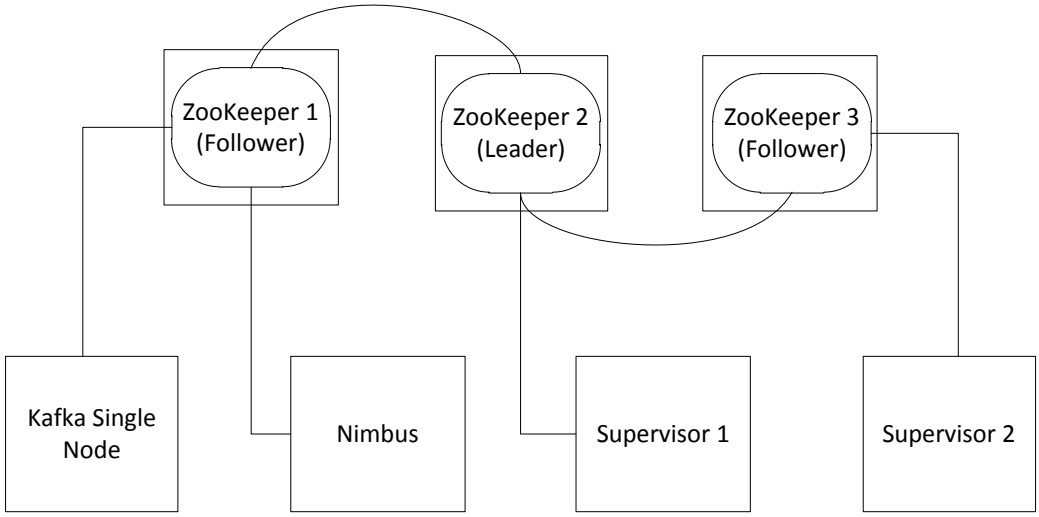
- *Unreliable Network Connection* là việc *các nodes không thể giao tiếp được với nhau do mất liên kết, hoặc giao tiếp không liên tục do bị nhiễu*. Có 4 nguyên nhân chính gây ra mất kết nối giữa các nodes là Latency, Split brain, Deadlock, Race condition
- *Clock Synchronization* là việc *bất đồng bộ hóa thời gian giữa các nodes*. Nó thì rất đơn giản trên một node, nhưng trên hệ thống phân tán nhiều nodes thì nó một vấn đề rất phổ biến. Trong những Streaming System, việc chênh lệch thời gian giữa các sự kiện truyền vào sẽ gây ra mất ngữ nghĩa dữ liệu và cho ra kết quả sai.

❖ ***Apache ZooKeeper trong hệ thống khai thác Data Stream***

Dự án ZooKeeper (ZK) được phát triển bởi Yahoo, mục tiêu để cung cấp một dịch vụ nhằm giải quyết các vấn đề đã được nêu bên trên trong hệ thống phân tán.

Hình 3-2 thể hiện sự liên kết giữa ZK và các thành phần khác trong hệ thống khai thác Data Stream. Hệ thống ZK này bao gồm 3 server riêng biệt, trong đó 2 nodes là follower và 1 node leader. Những nodes còn lại là những client của ZK, những nodes đó sẽ giao tiếp với ZK server để thực hiện việc liên kết và trao đổi thông tin cấu hình

cho nhau. Node leader có nhiệm vụ ghi lại tất cả Metadata mà các client gửi về, follower sẽ là cầu nối cho các client đọc các dữ liệu được lưu trữ bên trong ZK. Ngoài khả năng duy trì sự thống nhất cho toàn hệ thống, ZK còn là nơi lắng nghe và gửi thông báo. Các client sẽ liên tục gửi các heartbeat [17]về cho ZK để nó nhận biết được việc 1 node nào đó bị mất kết nối với hệ thống. Từ đó, nó sẽ thực thi việc tạo lại kết nối cho node bị mất và liên kết nó vào lại hệ thống.



Hình 3-2: Mô hình thiết kế ZooKeeper trong hệ thống

Trong mô hình bên trên, chúng tôi dùng 3 server riêng biệt để làm ZooKeeper cluster để nhằm đảm bảo hệ thống ZK luôn hoạt động và có hiệu năng tối ưu hơn trong trường hợp ta tích hợp thêm nhiều nodes. ZK là kênh giao tiếp chính giữa các nodes với nhau, vì thế nếu ZK bị sập toàn hệ thống sẽ bị ảnh hưởng nghiêm trọng. Trong trường hợp 1 node trong ZK cluster bị mất kết nối, ZK sẽ thực hiện việc tái cân bằng lại ZK cluster, nhằm đảm bảo hệ thống luôn ổn định.

3.2.2. Thành phần quản lý và phân phối dữ liệu

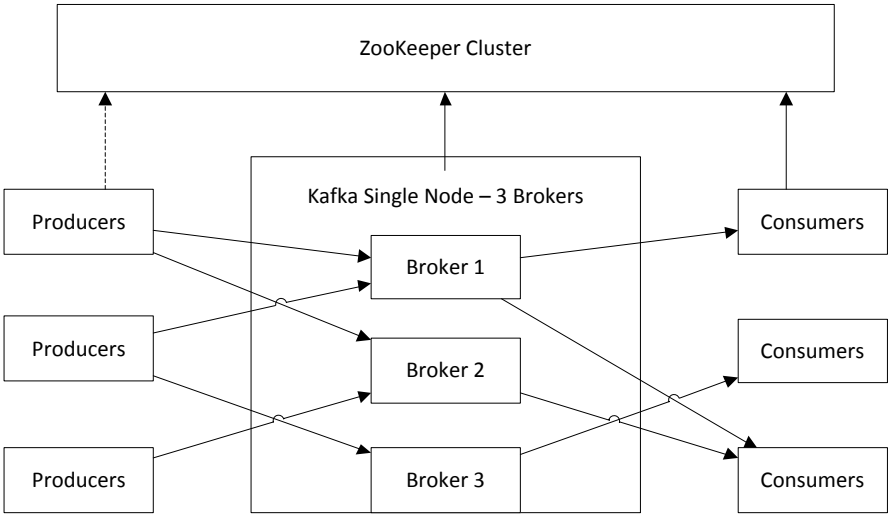
Việc thu thập và xử lý dữ liệu lớn được phát ra từ nhiều nguồn với cường độ cao đặt ra không ích khó khăn và thách thức trong việc phát triển một hệ thống khai thác Data Stream. Thông thường thời gian xử lý một đơn vị dữ liệu thấp hơn thời gian dữ liệu được truyền vào, trong một số trường hợp lượng dữ liệu đổ về quá nhiều sẽ gây

[17] Heartbeat là những request được các client của ZK gửi lên ZK cluster trong một khoảng thời gian xác định, điều này giúp ZK nhận biết được việc mất kết nối / bị sập của nodes

ra tình trạng tắc nghẽn làm mất mác dữ liệu hoặc sập hệ thống. *Do đó, một hệ thống mạnh mẽ phải có khả năng giải quyết khi xảy ra việc bùng nổ dữ liệu. Bên cạnh đó, ta cần một nơi làm việc như một nguồn dữ liệu cục bộ để cấp phát dữ liệu liên tục cho nhiều ứng dụng khác nhau trong hệ thống*. Dựa trên những nhu cầu đó, Apache Kafka ra đời nhằm giải quyết những vấn đề trên.

❖ ***Apache Kafka***

Apache Kafka được phát triển bởi LinkedIn *nhằm giúp việc truyền tải dữ liệu giữa những dịch vụ web và kho dữ liệu bên trong hệ thống. Mục tiêu của Kafka là để giải quyết các vấn đề trong việc truyền tải dữ liệu lớn với tốc độ cao giữa nhiều dịch vụ khác nhau trong hệ thống khai thác Data Stream*. Sau đây, chúng tôi sẽ giới thiệu ngắn gọn những thành phần quan trọng trong một Kafka cluster.



Hình 3-3: Mô hình kiến trúc của một Kafka single node [9]

Hình 3-3 là mô hình Kafka single node, mô hình gồm 4 thành phần chính là ZooKeeper, Producers, Consumers, và Brokers. Kafka cluster khi triển khai luôn cần sự giúp đỡ từ ZooKeeper để trao đổi thông tin giữa Brokers và Consumers. Dữ liệu sẽ được phun ngẫu nhiên từ các Producers lên hệ thống Kafka để thực hiện phân luồng dữ liệu và trả về cho các Consumer đang lắng nghe bên trong hệ thống. Khi dữ liệu được truyền lên phải được gắn vào một topic cụ thể, topic này sẽ được Kafka lưu trữ bên trong ZooKeeper để có thể đồng bộ với các Nodes khác.

Broker	Thành phần xử lý vật lý, những Brokers này làm nên một Kafka cluster. Thông thường, một Broker tương ứng với 1 server vật lý
Partition	Được dùng để tổ chức một topic cho hợp logic. Mỗi Broker sẽ chứa 1 hay nhiều partitions. Khi producer thực hiện việc phun dữ liệu, Kafka sẽ tự động chọn partitions để ghi dữ liệu vào ổ cứng vật lý tại Broker đó
Producer	Thành phần thu thập và phun dữ liệu lên các Brokers, nó có thể lấy hoặc nhận dữ liệu từ các nguồn bên ngoài như Log-file…
Consumer	Thành phần nhận dữ liệu từ các Broker trả về. Thông thường các consumers sẽ được tính hợp vào trong thành phần xử lý dữ liệu
Topic	Thành phần tổ chức dữ liệu trong Kafka, nơi chứa những dữ liệu có liên quan mật thiết với nhau. Một topic có thể lưu trong nhiều partitions trên những server khác nhau nhằm phục vụ quá trình xử lý song song

Kafka không phải là một hệ thống truy vấn trên dữ liệu Log, và khi đã tiêu thụ dữ liệu, thì dữ liệu đó sẽ không bị xóa như những hệ thống khác. Như ta đã biết, một topic có thể có nhiều partitions và việc ghi dữ liệu vào những partitions này thì không theo một thứ tự nhất định. Vì vậy, dữ liệu được Kafka trả về sẽ không đảm bảo thứ tự ngữ nghĩa.

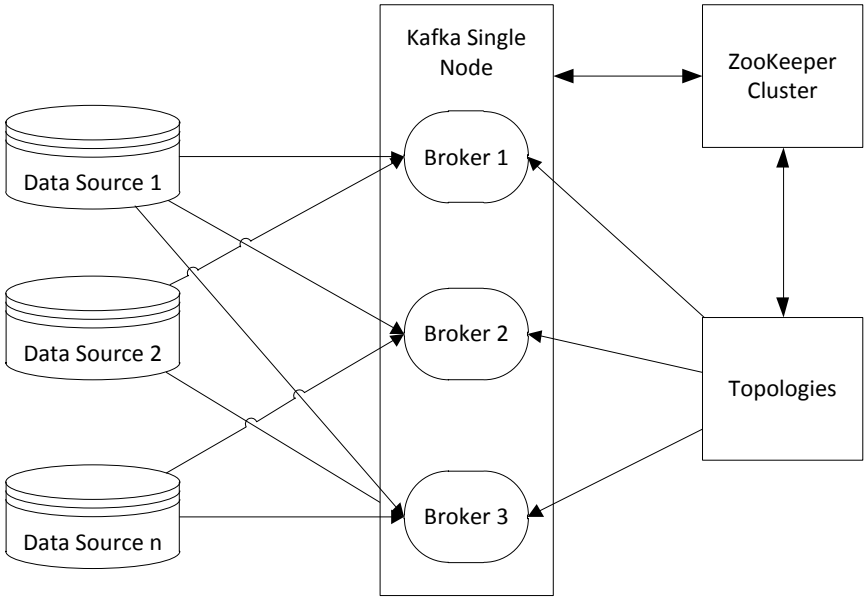
❖ ***Apache Kafka trong hệ thống khai thác Data Stream***

Mô hình Kafka được thiết kế trong kiến trúc khai thác Data Stream tương tự như mô hình mẫu được trình bày bên trên, *với 3 broker được cài đặt trong một node*, mỗi *broker sẽ quản lý 1 partition*. Điều này nhằm mục đích làm giảm thời gian phản hồi của Kafka với những ứng dụng tiêu thụ dữ liệu như Storm, và tăng khả năng chịu tải của Kafka. Mô hình được trình bày như *hình 3-4*.

Trong mô hình trên, dữ liệu đi từ nhiều nguồn khác nhau và cùng đỗ về 3 Brokers trong Kafka server. Kafka hổ trợ 2 cơ chế để phun dữ liệu từ các Producers lên Brokers là *async*, và *sync*. Cơ chế *sync* là cơ chế gửi gói tin đảm bảo, nghĩa là khi nhận được các *message* [18] gửi lên Broker sẽ gửi xác nhận quá trình gửi thành công

[18] Message là gói tin được truyền tải trong Kafka cluster, nó có thể chứa 1 hoặc nhiều requests

cho các Producers, trong thời gian này thì Producer sẽ chờ tới khi nhận xác nhận gửi thành công mới thực hiện tiếp. Trong khi đó, *async* không cần chờ việc hồi đáp từ Broker, nó thực hiện gửi message liên tục lên các Brokers. Vì vậy, thời gian gửi message bằng giao thức sync sẽ lâu hơn nhiều so với async. Tuy nhiên, việc gửi bằng *async* sẽ bị độ nhiễu cao, và dễ mất message.



Hình 3-3: Mô hình thiết kế Kafka trong hệ thống

Mô hình trên thiết kế với 3 Brokers để nhằm tăng khả năng nhận các messages từ các Producers gửi về hệ thống. Việc nay làm tăng tốc độ truyền tải dữ liệu vào bên trong hệ thống gấp 3 lần, cho việc dù ta dùng cơ chế *async* hay *sync* đều vậy.

Topologies là những ứng dụng dùng để xử lý Data Stream được cài đặt bên trong Storm cluster. Những topologies này được cài đặt thành phần Kafka consumer vào bên trong để thực hiện việc hút dữ liệu từ Kafka cho quá trình phân tích dữ liệu. Những thông tin cấu hình, tạo kết nối giữa Topology và Kafka sẽ được ZooKeeper hổ trợ thông qua Metadata bên trong ZooKeeper.

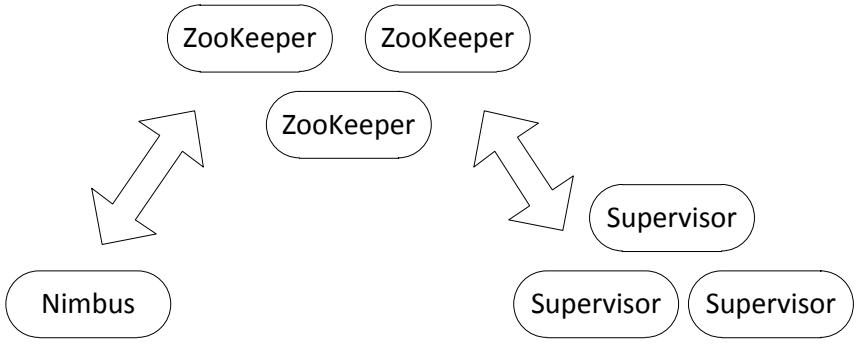
3.2.3. Hệ thống xử lý Data Stream

Có rất nhiều phương pháp khác nhau để phân tích Data Stream, một trong những phương pháp đơn giản đó là xây dựng một ứng dụng phân tích Data Stream trực tiếp mà không cần sự trợ giúp của những công nghệ hiện đại. Tuy nhiên, phương pháp

này sẽ không khả thi trong việc phân tích dữ liệu lớn, mặc dù phương pháp này tuy đơn giản những lại tìm ẩn nhiều nguy cơ. Một phương pháp tiên tiến hơn là ứng dụng Hadoop để khai thác dữ liệu lớn thông qua cơ chế Map-Reduce. Mặc dù phương pháp này làm việc rất hiệu quả trong hầu hết các trường hợp, nhưng lại bị điểm khuyết vì độ trễ quá cao trong việc thực thi 1 job. Điều này làm mất đi tính chất quan trọng high-availability của một hệ thống phân tích Data Stream. Vì vậy, *ta cần một hệ thống có khả năng xử lý dữ liệu lớn trong thời gian thực*.

❖ ***Xử lý dữ liệu với Apache Storm***

Apache Storm là một framework mã nguồn mở, nó là một *hệ thống tính toán phân tán dữ liệu lớn trong real-time*. Storm giúp việc tính toán song song trên Data Stream được dễ dàng và hiệu quả, và có thể được triển khai trên nhiều ngôn ngữ khác nhau. Storm có 4 đặc tính nổi bật *[6]* là *khả năng mở rộng và sử lý dữ liệu lớn, khả năng chịu lỗi cao*, và *đảm bảo việc xử lý trọn vẹn dữ liệu*.



Hình 3-4: Mô hình tổng quan của Storm Cluster [1]

Storm cluster được cấu thành từ 3 thành phần chính là Nimbus, Supervisors, và ZooKeeper cluster như hình 3-5. Hình bên dưới trình bày mô hình tổng quan của một Storm cluster với 3 nodes ZooKeeper và 3 Supervisors. Trong đó, ZooKeeper đóng vai trò là nơi hỗ trợ Nimbus quản lý các Supervisors. ZooKeeper làm nhiệm vụ lưu vết lại các thông tin của các mẫu Topologies hoạt động trên Storm, và lưu trữ tình trạng hoạt động của các Supervisors.

Trước hết, chúng tôi sẽ giới thiệu một số khái niệm quan trọng trong Storm cluster [6].

Nimbus: là một thành phần quan trọng nhất trong hệ thống Storm cluster, nó đảm nhận nhiều công việc khác nhau như là:

- Chịu trách nhiệm cho việc chia tán công việc cụ thể cho các spouts hoặc bolts thông quá các workers trong cluster
- Bên cách đó, nó còn chịu trách nhiệm tái cân bằng storm cluster trong trường hợp một trong những supervisor bị sập
- Nimbus không tham gia trực tiếp việc xử lý của các supervisors, mà nó chỉ phân chia công việc cho các supervisors
- Trong trường hợp Nimbus bị sập, storm cluster sẽ không còn khả năng quản lý các Topologies và các Supervisors, vì thế ta phải khởi động lại Nimbus

Supervisor: là những nodes con được quản lý bởi Nimbus, đây là nơi thực hiện các công việc xử lý dữ liệu cho toàn bộ hệ thống

- Supervisors quản lý các workers, nhằm đảm bảo workers luôn hoạt động
- Nimbus thực hiện việc phân chia công việc cho các worker, và những Supervisors thực hiện việc quản lý những workers này

Bảng 3-1 bên dưới trình bày một số khái niệm khác trong Storm cluster, 8 thành phần bên dưới là những khái niệm cơ bản trong Storm. Những khải niệm này để hổ trợ cho những thảo luận sâu hơn bên dưới về mô hình thiết kế Storm cluster và mô hình triển khai các ứng dụng phân tích Data Stream với Storm cluster trong những mục bên dưới.

Topology	Một ứng dụng logic cho việc xử lý Data Stream sẽ được đóng gói bên trong một Storm topology. Topology tương tự như một *MapReduce job*, nhưng Topology chạy mãi không bao giờ dùng, trong khi đó MapReduce sẽ dùng khi thực hiện hết công việc Topology chứa các *spouts* và *bolts*, những thành phần này được liên kết với nhau bởi *Stream grouping*. Topology là mẫu thiết kế cơ bản của Storm so với Trident

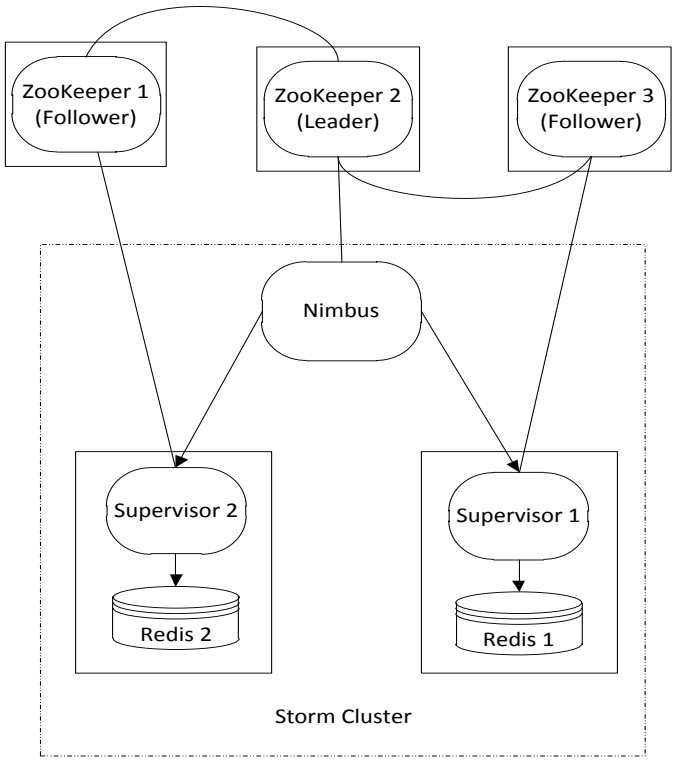
Trident	Trident là một mẫu thiết kế logic trừu tượng cho việc tính toán real-time dựa trên đỉnh của Storm. Nó cho phép xử lý dữ liệu với tốc độ cao, và hổ trợ truy vấn phân tán với độ nhiễu thấp, xử lý thống nhất trên nhiều nodes, và khả năng đảm bảo xử lý dữ liệu 1 lần duy nhất Trident hổ trợ một số hàm thực thi như joins, aggregations, grouping, function, những hàm này sẽ được trình bày trong những mục sau
Tuple	Tuple là một đối tượng chứa dữ liệu được truyền và xử lý trong nội bộ Storm cluster
Spouts	Một spout là một nguồn dữ liệu trong mẫu Topology. Nó thực hiện việc đọc dữ liệu từ những nguồn bên ngoài và phun vào Topology
Bolts	Là thành phần xử lý dữ liệu trong Topology, sau khi được spout cung cấp dữ liệu
Stream grouping	Stream grouping định nghĩa việc làm thế nào để phân chia Data Stream giữa các Bolts
Worker	Một Topology thực hiện việc xử lý thông qua một hoặc nhiều worker. Từng worker là một đơn vị xử lý vật lý *JVM*[19]. Một Supervisor (*physical machine*) có một hoặc nhiều JVM
Task	Từng spout hoặc bolt sử lý nhiều tasks thông qua Storm cluster. Mỗi task tương ứng với một thread xử lý

Bàng 3-1: Một số khai niệm trong Storm Cluster

❖ ***Apache Storm trong hệ thống khai thác Data Stream***

Trong mô hình thiết kế như hình 3-6 bên dưới, chúng tôi xây dựng 1 Storm cluster với 3 nodes trên 3 server vật lý, 1 node dùng làm Nimbus và 2 nodes làm Supervisors. Những nodes này được giao tiếp với nhau thông qua ZooKeeper cluster được trình bày bên trên. Bên cạnh đó để hổ trợ việc lưu trữ lại kết quả sau khi xử lý chúng tôi tích hợp thêm vào mỗi Supervisor một RedisDB, phần này sẽ được trình bày chi tiết hơn ở mục tiếp theo.

[19] JVM (java virtual machine): máy ảo java chạy trên từng server, một server vật lý có 1 hoặc nhiều jvm để thực thi các tiểu trình



Hình 3-5: Mô hình thiết kế Storm cluster trong hệ thống

Trong mẫu thiết kế này, chúng tôi chỉ dùng 2 nodes làm supervisors, và trong mỗi supervisor tích hợp thêm 1 RedisDB. Tuy nhiên, trong thực tế tùy vào nhu cầu thực tế mà ta có thể thêm vào nhiều nodes hơn để nhằm gia tăng hiệu năng của hệ thống xử lý Data Stream. Việc thêm node này được thực hiện một cách dễ dàng bằng việc ta cấu hình dịch vụ, và gọi thực tái cân bằng (*rebalance*) từ command line thì Nimbus sẽ tự động thêm node vào trong cluster. Tuy nhiên, một điều ta nên lưu ý về khả năng chịu tải của ZooKeeper, ở đây chúng tôi chỉ dùng 3 node làm ZooKeeper cluster vì vậy khi thêm nhiều nodes hơn nữa nên chú ý tới việc thêm một số node cho ZooKeeper cluster để tăng khả năng chịu tải của nó.

Một điểm đáng chú ý trong hệ thống Storm cluster này là khả năng *fault-tolerant*. Khác với Hadoop, trong trường hợp JobTracker sập thì toàn bộ công việc đang thực thi sẽ bị mất, nhưng trong Storm cluster khi Nimbus sập thì các công việc vẫn được thực thi bình thường. Bởi vì, thay vì Nimbus quản lý tất cả các workers như JobTracker trong Hadoop thì Supervisor sẽ đảm nhận việc này. Tuy nhiên, trong trường hợp Supervisor bị sập thì tất cả các công việc đang được xử lý trong node đó

sẽ bị mất, và Nimbus sẽ thực hiện việc phân lại những việc đó cho những workers khác.

3.2.4. Hệ thống lưu trữ dữ liệu phân tán

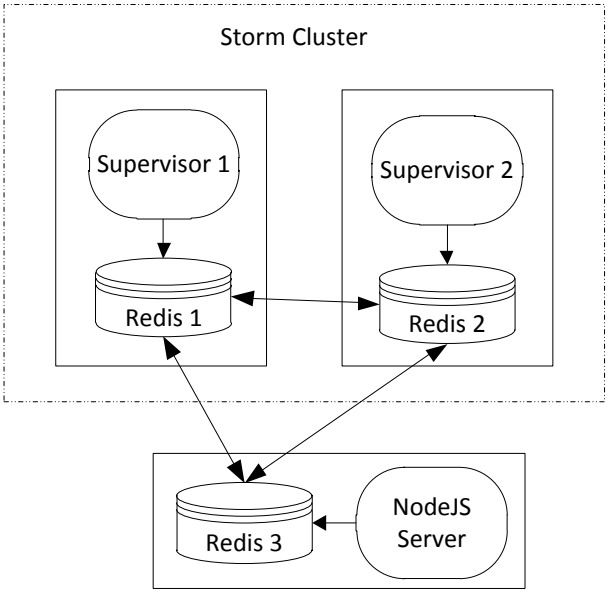
Như đã trình bày bên trên về khả năng xử lý dữ liệu lớn thơi gian thực của Storm cluster. Nhưng để làm được điều này ta phải *đảm bảo rằng việc lưu trữ dữ liệu sau khi đã được xử phải được thực một cách nhanh nhất có thể, để đảm bảo rằng nó không làm gián đoạn việc xử lý của toàn hệ thống*. Bởi vì, trong mô hình Trident thực hiện việc xử lý dữ liệu đảm bảo, một task chỉ được thực thi khi task trước đó đã được thực thi xong. Vì thế, ta cần một cơ sở dữ liệu có khả năng lưu trữ dữ liệu nhanh với độ nhiễu thấp.

Trong một số mẫu thiết kế trước đây, dữ liệu thường được lưu trữ bởi một cơ sở dữ liệu bên ngoài hệ thống. Điều này làm độ trễ cao do chi phí thời gian truyền dữ liệu giữa các server với nhau, bên cạnh đó để đảm bảo không bị mất mác dữ liêu khi truyền thường các frameworks sẽ thực hiện việc cơ chế *sync*[20]. Do đó, thời gian truyền tải và lưu trữ dữ liệu sẽ lâu hơn thời gian Storm xử lý 1 đơn vị dữ liệu. Dựa trên những khuyết điểm này, chúng tôi đóng góp một mô hình lưu trữ dữ liệu phân tán thông qua công cụ Redis Cluster. Những cơ sở dữ liệu Redis này được cài đặt ngay bên trong những con server chạy Supervisor, việc này làm cho giảm đáng kể chi phí truyền dữ liệu, nhằm tăng hiệu năng lưu trữ cho các workers. Tuy nhiên, trong hệ thống xử lý dữ liệu phân tán ta sẽ gặp vấn đề về việc đồng bộ các cơ sở dữ liệu Redis lại với nhau thành một cơ sở dữ liệu thống nhất. Hình 3-7 là mô hình thiết kế Redis Cluster được tích hợp vào bên trong hệ thống xử lý Data Stream.

Trong phiên bản mới công bố [8], Redis chính thức hỗ trợ khả năng truy vấn dữ liệu trong một cụm các cluster được phân tán trên nhiều server khác nhau mà vẫn giữ được hiệu năng cao. Nhờ vào sự phát triển này, *vấn đề thống nhất dữ liệu trên nhiều nodes khác nhau đã được giải quyết*. Hơn nữa, *việc này giúp cho truy vấn dữ liệu trên Storm cluster dễ dàng hơn*, bởi vì các RedisDB được cài đặt ngay trong các server

[20] Sync cơ chế truyền tải dữ liệu đồng bộ, việc truyền 1 đơn vị dữ liệu chỉ kết thúc khi người gửi nhận được thông báo lưu trữ dữ liệu thành công từ server lưu trữ.

thực thi của Storm nên *giảm được chi phí truyền tải dữ liệu, khả năng đọc ghi bằng Redis nhanh hơn hẳn các cơ sở dữ liệu khác*.



Hình 3-6: Mô hình thiết kế Redis cluster trong hệ thống

Trong mô hình được trình bày bên trên, chúng tôi cài đặt Redis trên server chạy NodeJS để thực hiện việc truy vấn dữ liệu nhanh hơn, và nó là cầu nối với những cơ sở dữ liệu khác nằm trong những Supervisors của hệ thống Storm cluster.

3.2.5. Thành phần phân tán dữ liệu

NodeJS server chịu trách nhiệm lấy dữ liệu từ Redis Cluster và dùng những thông tin này trình diễn ra những real-time report tương ứng. Để giao tiếp giữa webserver và client, chúng tôi sử dụng thư viện JavaScript *socket.io*[21]. Trong khi web socket, một giao thức bắt tay giữa client và server, vẫn còn đang phát triển và còn nhiều hạn chế [10] như là chỉ được hỗ trợ cho một số trình duyệt mới và vấn đề khác firewalls và proxies ngăn chặn kết nối của web socket. Thư viện socket.io có thể chạy tốt trên hầu hết các trình duyệt được sử dụng phổ biến hiện tại và không bị chặn bởi proxies hay firewalls. Bên cạnh đó, socket.io cũng hỗ trợ nhiều API giúp cho việc lập trình trở nên dễ dàng hơn.

[21] Socket.io: http://socket.io/

Chúng tôi sẽ sơ lược về cách hệ thống sẽ phân tán dữ liệu. Giả sử, hệ thống Data Stream hoạt động và dữ liệu được cập nhật liên tục vào Redis Cluster. Sau mỗi thời gian ngắn, Redis sẽ sử dụng cơ chế *publisher/subscriber* để thông báo về dữ liệu mới cập nhật cho NodeJS server, mà đang lắng nghe thông tin từ Redis. Khi nhận được thông tin cập nhật, với cơ chế event-driven, NodeJS sẽ gọi hàm callback để broadcast dữ liệu mới cập nhật cho tất cả client đang kết nối thông qua socket.io. Dữ liệu mới được client trình diễn ra dưới dạng real-time report.

3.3. Cài đặt và tích hợp thuật toán

Trong mục này, chúng tôi sẽ trình bày mô hình tích hợp 2 thuật toán khai thác tập phổ biến trên Data Stream vào hệ thống đã được trình bày trong những mục trên. *Mô hình này được xây dựng dựa trên mẫu thiết kế Trident với tính năng lưu trữ và truy vấn dữ liệu trên hệ thống phân tán thông qua Redis Cluster*. Nội dung của 2 thuật toán tích hợp được trình bày chi tiết trong chương 5 và 6.

3.3.1. Mô hình triển khai thuật toán trên hệ thống Storm cluster

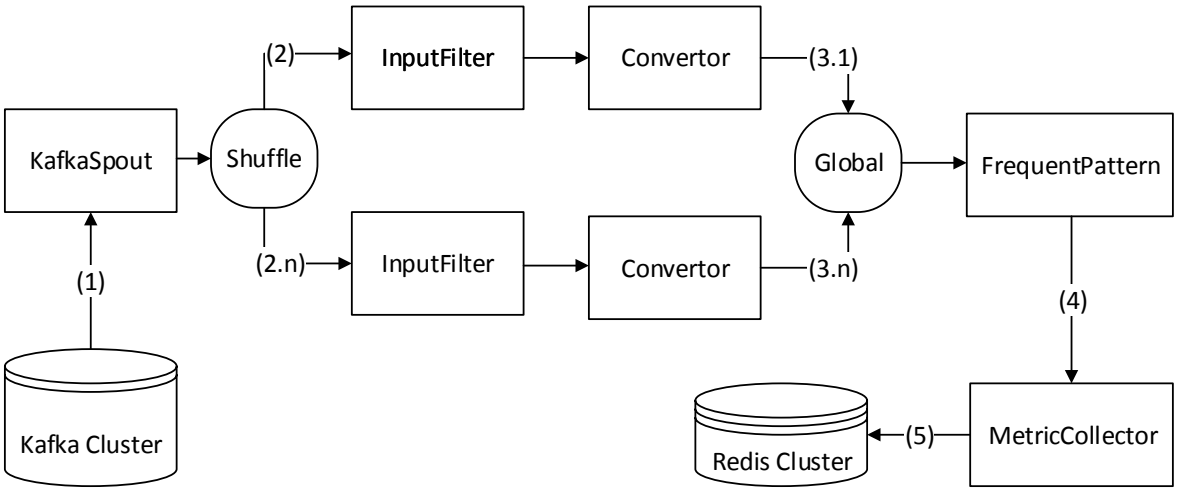
Mô hình tích hợp thuật toán được chúng tôi thiết kế trên mẫu Trident, *việc làm này đem lại 3 ưu điểm* [6]:

- *Truyền tải và xử lý dữ liệu nhanh* với mẫu Trident (*high throughput*)
- Trident hỗ trợ cơ chế *OpaqueTridentKafkaSpout* giúp đảm bảo tất cả dữ liệu được xử lý một lần duy nhất. Điều này giúp *tránh việc xử lý trùng lập dữ liệu*
- Trident hỗ trợ nhiều cơ chế mạnh mẽ khác của Storm, như việc truy vấn trực tuyến trên Data Stream bằng DRPC với độ trễ thấp. Bên cạnh đó, *Trident có khả năng tích hợp những thuật toán máy học* vào bên trong, thông qua bộ thư viện TridentML.

Trong mô *hình 3-8* bên dưới mô tả quá trình xử lý dữ liệu theo mẫu Trident, dữ liệu được truyền từ *KafkaCluster* vào Storm thông qua thành phần *KafkaSpout*. Sau đó được phân tán một cách ngẫu nhiên ra những *phân vùng* [22]khác để xử lý. Sau đó

[22] Phân vùng (Partition): là một nhánh xử lý dữ liệu trong Storm. Để xử lý dữ liệu song song tại nhiều workers khác nhau, Storm tạo nhiều Partition để xử lý, và dùng các Operators để quản lý việc truyền dữ liệu

được đổ về chung một nơi để thực hiện việc khai thác tập phổ biến tại lớp *FrequentPattern*. Thành phần *MetricCollector* sẽ thực hiện việc thu thập những thông tin khác mà không cần đến những thuật toán phức tạp, như việc đếm số lượng tuple đã xử lý, lượng bộ nhớ được sử dụng,... Cuối cùng, kết quả của việc phân tích sẽ được lưu trữ vào bên trong Redis Cluster để phục vụ cho những mục đích khác. Việc sử dụng *Redis Cluster* sẽ rất có lợi vì *giảm chi phí truyền tải dữ liệu trên đường truyền, tốc độ lưu trữ cao*, và *vẫn đảm bảo tính nhất quán dữ liệu giữa các Nodes khác nhau*.



Hình 3-7: Mô hình tích hợp thuật toán vào Storm theo mẫu thiết kế Trident

Như đã thấy, tại *giai đoạn (2) ta có thể khai báo cho Storm thực hiện việc sử lý song song trên nhiều executors [23] khác nhau để tăng hiệu năng* sử lý dữ liệu lớn. Tuy nhiên, *giai đoạn (3) tất cả kết quả đã xử lý phải được kết hợp lại thành một phân vùng* để thực hiện việc khai thác tập phổ biến. *Việc này là một khuyết điểm lớn*, nó làm giảm hiệu năng của việc xử lý song song của Storm. Lý do chúng tôi làm vậy bởi *vì 2 thuật toán được trình bày tại chương 5, 6 không hổ trợ việc sử lý song song*, cho nên tất cả dữ liệu đã được xử lý trong những thành phần trước phải được đổ về 1 nơi để thực hiện việc khai thác dữ liệu.

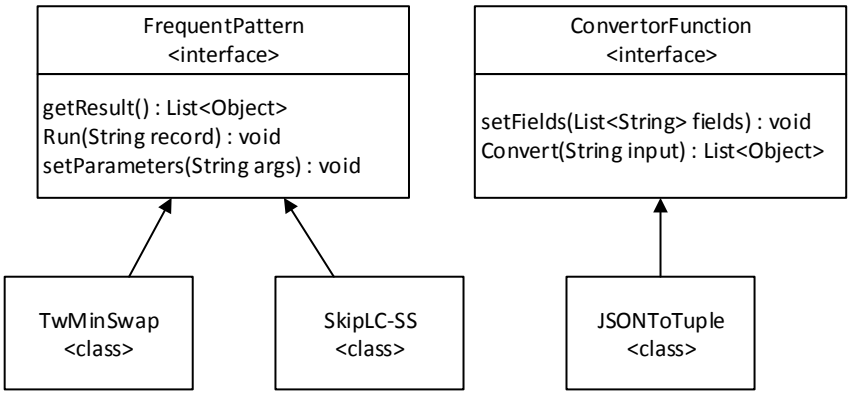
[23] Executor (thread): là dòng xử lý trong Storm, một executor tương đương một Thread

3.3.2. Cài đặt tích hợp thuật toán vào trong hệ thống

Dựa vào mô hình triển khai được trình bày trong ục 3.3.1 bên trên, *chúng tôi đã xây dựng một bộ thư viện plug-in bằng ngôn ngữ Java để tích hợp những thuật toán mới về khai thác mẫu phổ biến trên Data Stream*.

Thông qua việc cấu hình từ file *storm-fp.properties* bên ngoài, chúng ta có thể dễ dàng tùy chỉnh các thông tin thiết lập cho Storm, Redis, và Kafka mà không cần quan tâm nhiều tới việc sử dụng những framework này ra sao, và làm sao để kết hợp những framework này lại thành một project hoàn chỉnh để thực thi trên Storm cluster.

Hơn nữa, để tăng sự linh động trong việc triển khai, chúng tôi đã xây dựng nhiều *interface* để ta có thể *dễ dàng override lại các lớp* InputFilter, Convertor, FrequentPattern, và MetricCollector. Bộ thư viện này, mặc định dữ liệu là vào là kiểu JSONObject hoặc JSONArray vì đây là 2 loại kiểu dữ liệu rất phổ biến trong các bộ dữ liệu thử nghiệm về item/itemset mining, tuy nhiên tùy vào mục đích ta có thể override lại hàm chuyển đổi dữ liệu đầu vào này. Một số interface cho trong bộ thư viện tích hợp được trình bày trong *hình 3-8* bên dưới.



Hình 3-8: Một số interface trong bộ thư viện tích hợp

Thông qua việc khảo sát nhiều thuật toán phổ về khai thác mẫu phổ biến. Chúng tôi thấy rằng những thuật toán này cần truyền một hoặc một vào các siêu tham số để thực thi thuật toán, đầu ra của những thuật toán này thường là một danh sách các item/itemset và độ phổ biến của chúng. Từ đó, chúng tôi xây dựng 1 lớp chung cho các thuật toán để dễ dàng hơn khi tích hợp vào Storm.

3.4. Tổng kết

Thông qua nền tảng kiến trúc tổng quan của tài liệu [1], những đặc điểm của một Streaming System, những đặc trưng dữ liệu và việc khảo sát, phân tích nhiều framework phổ biến hiện này. *Chúng tôi đã đề xuất ra một kiến trúc cho hệ thống phân tích Data Stream.* Và để cải tiến hiệu năng về lưu trữ, và truyền tải dữ liệu ra các cơ sở dữ liệu bên ngoài hệ thống, chúng tôi đã tích hợp một hệ thống lưu trữ dữ liệu mới là Redis cluster, điều này giúp giảm chi phí truyền tải dữ liệu, và tăng tốc độ lưu trữ và truy vấn dữ liệu. Hơn nữa, *chúng tôi đã xây dựng một bộ thư viện plug-in để người dùng có thể dễ dàng cài đặt và tích hợp những thuật toán mới về khai thác mẫu phổ biến trên Data Stream*, dựa trên mô hình triển khai thuật toán bên trên.

Trong chương tiếp theo, khóa luận trình bày về một số khái niệm trong lĩnh vực khai thác mẫu phổ biến trên Data Stream, một số thuật toán nền, và hướng tiếp cận của luận văn. Để từ đó dẫn nhập tới 2 paper chúng tôi đã nghiên cứu về việc khai thác item phổ biến và itemset phổ biến.

CHƯƠNG 4: TỔNG QUAN KHAI THÁC MẪU PHỔ BIẾN TRÊN DATA STREAM

Trong chương này, chúng tôi sẽ trình bày tổng quan một số định nghĩa, vấn đề gặp phải trong việc khai thác data stream và ý nghĩa của nó. Chi tiết hơn nữa, chúng tôi trình bày về khái niệm data windows và định nghĩa hình thức cho bài toán khai khai thác mẫu phổ biến. Hai bài toán của khai thác mẫu phổ biến là khai thác item và itemset phổ biến, là đối tượng của khóa luận, cũng sẽ được trình bày trong chương này.

4.1. Giới thiệu

4.1.1. Mẫu phổ biến

Mẫu phổ biến (Frequent Pattern) [11] *là các mẫu (tập các item, chuỗi con, các cấu trúc con, đồ thị con, ...) xuất hiện trong tập dữ liệu một cách thường xuyên với tần suất không nhỏ hơn một ngưỡng cho trước*. Ví dụ, một tập các item gồm sữa và bánh mì thường xuyên xuất hiện cùng nhau trong một tập dữ liệu lớn (lớn hơn n lần) được gọi là mẫu phổ biến. Một dãy tuần tự các hành vi, chẳng hạn như "mua máy tính trước, sau đó mua máy camera kỹ thuật số, rồi đến thẻ nhớ", nếu chuỗi hành vị này thường gặp trong cơ sở dữ liệu của cửa hàng thì đó là một dãy tuần tự phổ biến. Một cấu trúc như đồ thị, cây, dạng lưới xuất hiện thường xuyên trong dữ liệu, thì nó được gọi là mẫu có cấu trúc phổ biến.

Việc tìm kiếm những mẫu phổ biến như vậy đóng vai trò quan trọng trong việc khai thác các luật kết hợp [12, 13]. Hơn thế nữa, nó còn giúp giải quyết các bài toán như phân loại, phân nhóm và nhiều bài toán KHAI THÁC DỮ LIỆU khác [14, 15, 16]. Chính vì vậy, việc khai thác mẫu phổ biến hiện nay là một bài toán rất quan trọng trong lĩnh vực KTHL và là một chủ đề đáng quan tâm khi nghiên cứu về KHAI THÁC DỮ LIỆU.

4.1.2. Việc khai thác mẫu phổ biến trong Data Stream

Việc khai thác mẫu phổ biến là một nghiên cứu để tìm ra các mẫu xuất hiện nhiều lần trong một bộ dữ liệu. Bộ dữ liệu này có cấu trúc đa dạng, như là bộ dữ liệu text

document, bộ dữ liệu bán cấu trúc như là XML, hay bộ dữ liệu có cấu trúc phức tạp như là Graph. Ví dụ, ta cho một bộ dữ liệu về giao dịch, từng giao dịch chứa một itemset, khai thác mẫu phổ biến là việc tìm kiếm các các cặp item/itemset khác nhau mà xuất hiện nhiều hơn một giá trị cho trước gọi là độ support.

Trong môi trường data stream, bài toán khai thác mẫu phổ biến là việc xử lý một chuỗi các phần tử một lần duy nhất và tìm ra các phần tử xuất hiện nhiều hơn số lần xuất hiện cho trước (thường là tỉ lệ phần trăm của kích thước dòng dữ liệu tính đến thời điểm hiện tại). Việc khai thác mẫu phổ biến trên dữ liệu dòng đặt ra 3 thách thức lớn [17], đó là:

- Số lượng mẫu phát sinh tăng theo cấp số nhân, đòi hỏi phải có một giải thuật mới để khai thác như là việc tính toán xấp xỉ
- Lượng bộ nhớ xử dụng phải được quản lý một cách hiệu quả, vì những thuật toán khai thác dòng đòi hỏi nhiều bộ nhớ hơn những thuật toán khai thác trên dữ liệu tĩnh
- Những thuật toán phải cân bằng giữa thời gian xử lý và sử dụng bộ nhớ

4.1.3. Ý nghĩa của việc khai thác tập mẫu phổ biến trên Data Stream

Ngày nay, ngày càng nhiều các ứng dụng như là mạng lưới giao thông, mạng xã hội, hệ thống giám sát và quá trình xử lý dữ liệu online như là thị trường chứng khoáng hay đơn thanh toán của các cửa hàng bán lẻ … tạo ra một lượng lớn dữ liệu dòng mỗi ngày. Việc tìm ra những mẫu phổ biến ẩn bên trong những dữ liệu này là cần thiết. *Khác với cơ sở dữ liệu truyền thống, dòng dữ liệu điển hình tới một cách liên tục ở tốc độ cao với một lượng lớn và phân phối dữ liệu cũng thay đổi. Vì đặc trưng này của data stream, những kỹ thuật cho dữ liệu truyền thống mà yêu cầu nhiều lần duyệt toàn bộ dữ liệu không thể áp dụng trực tiếp vào việc khai thác data stream, mà luôn luôn chỉ cho phép 1 lần duyệt dữ liệu*. Một hướng nguyên cứu về khai thác mẫu phổ biến trên dòng dữ liệu đã ra đời và thu hút nhiều sự quan tâm của những nhà khoa học vì tính thử thách cũng như sự cần thiết của nó. Đã có một số công trình nguyên cứu được đề xuất áp dụng những kết quả nguyên cứu về khai thác mẫu phổ

biến vào ứng dụng thực tế. Sau đây là ba ứng dụng chính của khai thác mẫu phổ biến [16].

Theo dõi hiệu năng: Theo dõi giao thông mạng và hiệu năng, xác định dị thường và sự xâm nhập, và dự đoán về ước lượng sự phổ biến của dòng gói tin trên internet khi có sự cố báo động từ dòng dữ liệu

Theo dõi giao dịch*:* Theo dõi nhưng giao dịch trong cửa hàng bán lẻ, máy ATM, và bản ghi Log thị trường chứng khoáng, bản ghi các cuộc gọi. Tìm mẫu phổ biến được áp dụng để theo dõi dòng nghiệp vụ để dự đoán sai sót hay tạo những report dựa trên dòng web log.

Khai thác mạng lưới sensor: Khai thác những mẫu phổ biến từ mạng lưới sensor hay camera giám sát và còn có thể ước lượng được giá trị thiếu.

4.2. Data Windows

Trong môi trường data stream, những chuỗi đối tượng dữ liệu, dạng $T=(T_1, T_2, T_3,...,T_i,....)$, đến liên tục và không kết thúc. Kết quả của quá trình khai thác dựa trên những *windows data* được cập nhật liên tục theo thời gian. Một window là một chuỗi các giao dịch giữa *i-th* và *j-th*, được ký hiệu như sau $T_{i,j} = (T_i , T_{i+1} ,...., T_j)$ với $i <= j$. Mặc dù có rất nhiều cách để xác định một mô hình *window*, nhưng có 4 loại mô hình window phổ biến nhất là: Landmark window, Sliding window, Damped window, Time-tilted window [4].

- **Landmark Window**: trong mô hình này chúng ta tìm kiếm mẫu phổ biến chứa trong window bắt đầu từ lúc khởi tạo *Data Stream (s)* tới thời điểm hiện tại ta xét *(t)*
- **Sliding Window**: cho kích thước của một window (*w)* và thời gian hiện tại (*t).* Việc tìm những mẫu phổ biến theo mô hình này sẻ chỉ được xét trong window [$t – w +1, t$], window này sẽ được thay đổi liên tục theo thời gian *t* nhưng vẫn giữ nguyên giá trị *w*
- **Damped Window**: mô hình này vừa cân nhắc những dữ liệu trong quá khứ, vừa cân nhắc những dữ liệu hiện tại nhưng ở những mức độ khác nhau. Nó phân phối trọng số cao hơn cho những dữ liệu gần đây và giảm trọng số những

dữ liệu trước đó, bằng cách xác định một *decay rate δ (*với *0 < δ <= 1*). Khi một dữ liệu mới tới, những trọng số của dữ liệu trước đó sẽ nhân *δ* để giảm mức độ ảnh hưởng của nó.

- **Time-Tilted Window**: việc khai thác mẫu phổ biến trên mô hình này dựa trên một bộ những window có độ dài khác nhau, và từng window tương ứng với những thời giản khác nhau.

4.3. Khai thác mẫu phổ biến: Định nghĩa

Mặc dù các khảo sát hay các bài tổng quan về khai thác mẫu phổ biến chỉ tập trung chỉ khai thác itemset phổ biến. *Chúng tôi đưa ra một định nghĩa hình thức cho vấn đề khai thác mẫu phổ biến cho tất cả các mẫu bao gồm items, itemsets, chuỗi, cây và đồ thị* [4].

Cho $X = \{x_1, x_2, \ldots, x_m\}$ là tập có thể có của items x_i. Một mẫu P là một chuỗi hay là một tập các items, với P là tất cả các mẫu P có thể mà chúng ta đang quan tâm. Một dòng dữ liệu T là một chuỗi các giao dịch, ví dụ $T = \{T_1, T_2, T_3 \ldots\}$. Dòng dữ liệu có chiều dài vô hạn. Tại thời điểm *j*, data window $T_{i,j}$ là một chuỗi các giao dịch từ thời điểm *i* trước đó tới thời điểm hiện tại: $T_{i,j} = \{T_i, T_{i+1}, T_{i+2}, \ldots, T_j\}$. Vì các mẫu nhỏ hơn có thể là tập con của các mẫu lớn hơn, bất kì data window có thể chứa lượng lớn các mẫu. Cho *Patt(T)* là một hàm liệt kê tất cả mẫu con mà nằm trong T. Độ hổ trợ s của mẫu con P trong dòng dữ liệu T được tính bởi:

$$s(P) = \frac{count(P, Patt(T))}{|T|}$$

Nơi *count(P,Patt(T))* là số lần mẫu P xuất hiện trong tập các mẫu con *Patt(T)*. Sau đó, cho một hổ trợ tối thiểu $\sigma, 0 < \sigma < 1$, mẫu P là mẫu phổ biến của T nếu và chỉ nếu $s(P) \geq \sigma$. Vấn đề khai thác mẫu phổ biến trong data stream là tìm kiếm tập tất cả các mẫu phổ biến $P \in P$ được chứa trong data window $T_{i,j}$. một biến thể khác là khai thác top-k mẫu phổi biến, tìm k mẫu có độ phổ biến cao nhất bỏ qua độ hổ trợ tối thiểu.

Mô hình chung này thích hợp cho tất cả loại mẫu được tìm thấy trong data stream: itemets. chuỗi con, cây con, và độ thị con. Lưu ý rằng *Patt* xem toàn bộ dữ liệu trong $T_{i,j}$ để liệt kê các mẫu con. Tuy nhiên, trong đa số các công trình nghiên cứu, chúng ta xem mẫu con chỉ bên trong mỗi đối dữ liệu riêng lẻ, $Patt(T_{i,j}) = \cup\, Patt(T_a) \forall T_a \in T_{i,j}$.

Itemset ID	Contents
1	A,B,D,E
2	B,C
3	A,B,C,D,E
4	B,C,D,E
5	A,C,E

Bàng 4-1: Ví dụ của itemset stream [4]

Hãy xem cách mô hình này phù hợp như thế nào đối với bài toán khai thác itemset phổ biến. Trong ví dụ ở bảng 9.1, mỗi đối tượng dữ liệu trong stream là một itemset. Các mẫu đang tìm là các itemsets phổ biến đang nằm bên trong mỗi đối tượng dữ liệu. Ví dụ, T_1 = *{A,B,D,E}* và *Patt(T_1)* = *{(A,B),(A,D),(A,E),(B,D),(B,E),(D,E),(A,B,D),(A,B,E),(A,D,E),(B,D,E),(A,BD,E)}*, bỏ qua các mẫu chỉ có một item. Nếu độ hổ trợ tối thiểu $\sigma = 0.6$, itemset phải xảy ra ít nhất 3 lần trong 5 đối tượng trong stream. Các itemset phổ biến là $Patt_{\sigma=0.6}(T_{1,5}) = (A,E), (B,C), (B,D), (B,E), (D,E)$, và (B,D,E).

Data object	Contents
Item or Itemset	Item
Itemset	Subitemset
Sequence	Subsequence
Itemset	Sequence of items spanning a sequence of itemsets
Tree	Subtree
Graph	Subgraph or subtree

Bàng 4-2: Một số mẫu khác được khai thác từ stream [4]

Bảng 4-2, liệt kê danh sách các loại đối tượng dữ liệu và mẫu tương ứng có thể được khai thác trên stream.

Trong cùng một loại dữ liệu tùy cách ta xác định đối tượng dữ liệu mà mẫu được khai thác cũng khác nhau. Ví dụ, một tài liệu văn bản có thể được xem tập hợp các từ vựng, cho khai thác item phổ biến, hay một chuỗi các từ, cho khai thác cuỗi phổ biến.

Có rất nhiều mẫu phổ biến để khai thác trên data stream. Nhưng hai mẫu item và itemset là hai mẫu phổ biến cơ bản và cũng nhận được sự quan tâm của những nhà nguyên cứu nhất. Trong khóa luận, chúng tôi chỉ tập trung khai thác hai mẫu phổ biến này trên data stream. Để hiểu rõ hơn về khai thác item và itemset phổ biến, chúng tôi sẽ trình bày cụ thể hơn 2 bài toán này ở 2 chương sau.

4.4. Khai thác items phổ biến trên data stream

Ngày nay, bài toán khai thác items phổ biến trên data stream vẫn nhận được nhiều sự chú ý từ các nhà nguyên cứu. *Ứng dụng của khai thác items phổ biến là rất lớn. Nếu items là món hàng trong hơn hàng thì bài toán khai thác items phổ chính là bài toán tìm món hàng thường xuyên được mua. Nếu items là một từ khóa trong dòng trạng thái trong mạng xã hội thì bài toán khai thác items phổ biến chính là bài toán tìm từ khóa được mọi người quan tâm nhất*. Chính vì vậy, nhiều nhà nguyên cứu đã đề xuất ra nhiều thuật toán để khai thác một cách hiệu quả items phổ biến trên data stream.

Có nhiều cách tiếp cận cho việc tìm item phổ biến trên data stream. Nhìn chung có thể phân thành ba cách tiếp cận sau: Counter, Sketch và Sampling [2].

4.4.1. Hướng tiếp cận Counter

Một dạng rất cơ bản của hướng tiếp cận Counter-based là Majority trong việc tìm item có độ phổ biến cao nhất trên stream nếu nó tồn tại [18, 19]. Bằng việc tổng quát hóa Majority, Mirsa và Gries [20] đã đề xuất 1 phương thức để tìm những items mà xảy ra ít nhất N/k lần và nó được cải thiện trong tốc độ xử lý mỗi item bởi [21, 22].

Lossy counting [23] làm cùng 1 công việc nhưng nó đảm bảo thêm rằng không có item nào mà số đếm (count) của nó thấp hơn $N(1/k - \epsilon)$ được báo cáo. SpaceSaving [24] giảm yêu cầu về vùng nhớ không chỉ cho data chung chung mà còn cho phân phối Zipf.

4.4.2. Hướng tiếp cận Sketch

Tiếp cận này luôn luôn dựa trên nhiều hàm băm (hash function) để ánh xạ (map) item mới tới vào trong bảng băm (hash table). Điều này có thể được hiểu như là duy trì một danh sách các số đếm không phụ thuộc (independent counter) nơi mỗi số đếm được chia sẽ bởi 1 số ít items và cách chia sẽ được xác định thông qua hàm băm. Charikar và các đồng nghiệp đã đề xuất CountSketch mà tính các items xuất hiện ít nhất $N/(k+1)$ lần với xác xuất $1-\delta$ trong khi yêu cầu $O(k/\epsilon^2 \log N/\delta)$ không gian bộ nhớ. Yêu cầu bộ nhớ được cải thiện bởi CountMin [25]. GroupTest [26] được phát triển cho truy vấn hot items. Jin và các đồng nghiệp của ông cải tiến GroupTest trong không gian và thuật toán của họ đảm bảo số đếm tối thiểu (minimum count) của chúng được xuất [27].

4.4.3. Hướng tiếp cận Sampling

. Tiếp cận này gắn liền với quá trình lấy mẫu, và tất cả items đang xét được lấy mẫu từ stream; vậy chỉ 1 phần của stream được xét. Vitter [28] đề xuất 1 phương thức lấy mẫu độc lập từ 1 data stream bởi items có độ phổ biến cao hơn được lấy nhiều hơn so với items có độ phổ biến thấp. Cải thiện yêu cầu không gian cho lấy mẫu, Gibbons và Matias [18] đã đề xuất 1 phương thức được gọi là concise sampling mà thể hiện một cách hiệu quả items được lấy mẫu. Bằng thay đổi nhẹ phương thức, họ còn đề xuất counting sampling [23] để ước lượng độ phổ biến item chính xác hơn.

4.5. Khai thác itemsets phổ biến trên data stream

Ý tưởng khai thác itemsets phổ biến lần đầu tiên được đề xuất cho dữ liệu giao dịch bởi R.Agrawal và R.Srikant với thuật toán Apriori [12] cho dữ liệu tĩnh, từ itemset phổ biến tìm được các tác giả cũng đề hướng khai thác luật kết hợp từ itemsets

này. Từ lúc này, khai thác itemsets thu hút được nhiều sự chú ý từ các nhà nguyên cứu và nhiều công trình nguyên cứu ra đời.

Bài toán khai thác itemset có thể được định nghĩa như sau: Cho một bộ dữ liệu gồm có N giao dịch T = {T_1,T_2,…,T_N} và độ hổ trợ tới thiểu là σ. Mỗi giao dịch T_i chứa n_{T_i} items. Vậy số tập itemset có thể có từ T_i là $2^{n_{T_i}} - 1$.Nhiệm vụ của bài toán là khai thác itemset phổ biến là tìm tất cả các itemset được lấy mỗi transaction T mà số lần xuất hiện của nó lớn hơn σN.

Vì giới hạn chỉ được duyệt 1 lần dữ liệu khi xử lý trên stream, hầu hết tất cả thuật toán khai thác itemset phổ biến chỉ tìm được một kết quả xấp xỉ phổ biến. Những thuật toán rơi vào trong hai loại [4]: Thuật toán **false positive** đảm bảo kết quả bao gồm tất cả những itemset thật sự phổ biến, nhưng còn thêm số itemset khác không phổ biến. Thuật toán **false negative** đảm bảo kết quả trả về chắc chắn là itemset thật phổ biến, nhưng có thể còn những itemset phổ biến nhưng chưa được tìm thấy.

4.6. Thuật toán nền cho bài toán khai thác itemset phổ biến

Trong chương 7, chúng tôi sẽ trình thuật toán khai thác itemset phổ biến Skip LC-SS. Thuật toán này được xây dựng dựa trên hai thuật toán cơ sơ là Lossy Counting [23] và Space Saving [24]. Cả hai thuật toán đều rất nổi tiếng và là thuật toán cơ sở cho nhiều thuật toán khai thác item và itemset phổ biến khác.

4.6.1 Thuật toán Lossy Counting

Vào năm 2002, G.S. Manku và R.Motwani đã đề xuất ra thuật toán nổi tiếng Lossy Counting. LC là thuật toán xấp xỉ tìm được cả frequent items hay frequent itemset trên data stream chỉ với 1 lần duyệt. Cho trước minimal support σ, error parameter ϵ ($0 < \epsilon < \sigma$), và 1 data stream với N transactions, thuật toán LC trả về 1 tập các itemset mà có support $\geq (\sigma - \epsilon)N$. Như vậy, thuật toán LC là *no false negative* vì có thể trả về những itemset có support < σN.

Thuật toan Lossy Counting có thể được trình bày bằng mã giả như sau:

Thuật toán 1: LossyCounting

Input: Chiều dài k của mỗi bucket

1 $n \leftarrow 0, \Delta \leftarrow 0, T \leftarrow \theta$;
2 ***foreach*** i ***do***
3 $n \leftarrow n + 1$;
4 ***if*** $i \in T$ ***then*** $c_i \leftarrow c_i + 1$;
5 ***else***
6 $T \leftarrow T \cup \{i\}$;
7 $c_i \leftarrow \Delta + 1$;
8 ***endif***
9 ***endfor***
10 ***if*** $[N/k] \neq \Delta$ ***then***
11 $\Delta \leftarrow [N/k]$;
12 ***foreach*** $i \in T$ ***do***
13 ***if*** $c_j < \Delta$ ***then*** $T \leftarrow T \backslash \{j\}$;
14 ***endfor***
15 ***endif***

Thuật toán 1: thuật toán LossyCounting [23]

Trong thuật toán này, một entry cho itemset α được thể hiện dưới dạng tuple {α, f(α), Δ(t) }, nơi f(α) là số lần xuất hiện của a sau thời điểm t và Δ(t) là error count tại thời điểm t, với t là thời điểm gần nhất mà α được lưu

Thuật toán chia N transactions thành các buckets có cùng kích thước k (k = 1/ ϵ). Có [N/k] hay [N ϵ] buckets. Khi transaction thứ n tới (0 ≤ n ≤N), transaction này thuộc bucket [n/k]. Khi bucket thứ I đã đầy, thì bucket thứ I + 1 sẽ tiếp tục nhận transaction. Δ cho biết vị trí bucket hiện tại đang nhận transaction. Từ dòng 4 tới dòng thứ 8, mỗi lần 1 itemset α tới, ta cập nhật lại f(α) nếu α đã được lưu hay lưu mới α có tuple { α,1, Δ } nếu α chưa được lưu. Từ dòng thứ 9 tới 12, khi transaction n thuộc bucket thứ Δ + 1 tới, ta cập nhật lại Δ ⟵ [n ϵ] và bỏ đi những itemset j có $f(j) + \Delta_j <$ Δ

Có thể hiểu thuật toán này cách đơn giản như sau. Việc chia bucket thực chất là việc cắm các cột mốc để dần loại bỏ đi những phần tử itemset ít xuất hiện trong data

stream. Cột mốc của bucket là giai đoạn chuyển giữa hai bucket liên tiếp. Một itemset muốn được lưu trữ thì chúng phải xuất hiện ít nhất một lần tại mỗi bucket

Ý nghĩa của khái niệm error count Δ. Ở một số tài liệu sau này, người ta coi f(α) + Δ_α = c(α) tên là frequent count. Để ý sẽ thấy, thuật toán LC vẫn chạy tốt chỉ cần frequent count. Tác giả tách frequent count thành 2 thành là frequent f(α) và error count là Δ_α. Frequent f(α) là số lần xuất hiện của α từ lúc α bắt đầu được lưu trữ. Frequent f(α) không phản ánh được support của α vì có thể α đã được xuất hiện trước đó và đã bị xóa đi vì ít xuất hiện. Error count Δ_α là chặn trên của số lần α xuất hiện nhưng không được đếm. Ta có: f(α) + Δ_α = c(α) ≥ supp(α)

Một số tính chất của LC: f(α) < supp(α) < f(α) + ϵN và không gian bộ nhớ sử dụng được giới hạn $\frac{1}{\epsilon} log(\epsilon N)$.

Khuyết điểm của LC:

- Không giải quyết được vấn đề bùng nổ tổ hợp itemset. Khi 1 transaction tới, LC sẽ tổ hợp transaction thành 1 tập các itemset và tiến hành lưu trữ. Nếu bursty stream xảy ra, LC không quản lý vấn đề này.
- Thao tác xóa tại các mốc bucket tốn thời gian xử lý. Tại các mốc bucket, Lossy counting phải tiến hành duyệt lại và xóa đi những itemset không còn phổ biến nữa. Phép toán xóa tốn nhiều thời gian xử lý.

4.6.2 Thuật toán Space Saving

Vào năm 2005, A. Metawally, D. Agrawal và A. E. Abbadi đã đề xuất ra thuật toán nổi tiếng khác Space Saving (*SS*). SS chỉ khai thác frequent items trên data stream không như thuật toán LC.

Thuật toán 2: SpaceSaving

Input: Kích thước tối đa table k

1 $n \leftarrow 0,\ T \leftarrow \theta$;

2 ***foreach*** i ***do***

3 $n \leftarrow n + 1$;

4 ***if*** $i \in T$ ***then*** $c_i \leftarrow c_i + 1$;

5 ***elseif*** $|T| < k$ ***then***

```
6                   T ← T ∪ {i};
7                   c_i ← 1;
8           else
9                   j ← argmin_{j∈T} c_j;
10                  c_j ← c_j + 1;
11                  T ← T ∪ {i}\{j};
12          endif
13    endfor
```

Thuật toán 2: thuật toán SpaceSaving [24]

SS sử dụng dụng 1 table D có kích thước k cố định để lưu trữ entry. Nếu item tới và item đó đã nằm trong table, dòng 4, cập nhập frequent count. Nếu item tới và là mới thì sẽ có trường hợp. Nếu table D có kích thước |D| < k, dòng 5 tới 7, item được thêm vào table. Nếu table đã đầy|D| = k, dòng 9 tới 11, SS sẽ thay item có frequent count thấp nhất trong table D bằng item mới

Liệu phép toán thay thế item có frequent count thấp nhất trong table D có thể sẽ làm những item, mà nó mới được thay, bị thay ngay sau đó vì item mới được thêm đó có frequent count rất thấp sẽ là đối tượng cho phép toán thay thế, dòng 11. Vấn đề được giải quyết ở dòng 10, $c_i \leftarrow c_{min} + 1$, rõ ràng item mới được thay thế vẫn là mục tiêu của phép toán thay thế. Điều này dẫn tới item mới này đè lên item mới khác, dẫn tới tăng c_{min}. Tới một lúc nào đó c_{min} tăng vượt qua item cũ nằm trong table D, item cũ đó sẽ bị thay thế. Việc tăng c_{min} tạo cơ hội cho những item mới vượt qua những item cũ có frequent count cao nhưng đã khá lâu chưa được tăng

Cho ϵ là error parameter, tương tự như LC. Nếu $k \geq 1/\epsilon$ thì thuật toán SS sẽ output ra mỗi item e mà $suppI \geq \epsilon N$. Do đó cho mininal support σ ($0 < \epsilon < \sigma$), item mà có $suppI \geq \sigma N$ cũng sẽ là output của thuật toán SS (no false negative). Do đó cần ít nhất $O(1/\epsilon)$ không gian bộ nhớ

So sánh với LC. SS sử dụng phép toán thay thế thay vì phép toán xóa như LC, phép toán SS nhanh hơn. SS không tách frequent count thành frequent cộng với error count như LC.

4.7. Kết luận

Trong chương này, chúng ta đã trình bày cái khác niệm về mẫu phổ biến, vấn đề khai thác mẫu phổ biến trên data stream và ý nghĩa. Chúng ta cũng định nghĩa một cách hình thức về bài toán khai thác mẫu phổ biến dùng chung cho tất cả các mẫu như là: item, itemset, chuỗi, cây, đồ thị. Để chuẩn bị cho mục tiêu của khóa luận là khai thác hai mẫu phổ biến là item và itemset, hai bài toán khai thác này được trình bày chi tiết hơn.

CHƯƠNG 5: PHƯƠNG PHÁP KHAI THÁC FREQUENT ITEMS TRONG DATA STREAM

Trong chương này, chúng tôi sẽ trình bày thuật toán khai thác frequent item trên data stream là TwMinSwap. Thuật toán này được lựa chọn từ một hội nghị uy tín CIKM - International Conference on Information and Knowledge Management đạt rank A xuất bản vào năm 2014. Thuật toán TwMinSwap chỉ yêu cầu O(k) không gian bộ nhớ với k là top k items có độ phổ biến cao nhất. Cuối cùng, chúng tôi sẽ trình bày cách mà thuật toán TwMinSwap được triển khai vào hệ thống thời gian thực mà chúng tôi đang xây dựng như thế nào.

5.1. Giới thiệu

Khai thác recent frequent items là một bài toán con của khai thác frequent items, trong đó ta tập trung tìm ra những frequent items mới xuất hiện và dần loại bỏ đi những frequent items đã cũ. Với ngữ cảnh ứng dụng, bài toán này là phổ biến. Ví dụ, khi ta theo dõi những từ khóa từ SNS, thật là quan trọng để biết từ khóa nào là xu hướng mới và từ khóa nào là xu hướng cũ. Nhiều nguyên cứu tìm kiếm những recent frequent items được nguyên cứu. Tuy nhiên, chúng bị giới hạn về độ chính xác, thời gian chạy, và sử dụng bộ nhớ.

Vào năm 2014, tại hội nghị CIKM, Y.Lim, J.Choi và U.Kang đã đề xuất một phương pháp nhanh, chính xác, và sử dụng hiệu quả bộ nhớ cho việc theo dõi recent frequent items đó là ***thuật toán TwMinSwap*** [2]. Thuật toán TwMinSwap là phiên bản tất định của thuật toán lấy mẫu TwSample mà có một sự đảm bảo tốt về mặt lý thuyết. Thuật toán TwMinSwap hoàn thiện TwSample cả về tốc độ xử lý, độ chính xác và sử dụng bộ nhớ. Cả hai chỉ yêu cầu O(k) không gian bộ nhớ và không cần biết trước về stream như là chiều của nó hay số lượng items phân biệt trên stream. Đặc biệt, thuật toán chỉ yêu cầu hai tham số là k và α, và sự đơn giản này không chỉ cho phép tiết kiệm vùng nhớ mà còn giảm thời gian xử lý mỗi items (per-item processing time). Thí nghiệm chỉ ra, thuật toán TwMinSwap chạy tốt hơn những thuật toán cạnh

tranh khác ở những khía cạnh precision & recall, error trong time-weighted count, sử dụng bộ nhớ và thời gian xử lý trong Bảng 5-1.

	[Recommended]		**Competitors**	
	TwMinSwap [2]	TwSample [2]	TwFreq [29]	TwHCount [30]
Precision & Recall	**Highest**	Low	~~Lowest~~	High
Error in TwCount	**Lowest**	Medium	~~Highest~~	**Lowest**
Memory Usage	**Samllest**	~~Largest~~	Small	~~Largest~~
Time	Fast	~~Slowest~~	Medium	**Fastest**

Bàng 5-1: Bảng so sánh giữa TwMinSwap với các thuật toán khác [2]

5.2. Ký hiệu được sử dụng trong thuật toán khai thác items

Bảng bên dưới liệt kê tập các ký hiệu và mô tả, những ký hiệu này được dùng xuyên suốt thuật toán sẽ được trình bày bên dưới.

Ký hiệu	**Mô tả**
N	Chiều dài của stream
n	Số lượng items phân biệt
K	Số lượng (time-weighted) frequent items
α	Time decay factor
u, v	Định danh của item
c,c_u	Số đếm (của u)
T_u	Tập các mốc thời gian mà u xảy ra
W(u)	Time-weighted count của u
P(u)	Penalized time-weighted count của u
t,t_i / t_{cur}	thời điểm / thời điểm hiện tại
K	Tập các (time-weighted) frequent items đã khai thác được

λ	Penalty term cho P(u)
σ	Tỉ lệ tăng của λ

Bàng 5-2: Các ký hiệu được dùng trong chương này [2]

5.3. Thuật toán đề xuất

Tác giả đã đề xuất 1 phương pháp để tìm time-weighted top-k items một cách hiệu quả từ data stream là thuật toán TwMinSwap. Thuật toán TwMinSwap chỉ yêu cầu O(k) không gian bộ nhớ với k là số time-weighted top-k items mà ta muốn tìm. Thuật toán TwMinSwap đạt được bằng cách derandomization từ thuật toán lấy mẫu TwSample. Để phát triển thuật toán TwMinSwap, ta sẽ trình bày trước về thuật toán TwSample. Thuật toán TwSample có sự đảm bảo tốt về mặt lý thuyết. Tuy nhiên, đặc điểm xác xuất của TwSample yêu cầu nhiều session lấy mẫu không phụ thuộc để đạt được độ chính xác cao, dẫn tới không gian sử dụng bộ nhớ lớn và thời gian chạy chậm. Mặt khác, TwMinSwap đạt được độ chính xác cao với thời gian chạy nhanh trong khi chỉ sử dụng session đơn.

Định nghĩa 1: (*Time-Weighted Count*). Cho *u* là 1 item xảy ra tại thời điểm $t_1,t_2,\ldots,t_c$. *Time-weighted count* của item *u* được định nghĩa bởi công thức sau và :

$$W(u) = \sum_{i=1}^{c} \alpha^{t_{cur} - t_i}$$

- a là decay paremeter
- t_{cur} là thời điểm hiện tại

5.3.1. Thuật toán ngẫu nhiên TwSample

Để phát triển 1 thuật toán lấy mẫu, đầu tiên ta định nghĩa penalized time-weighted count như bên dưới.

Định nghĩa 2: (Penalized Time-Weighted Count). Cho u là 1 item xảy ra trong data stream và T_u là tập các thời điểm mà u xảy ra. Penalized time-weighted count của u được định nghĩa bởi

$$P(u) = \sum_{t \in T_u} \alpha^{t_{cur} - t_i + \lambda - 1}$$

- a là tham số decay
- t_{cur} là thời điểm hiện tại
- λ là penalty term cho items vừa tới

Thuật toán 3: TwSample: Randomized Time-Weighted Counting

Input: Stream S, k số items được theo dõi, tham số decay α, tỉ lệ tăng σ của penalty term.

$\lambda \leftarrow 1$;
$K \leftarrow \emptyset$;
foreach *item mới u từ S* **do**
    $Domsampling(\alpha)$;
    if $bernoulli(\alpha^{\lambda-1}) = 1$ **then**
        if $u \in K$ **then** $c_u \leftarrow c_u + 1$;
        else
            $K \leftarrow K \cup \{u\}$;
            $c_u \leftarrow 1$;
        endif
        while $|K| < k$ **do**
            $\lambda \leftarrow \lambda + \sigma$;
            $Downsampling(\alpha^{\sigma})$;
        endwhile
    endif
endfor
Thủ tục $Dowmsapling(\theta)$
foreach $v \in K$ *với số đếm* c_v **do**
    $c_v \leftarrow binomial(c_v, \theta)$;
    if $c_v = 0$ **then**
        $K \leftarrow K \backslash \{v\}$;
    endif
endfor

Thuật toán 3: thuật toán TwSample[14]

Thuật toán TwSample là thuật toán dựa trên lấy mẫu (sampling based algorithm). Một chút về thuật toán dựa trên lấy mẫu, thuật toán sẽ xem những items trên 1 stream như là những mẫu thử (sample). Khi 1 item tới, thuật toán sẽ lấy mẫu (sample) item đó theo xác xuất. Nếu lấy mẫu thành công, item được ghi nhận là đã xuất hiện tại thời điểm đó. Nếu thật bại, bỏ luôn item đó và thuật toán không nhớ gì về sự xuất hiện của nó. Những item có độ phổ biến càng cao thì càng dễ được lấy mẫu và ngược lại. Cho một chuỗi các items cho đến thời điểm t_{cur} là $u_1,\ldots,u_{t_{cur}}$ và $0 < \alpha \leq 1$ và$\lambda \geq 1$, thuật toán TwSample lấy mẫu mỗi item u_t với xác xuất $\alpha^{t_{cur}-t+\lambda-1}$. Khi λ càng tăng thì xác xuất lấy mẫu thành công càng giảm. Để duy trì việc theo dõi top-k frequent items chỉ với O(k) không gian bộ nhớ, khi số lượng item đang theo dõi vượt quá k thì thuật toán TwSample buộc phải loại bỏ đi những item ít xuất hiện nhất. Để làm được điều này, thuật toán TwSample chạy 1 thủ tục được gọi là downsampling. Thủ tục downsampling sẽ tiến hạnh lấy mẫu lại những items đã theo dõi dựa theo counter của item đó theo xác xuất α^{σ} và sẽ loại bỏ đi những item không xuất hiện lần nào tại lần lấy mẫu lại này.

Một cách chi tiết, Thuật toàn TwSample yêu cầu 3 tham số: k là số items tối đa được theo dõi, time-weighting factor α, và độ tăng σ cho penalty term λ. Còn có 3 loại thông tin được cập nhật tăng dần: penalty term λ, tập K của các items được theo dõi, và counter c_v được liên kết với mỗi v ∈ K. Khởi tạo, $\lambda = 1$ và K = $\emptyset$. Khi một item mới u tới, u được lấy mẫu với xác xuất $\alpha^{\lambda-1}$. Trong khi u được lấy mẫu thành công, nếu u ∈ k, counter c_u được tăng thêm 1, ngược lại u được thêm vào tập K với counter c_u =1. Sau khi lấy mẫu, Nếu |K| > k, λ được tăng 1 lượng σ và thực hiện downsampling. Cụ thể cho downsampling, mỗi $v \in K$, c_v được cập nhật bởi 1 con số được trả về từ hàm xác xuất $binomial(c_v, \alpha^{\sigma})$. Nếu c_v trờ thành 0, v được loại bỏ khỏi K. Quá trình này được lập lại cho tới khi $|K| \leq k$. Cuối cùng, trước mỗi lần 1 item mới tới thì TwSample sẽ downsampling tất cả item trong K trước khi xử lý item mới đó.

Hiệu quả của σ: Thực chất, σ xác định tốc độ lấy mẫu cho việc loại bỏ những items đang theo dõi để mà số items trong K không vượt quá k. Khi σ nhỏ, tổng lượng

counter của các items giảm tại dòng 14, dẫn tới cần thời gian dài cho quá trình loại bỏ tại dòng 12-15. Độ chính xác có thể tăng vì không loại bỏ hơn yêu cầu. Mặt khác, khi σ lớn, quá trình loại bỏ kết thúc nhanh, mặt dù độ chính xác có thể giảm vì ta có thể loại bỏ nhiều hơn một item.

Lemma 1: Tại thời điểm t_{cur} với penalty term λ , mỗi item u xảy ra tại thời điểm $t <= t_{cur}$ được lấy mẫu với xác xuất

$$Pr[u \text{ được } lấy\ mẫu] = \alpha^{t_{cur}-t+\lambda\ -1}$$

Chứng minh: Cho u là item mới tại thời điểm t = 1 và t_{cur} = 1. U được lấy mẫu với xác xuất $\alpha^{t_{cur}-t+\lambda\ -1} = 1$.

Cho $t_{cur} \geq 1$. Giả sử lemma đúng cho thời điểm t_{cur}. Điều này tương đương với cho mỗi u tại thời điểm $t \leq t_{cur}$, u được lấy mẫu với khả năng $\alpha^{t_{cur}-t+\lambda-1}$. Ta chứng minh lemma đúng tại thời điểm $t_{next} = t_{cur} +1$. Ở dòng 4, tất cả mẫu thử được downsample với xác xuất α. Điều này có nghĩa rằng sau dòng 4, những mẫu thử còn lại được lấy với xác xuất $\alpha^{t_{cur}-t+1+\lambda-1}$ mà trùng với xác xuất lấy mẫu tại thời điểm $t_{next} = t_{cur}+1$ của lemma 1.

Tiếp theo, ta kiểm tra dòng 5 tới 11. Cho u là item mới tới tại thời điểm $t = t_{next}$. Rõ ràng, u được lấy mẫu với xác xuất $\alpha^{t_{cur}-t+\lambda-1} = \alpha^{\lambda-1}$ bởi dòng 5, không quan tâm u có được theo dõi hay không. Ta sẽ kiểm tra downsampling. Coi d là số vòng lặp của cậu lệnh while tại dòng 12. Sau đó, sau khi downsampling, mỗi mẫu thử được lấy mẫu lại với xác xuất $\alpha^{d\sigma}$, và như vậy mỗi mẫu thử được lấy với xác xuất $\alpha^{t_{cur}-t+1+\lambda\ +d\sigma\ -1}$ mà trùng với xác xuất lấy mẫu tại thời điểm $t_{next} = t_{cur}+1$ của lemma 1 với $\lambda = \lambda + d\sigma$.

Lưu ý rằng khả năng lấy mẫu không tăng theo thời gian vì λ chỉ tăng chứ không giảm. Do đó, một item không được lấy mẫu tại thời điểm nào đó sẽ không được lấy mẫu trong tương lai. □

Hệ luận bên dưới có được trực tiếp từ Lemma 1:

Hệ luận 1: Tại bất kì thời t_{cur} nào trong TwSample, kì vọng của counter c_u của item u được theo dõi $u \in K$ là

$$E[c_u] = P(u) = \sum_{t \in T_u} \alpha^{t_{cur}-t+\lambda-1}$$

Với T_u là tập những thời điểm mà u xảy ra.

Lemma 2: Tại bất kì thời điểm nào, xác xuất p_u với u là item được theo dõi thỏa mãn bất đẳng thức bên dưới:

$$P_u >= 1- \exp(-P(u)/2)$$

Chứng minh: Lưu ý rằng $p_u = Pr[c_u >0]$ vì cu = $\sum_{i=1}^{|T_u|} c_u(i)$ là một biến ngẫu nhiên với $c_u(i)$ xác định sự xuất hiện thứ I của u có được lấy mẫu không. Áp dụng Chernoff bound, ta đạt được bất đẳng thức bên dưới:

$$P_u = Pr[c_u > 0] = 1 – Pr[c_u = 0] \geq 1 – \exp(-E[c_u]/2)$$

Vì $E[c_u] = P(u)$ bởi hệ luận 1, Lemma được chứng minh. □

Lemma 2 phát biểu rằng 1 item là ít quan trọng hơn, khả năng nó không được theo dõi tăng theo hàm số mũ.

Mặc dù TwSample là đơn giản và cung cấp một sự đảm bảo về lý thuyết cho kết quả của nó, nhưng hiệu năng của nó bị suy giảm vì tính chất của xác xuất. Đầu tiên, thời gian chạy trở nên chậm vì số vòng lặp cho downsampling với k tăng không cố định và thời gian cho việc lấy được biến ngẫu nhiên từ hàm binomial(c, θ) phụ thuộc vào c. Thứ hai, top k-items tìm được và counter c tương ứng có thể không chính xác vì λ lớn không thể kiểm soát. Sự không chính xác này có thể được giải quyết bằng duy trì s sessions độc lập mà mỗi session theo dõi k items, nhưng nó dẫn tới tốn nhiều không gian bộ nhớ và thời gian chạy. Trong mục tiếp theo, biến thể tất định của TwSample được đề xuất mà nó là nhanh và yêu cầu 1 session đơn của kích thước k.

5.3.2. Thuật toán tất định TwMinSwap

Thuật toán TwMinSwap được đề xuất như là một phiên bản tất định (derterministic) của thuật toán TwSample. Thuật toán TwMinSwap khai thác 1 cách hiệu quả top-k time-weighted frequent items.

Thuật toán **TwMinSwap** được trình bày với mã giả như sau:

Thuật toán 4: TwMinSwap: Deterministic Time-Weighted Counting

Input: Stream S, k số items được theo dõi, tham số decay α

$K \leftarrow \emptyset$;
$t_{cur} \leftarrow 0$;
foreach *item mới u từ S* **do**
    $t_{cur} \leftarrow t_{cur} + 1$;
    foreach $v \in K$ **do**
        $c_v \leftarrow c_v \times \alpha$;
    endfor
    if $u \in K$ **then**
        $c_u \leftarrow c_u + 1$;
    else if $|K| < k$ **then**
        $K \leftarrow K \cup \{u\}$;
        $c_u \leftarrow 1$;
    else
        $v^* \leftarrow argmin_{v \in K} c_v$;
        if $c_{v^*} < 1$ **then**
            $K \leftarrow K\{v^*\} \cup \{u\}$;
            $c_u \leftarrow 1$;
        endif
    endif
endfor

Thuật toán 4: thuật toán TwMinSwap [14]

Ý tưởng chính là việc lưu trữ số kì vọng của mẫu thử cho mỗi item một cách trực tiếp thay vì áp dụng quá trình ngẫu nhiên. Cụ thể, cho 1 item u xảy ra tại thời điểm $t \leq t_{cur}$, thay vì tăng c_u bởi 1 với xác xuất $\alpha^{t_{cur}-t}$, ta tăng c_u bởi $\alpha^{t_{cur}-t}$ với xác xuất 1. Sau đó, downsampling với tốc độ θ trở thành cv = cv θ, cho mỗi item được theo dõi v ∈ K. Tuy nhiên, đối với tình huống tất định này, ta gặp phải một vấn đề khi tất cả counters trở lên đầy. Cụ thể, vì số kì vọng cho item xuất hiện ít nhất một lần trên data stream không bao giờ trờ về 0, nó cần được cắt đi để loại bỏ item đang theo dõi mà không trọng bằng một item mới. Thuật toán TwMinSwap đề xuất một heuristic

đơn giản cho việt cắt đi mà không yêu cầu thêm không gian bộ nhớ, dẫn tới sử dụng bộ nhớ thấp khi so sánh với các cách tiếp cận khác [5, 26, 27]. Lưu ý rằng phương thức cắt ở đây tương ứng với việc giảm tốc độ lấy mẫu bởi việc tăng λ trong TwSample.

Chi tiết của heuristic cho việc cắt tỉa được bàn luận bên dưới. Cho K là một tập items hiện tại được theo dõi với |K| = k, và u là item vừa tới từ data stream. Phương thức cắt tỉa đầu tiên tìm một item v* ∈ K có time-weighted count tối thiểu $c_{v^*} = min_{c \in K} c^*$. Nếu $c_{v^*} < 1$, ta bỏ c_{v^*} và bắt đầu theo dõi u với count khởi tạo là 1; ngược lại, u được bỏ qua. Sự trao đổi giữa u và v^* là hiểu được vì 1 là count của u khi mới vừa được khởi tạo và nếu $c_{v^*} \geq 1$ thì lúc này nếu u nằm trong K thì sẽ bị xóa vì c_u là count thấp nhất trong K dẫn tới việc theo dõi u là không cần thiết; điều này giải thích tại sao u lại bị bỏ qua khi $c_{v^*} \geq 1$ và việc tráo đổi chỉ được thực hiện khi $c_{v^*} < 1$.

Điểm mạnh của TwMinSwap tổng kết lại bên dưới. Đầu tiên, bộ nhớ mà nó sự dụng là nhỏ. Cho mỗi item, chỉ định danh (id) và time-weighted count của nó được duy trì. Thứ hai, TwMinSwap yêu cầu tham số tối thiểu: k và α. Điều này đặc biệt thuận lợi cho việc giảm nổ lực điều chỉnh tham số trong thực tế. Thứ ba, trái với TwSample, TwMinSwap đảm bảo output k items miển là N ≥ k.

Thời gian xử lý mỗi item. Có hai phép toán tốn thời gian chạy là: (1) tìm item với count tối thiểu, và (2) nhân α với tất cả counters của tất cả items trong K. Cả 2 phép toán điều tốn O(k) thời gian chạy. Vậy, TwMinSeap yêu cầu O(k) thời gian chạy cho mỗi vòng lặp.

5.3.3. Phân tích thuật toán TwMinSwap

Ta sẽ phân tích TwMinSwap về điều kiện khi nào item được theo dõi không bị loại khỏi K. Một cách chính xác hơn, Lemma 3 và 4 phát biểu rằng TwMinSwap sẽ không loại bỏ những items mà số lần xuất hiện của nó trên một ngưỡng nào đó cho cả trường data chung chung và trường hợp phân phố lũy thừa.

Lemma 3: Cho $0 < \alpha \leq 1$ là tham số time-decay của TwMinSwap. Bất kì item với c ≥ 1 sẽ không bị loại bỏ khỏi K nếu nó xuất hiện ít nhất một lần mỗi $1\text{-}\log_\alpha \gamma$ nơi $\gamma = \min\{1+\alpha, c\}$.

Chứng minh: Giả sử v ∈ K với count $c_v = c \geq 1$ tại mốc thời gian t và nó xuất hiện một lần mỗi d thời gian. Ta định nghĩa hàm bên dưới:

$$g(r+1) = (g(r)\alpha + 1)\alpha^{d-1}$$

- r là số lần mà v xuất hiện kể từ mốc thời gian t
- gI cho biết c_v tại lần thứ r mà v xuất hiện từ mốc thời gian t. Cụ thể, $g(1) = c\alpha^{d-1}$ và $gI = c_v$ sau rd-1 thời gian kể từ mốc thời gian t.

Ta sẽ chứng minh $g(r) \geq 1$ với $d \leq 1-\log_\alpha\gamma$ với mỗi r >= 1 nơi $\gamma = \min\{1+\alpha, c\}$.

Trường hợp r = 1, giả sử rằng $c \leq 1 + \alpha$, thì

$$g(1) = c\alpha^{d-1} \geq c\alpha^{log_\alpha c} = 1.$$

Giả sử rằng $c > 1 + \alpha$ thí

$$g(1) = c\alpha^{d-1} \geq (1+\alpha)\alpha^{d-1} \geq (1+\alpha)\alpha^{log_\alpha(1+\alpha)} = 1.$$

Trường hợp r > 1 , giả sử rằng g(r-1) ≥ 1 thì

$$g(r) = (g(r-1)\alpha + 1)\alpha^{d-1} \geq (\alpha+1)\alpha^{d-1}$$

$$\geq min\{1+\alpha, c\} * \alpha^{-log_\alpha min\{1+\alpha,c\}} = 1 \quad \square$$

Lemma 4: Cho n là số items và phân phối của items là phân lũy thừa cơ số 2. Bất kì item được theo dõi $i \leq \sqrt{1 - log_\alpha(1+\alpha)/1.7}$ với count $c_i \geq \alpha^{1-1.7i^2}$ được kì vọng là không bị loại bỏ.

Chứng minh: Cho item I ∈ [1,n], khả năng mà nó xảy ra trên stream là

$$Pr[I \text{ xảy ra}] = i^{-2}/Z.$$

Nơi Z là normalization constant. Do đó, giá trị kì vọng của item xuất hiện trước i là i^2Z.

Giả sử rằng $c_i \leq \alpha + 1$. Bất đẳng thức bên dưới đảm bảo kì vọng mà i sẽ không bị loại bỏ bởi Lemma 3 :

$$1 - log_\alpha c_i \geq i^2 Z \iff 1 - i^2 Z \geq log_\alpha c_i \iff \alpha^{1-i^2Z} \leq c_i$$

$$\iff \alpha^{1-i^2Z} \leq \alpha + 1 \iff i \leq \sqrt{\frac{1 - log_\alpha(1+\alpha)}{Z}}$$

Gỉa sử $c_i > \alpha + 1$. Bất đẳng thức bên dưới đảm bảo kì vọng mà I sẽ không bị loại bỏ bởi Lemma 3:

$1 - log_{\alpha}(1+\alpha) \geq i^2 Z \Leftrightarrow i \leq \sqrt{\frac{1-log_{\alpha}(1+\alpha)}{Z}}$

Cuối cùng, $Z \leq \sum_{j=1}^{\infty} j^{-2} = \frac{\pi^2}{6} \leq 1.7$, chứng minh đã hoàn thành.

5.4. Thí nghiệm

Ta trình bày kết quả thí nghiệm cho việc thể hiển hiệu năng của TwMinSwap được đề xuất với data stream phát sinh là synthetic data stream. Đặc biệt, ta muốn trả lời những câu hỏi bên dưới:

Q1: Có bao nhiêu top-k time weighted frequent items mà ta có thể khám phá?

Q2:Ta ước lượng time-weighted count của những items khám phá được chính xác như thế nào?

Q3: TwMinSwap chạy nhanh như thế nào?

5.4.1. Cài đặt thuật toán

Ta xem xét 2 loại data stream được phát sinh từ luật phân phối lũy thừa để minh họa bursty item xảy ra nơi N là chiều dài của stream và n là số items phân biệt.

Phân phối tĩnh: N items được phát sinh dựa theo luật phân phối lũy thừa bên dưới

$$Pr[I \text{ được phát sinh}] \propto i^{-\beta}, \text{ với } I \in [1,n]$$

Phân phối động: cho 0 <= r <= 1, rN items đầu tiên được phát sinh từ

$$Pr[I \text{ được phát sinh}] \propto i^{-\beta},$$

và (1-r)N items cuối được phát sinh từ

$$Pr[I \text{ được phát sinh}] \propto (n - I + 1)^{-\beta}, \text{ vơi } I \in [1,n]$$

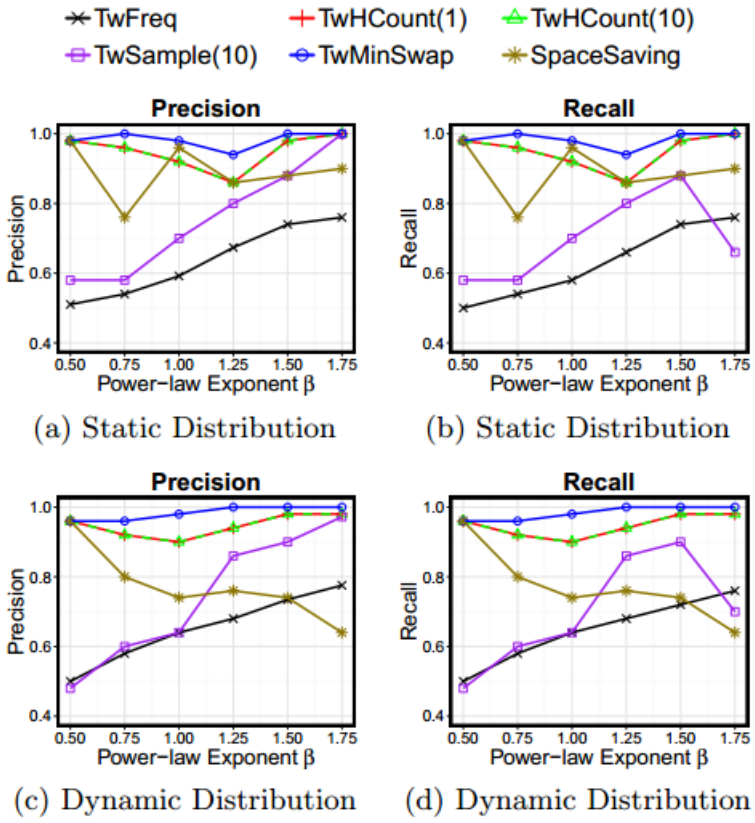
Trong thí nghiệm của ta, β được thay đổi trong {0.5,0.75,1,1.25,1.5,1.75}; N = 10^6, n = 10^4, r = 0.8 và k = 50 là cố định. Ở đây, khi β cao hơn, sự phổ biến của item bị lệch nhiều hơn. Lưu ý rằng mặc dù chiều dài stream N là cố định trong thí nghiệm này, thời gian xử lý mỗi item và không gian xử lý của thuật không phụ thuộc vào N.

Ta xem xét các thuật toán cạnh tranh khác, và trong thí nghiệm này, tất cả phương thức đều được cài đặt bằng java

- **TwFreg** [29]: thuật toán dựa trên counter để tìm time-weighted frequent items từ data stream.
- **TwHCount** [30]: thuật toán dựa trên sketch để tìm time-weighted frequent items từ data stream.
- **SpaceSaving** [24]: thuật toán dựa trên counter để tìm frequent items từ data stream. Vì thuật toán này không tìm time-weighted items, ta so sánh thuật toán này với những thuật toán khác chỉ với phương diện recall và precision.

Yêu cầu của tất cả thuật toán như bên dưới. TwMinSwap, TwFreg và SaceSaving yêu cầu O(k) không gian bộ nhớ. TwSample yêu cầu O(sk) với s là số session song song phục vụ cho mục đích lấy mẫu độc lập. TwHCount yêu cầu O(k+rm) nơi r là số lượng hash functions và m là phạm vi kích thước của hash function.

Tham số cho thuật toán TwSample là s = 10 và $\sigma = 0.0001$. Tham số cho thuật toán TwHcount, sử dụng tham số từ paper gốc [], và thiết lập kích thước table rm tới 1% và 10% của n.



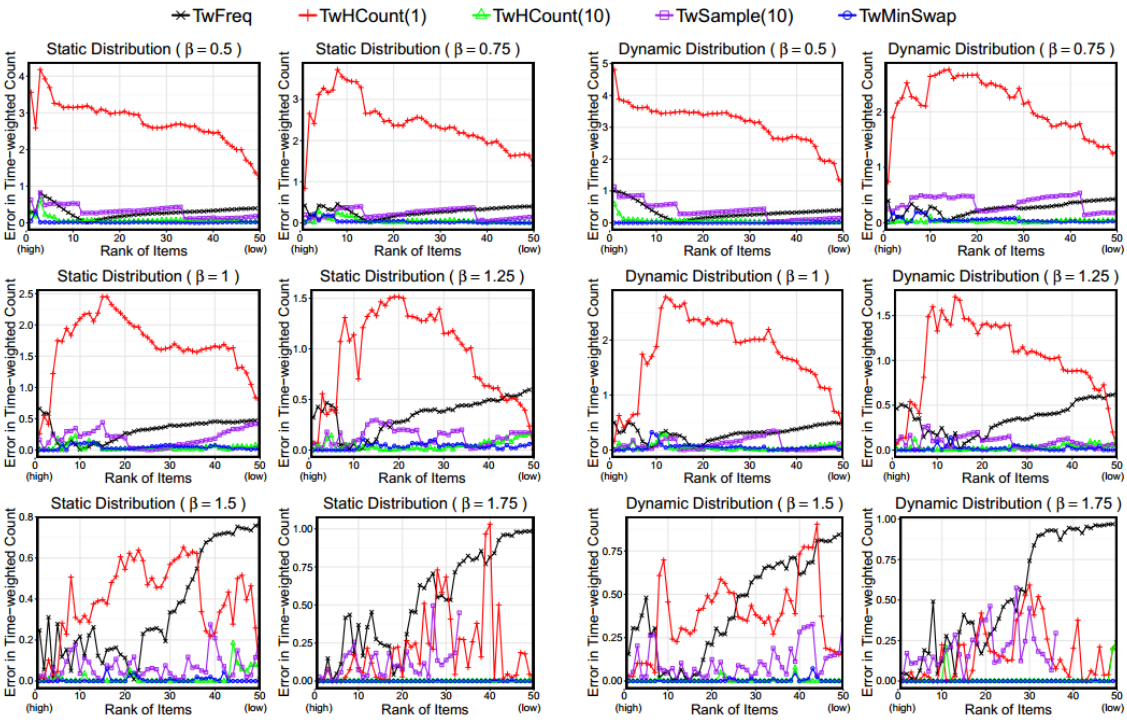
Hình 5-1: So sánh giữa các thuật toán về Precision và Recall [14]

5.4.2. Khám phá Top-k time-weighted frequent items

Hình 5-1 thể hiện sự chính xác của items được khám phá theo khía cạnh precision và recall. Một cách tổng thể, thuật toán TwMinSwap thực hiện tốt hơn các thuật toán khác bất chấp loại phân phối item và giá trị mũ β. Precision và recall của nó luôn gần bằng 1. TwSample(10) cái thiện được precision và recall khi β cao hơn nói chung. Lý do tại sao recall của TwSample(10) là thấp cho β = 1.75 là do lượng lớn mật độ xác xuất dày đặt được gán cho chỉ một ít items khi β lớn hơn, dẫn đến chỉ một số nhỏ $\bar{k} < k$ được khai thác. Số chính xác là 33 và 36 lần lượt cho phân phối tĩnh và phân phối động. Cho TwHCount, luôn luôn $\bar{k} = 50$, và cho TwFreg, luôn luôn $\bar{k} \geq 49$.

TwFreg trình bày pattern tương tự như TwSample(10) nhưng hiệu năng thấp hơn TwSample(10). TwHCount có kết quả cao ở precision và recall bất chấp kích thước của table nhưng vẫn thấp hơn TwMinSwap. Space Saving thực hiện khá tốt cho phân phối item tĩnh: cả hai precision và recall trung bình 0.9. Tuy nhiên, vì SpaceSaving

không xem xét time-weighting factor, cho phân phối item động hiệu năng của nó tụt giảm khi β lớn.



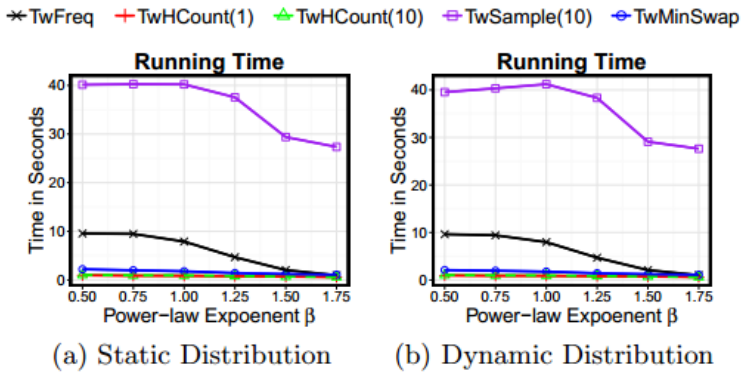
Hình 5-2: So sánh giữa các thuật toán về Time-weighted count [14]

5.4.3. Ước lượng Time-weighted Count

Hình 5-2 thể hiện ước lượng time-weighted count của top-k items được khám phá cho từng thuật toán. Đặc biệt, TwMinSwap ước lượng rất chính xác, mà được thể hiện bởi đường thẳng màu xanh dương (hầu như đè lên màu xanh lá cây).

Vì TwFreg cung cấp 1 lower bound và upper bound của true time-weighted count của top-k items được khám phá, ta chọn trung bình của 2 bound. Nói chung, khi rank của items dần thấp hơn, sự chính xác bị sụt giảm. Một lý do của ước lượng nghèo nàn của TwFreg cho item có rank thấp vì TwFreg được phát triển đầu tiên để tìm frequent items mà có time-weighted count lớn hơn một ngưỡng nào đó hơn tìm top-k items mặc dù nó duy trì cao nhất k items. TwSample(10) thể hiện hiệu năng tốt thứ ba, và ước lượng time-weighted count chính xác hơn khi β lớn tương đối. Hiệu năng trung bình tốt hơn TwFreg, nhưng TwSample(10) không bị tụt giảm tụt giảm đối với những items có rank thấp. Hiệu năng của TwHCount phụ thuộc nhiều vào kích thước của

hash table rm. Ước lượng kết quả TwHCount(1) có lỗi rất lớn trong khi TwHCount(10) là tương đương với TwMinSwap.



Hình 5-3: So sánh giữa các thuật toán về Running-time [14]

5.4.1. Thời gian thực thi

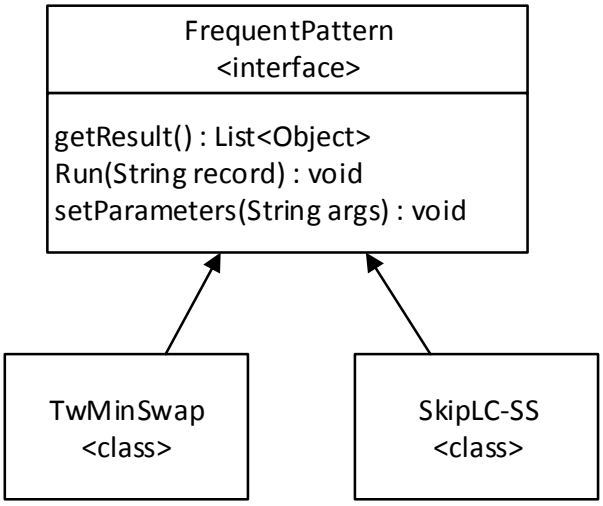
Hình 5-3 thể hiện thời gian chạy của thuật toán cho xử lý tất cả items trên data stream. Xu hướng chung là thời gian chạy giảm khi β tăng. Khi β tăng, xác xuất một item mới tới là item đang theo dõi tăng, dẫn tới không thường xuyên gọi tới bước loại bỏ như là DownSampling trong TwSample.

TwSample(10) thể hiện thời thời gian chạy thấp vì thực hiện 10 session lấy mẫu độc lập. Mặc dù TwFreg có cùng thời gian xử lý mỗi item như TwMinSwap trong khái niệm Big-O, TwFreg gọi nhiều phép tính cho bước loại bỏ hơn TwMinSwap. TwFreg cập nhật nhiều biến hơn và duyệt items được theo dõi hai lần trong TwMinSwap chỉ duyệt chúng một lần. Kết quả đạt được cho thấy TwFreg thay đổi nhiều trong thực tế hơn TwMinSwap.

Vì TwHCount có OI thời gian xử lý mỗi item, nó chạy nhanh nhất. Thật sự, thời gian chạy nhanh có được bởi duy trì một hash table với kích thước *rm* cho việc xấp xỉ time-weighted count, dẫn tới sử dụng bộ nhớ nhiều hơn TwMinSwap. Mặc dù thời gian chạy của TwHCount không phụ thuộc vào m, nhưng để đảm bảo error của ước lượng time-weighted count thấp, rm nên lớn như được thể hiện trong hình 5-6.

5.5. Vận dụng

Trong mục này, chúng tôi sẽ trình bày cách mà thuật toán TwMinSwap được tích hợp như thế nào trong hệ thống phân tích Data Stream mà chúng tôi đang xây dựng. *Hình 5-4* cho biết interface mà hệ thống cung cấp cho một thuật toán để tích hợp vào hệ thống.



Hình 5-4: Lớp interface tích hợp thuật toán

Theo interface FrequentPattern sẽ có function mà thuật toán cần implement cho lớp TwMinSwap:

- **setParameters**: là nơi thiết lập tham số cho thuật toán. Vì thuật toán yêu cầu 2 tham số là time decay $\alpha, 0 < \alpha \leq 1$ và $k, k > 0$ cho biết số lượng frequent item muốn khai thác, Input của function kiểu String có dạng "k, α".
- **Run**: là nơi chạy thuật toán xử lý cho mỗi item tới. Thuật toán được cài đặt trong function này. Input của function là item tới thường là idItem có kiểu String.
- **getResults**: là nơi lấy kết quả của thuật toán. Thuật toán trả về kiểu String có định dạng JSon trong đó có danh sách k phần tử là phần tử phổ biến kèm theo tham số cho biết độ phổ biến của nó.

Lớp TwMinSwap sẽ được tích hợp vào hệ thống và hoạt động như là thành phần lõi chuyên xử lý dữ liệu của hệ thống. Kết quả chuỗi Json được lưu vào Redis Cluster (Chương 3) và phân tán cho người dùng.

5.6. Kết luận

Trong chương này, chúng ta đã bàn luận về một thuật toán khai thác frequent items TwMinSwap trên data stream hiệu quả ở khía cạnh tốc độ xử lý, tính chính xác và sử dụng bộ nhớ. Thuật toán TwMinSwap được truyền cảm hứng bởi thuật toán xác xuất TwSample mà có một sự đảm bảo tốt về lý thuyết. Cà hai đều yêu cầu O(k) không gian bộ nhớ với k là số lượng items ta đang theo dõi. Dù thuật toán TwSample gặp khó khăn vì đặc điểm tự nhiên của xác xuất ảnh hưởng hiệu năng của thuật toán. Ngược lại, thuật toán TwMinSwap lại cho thấy hiệu năng tốt hơn hẳn không chỉ TwSample mà còn đối với các đối thủ cạnh tranh khác. Bên cạnh thuật toán TwMinSwap, chúng tôi đã trình bày cách tích hợp thuật toán vào hệ thống phân tích Data Stream.

CHƯƠNG 6: PHƯƠNG PHÁP KHAI THÁC FREQUENT ITEMSETS TRÊN DATA STREAM

Trong chương này, chúng tôi sẽ trình bày thuật toán khai thác itemset phổ biến trên Data Stream là Skip LC-SS. Thuật toán này được lựa chọn từ một tạp chí khoa học uy tín ACM SIGMOD xuất bản vào năm 2014. Thuật toán Skip LC-SS chỉ yêu cầu O(k) không gian bộ nhớ với k là hằng số và thời gian xử lý mỗi giao dịch là O(kL) dù có xảy ra bursty data stream. Cuối cùng, chúng tôi sẽ trình bày cách mà thuật toán Skip LC-SS được triển khai vào hệ thống thời gian thực mà chúng tôi đang xây dựng như thế nào.

6.1. Giới thiệu

Ngày nay, khai thác frequent itemsets trên data stream (FIM-DS) là lĩnh vực thu hút nhiều sự chú ý của nhà khoa học vì thách thức cũng như ứng dụng của nó. Đã có một mảng rộng lớn các ứng dụng sử dụng FIM-DS như là hệ thống giám sát, mạng lưới giao tiếp, và giao thông mạng cũng như những transaction on-line trong thị trường tài chính, công nghiệp bán lẻ và mạng lưới điện.

Bursty data stream: *hiện tượng chiều dài L của một hoặc nhiều transaction trong data stream đột ngột tăng rất cao, vượt quá kiểm soát*. Hiện tượng này có thể xảy ra trong thực tế, ví dụ như đối với dữ liệu thời tiết tại nơi xảy ra thảm họa lớn (như động đất hay lốc xoáy) hay giao thông mạng khi có tấn DOS xảy ra … thì lượng dữ liệu đổ về sẽ rất nhiều và chứa rất nhiều thông tin. *Bursty data stream* làm bùng nổ số tổ hợp của của itemset, dẫn tới tràn bộ nhớ đệm. Đây là một vấn đề vô cùng nghiêm trọng mà nhiều thuật toán FIM-DS được đề xuất trước đây không giải quyết được [31, 32, 23].

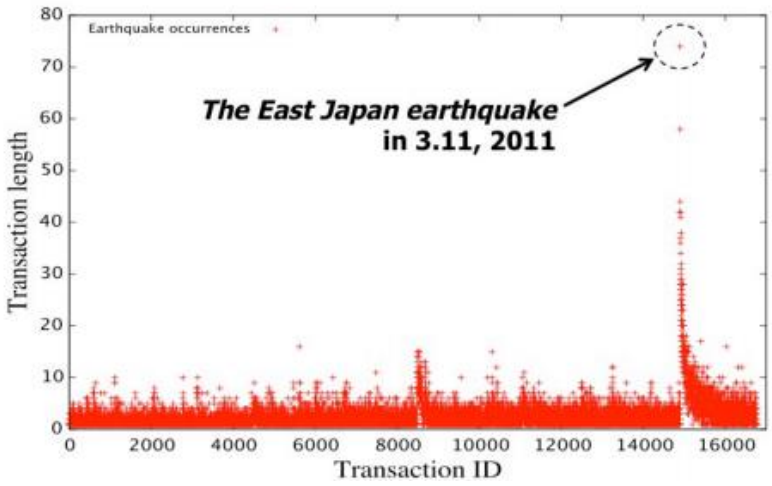
Trong FIM-DS, vấn đề thách thức là số lượng tổ hợp được của các entries (cụ thể là itemset) được tạo ra tại mỗi dòng giao dịch và được lưu trữ vào bộ nhớ. Những phương pháp xấp xỉ khác nhau cho FIM-DS đã được đề xuất. Tuy nhiên, những phương pháp đó yêu cầu cao nhất $O(2^L)$ không gian bộ nhớ cho chiều dài tối đa L của các giao dịch. Nếu một số giao dịch vì một sự kiện nào đó mà tăng đột ngột về chiều

dài như là **bursty data stream**, những phương pháp đó không thể làm việc vì giả định bộ nhớ tăng lên theo hàm số mũ khi L càng lớn.

Dựa trên những hạn chế của những thuật toán trước đây và nhu cầu thực tiễn, vào năm 2015 tại tạp chí SIGMOD, Y.Yamamoto, K.Iwanuma và S.Fukuda đã đề xuất ra 1 hướng tiếp cận mới hướng tài nguyên giải quyết được vấn đề ***bursty data stream****. Thuật toán được đề xuất tên là skip LC-SS* [3]*, giải quyết vấn đề tràn bộ nhớ bằng cách giới hạn vùng nhớ sẽ lưu trữ.* Thuật toán Skip LC-SS chỉ yêu cầu *O(k)* không gian bộ nhớ với k là hằng số và thời gian xử lý mỗi giao dịch là *O(kL)*. Thuật toán được đề xuất dựa trên ý tưởng từ 2 thuật toán nổi tiếng trước đó là Lossy counting *(LC)* và Space Saving *(SS)* đã được trình bày ở chương 4.

6.2. Một số nghiên cứu trước đây

Đã có nhiều nghiên cứu trước đó làm việc trên khai thác itemset bằng cách tiếp cận hướng tham số sử dụng 1 cấu trúc giữ liệu hiệu quả để lưu trữ liệu như là cây tiền tố hay cây hậu tố [31, 32, 23]. tuy nhiên tất cả nguyên cứu đều xác định trước ϵ và k bằng cách sử dụng các kiến thức biết trước về data stream, ví dụ như là sự phân phố dữ liệu, chiều dài tối đa của transactions. Tuy nhiên, dữ liệu thật thường gặp phải bursty data stream mà chiều dài của transaction là lớn hơn rất nhiều so với giả định của hệ thống, như là dữ liệu về địa lý. Ở Nhật Bản, dữ liệu về động đất rất được quan tâm.



Hình 6-1: Động đất xảy ra tại Nhật Bản [3]

Hình 6-1 cho thấy dòng transaction về động đất đã xảy ra tại Nhật Bản từ năm 1981 – 2013 (16768 transactions). Tại thời điểm 3.11.2011, busty stream xuất hiện, chiều dài của transaction đột ngột rất cao và ngoài tầm giả định bộ nhớ của các nguyên cứu trước đây. Khi trường hợp bursty stream xảy ra, các nguyên cứu trước đây sẽ gây ra một lỗi nghiêm trọng vì vượt qua giả định bộ nhớ.

Một nguyên cứu khác đã đề xuất một phương pháp giải quyết busty data stream [33]. Tuy nhiên, công việc đó tập trung vào một loại vấn đề khác của bursty data stream, cụ thể là tập trung vào khía cạnh giải quyết tỉ lệ bùng nổ transaction đổ tới tại mỗi đơn vị thời gian. Trái với vấn đề đang nguyên cứu là giải quyết bùng nổ tổ hợp được gây ra do chiều dài của transaction.

6.3. Các khái niệm và ký hiệu

Về itemset và *data stream* (*S*). Cho $I = \{x_1,x_2,x_3,\ldots,x_u\}$ là tập hợp các items. Một itemset là tập con không rỗng của I. Transaction data stream S là 1 chuỗi các transaction đang tới $\{T_1,T_2,\ldots,T_N\}$, với T_i là một itemset tới tại thời điểm i và N là số lượng transaction không biết trước.

Về tổ hợp itemset được tạo ra từ 1 transaction. mo(T_i) là tổ hợp itemset được tạo ra từ transaction T_i. ave là kích thước trung bình của mo, ave = $\frac{\sum_{i=1}^{N}\left(2^{|T_i|}-1\right)}{N}$. n_{mo} số lượng tổ hợp itemset có thể phát sinh của stream S, $n_{mo} = |(2^{T_1} \cup \ldots \cup 2^{T_N}) - \{\ \emptyset\ \}|$

Về lưu trữ trong thuật toán. Table D là nơi lưu trữ trong thuật toán. Đơn vị lưu trữ trong table là entry, mỗi entry đại diện cho 1 itemset được lưu. |D| là số lượng entry được lưu trong table. Frequent count c(α) là số lần xuất hiện xấp xỉ của itemset α trong stream S được lưu trữ trong table.

Về tham số trong thuật toán. Support của itemset α, kí hiệu supp(α), là số lần xuất hiện α trong stream S. Minimal Support σ ($0 \leq \sigma \leq 1$), item α được coi là phổ biến (frequent) nếu thỏa $supp(\alpha) \geq \sigma N$. Tác vụ FIM-DS là tìm tất cả frequent itemsets từ stream S. Error parameter ϵ ($0 < \epsilon < \sigma$) là tham số cho phép độ chính xác ở mức xấp xỉ của thuật toán. Thuật toán là *ϵ-accurate* nếu mỗi itemset kết quả α thỏa $c(\alpha) - \epsilon N$

$\leq$ supp(α) $\leq$ c(α). Thuật toán là *ϵ-honest* nếu c(α) $\leq$ ϵN cho mỗi α mà không lưu trong table D.

6.4. Hướng tham số và hướng tài nguyên

Trong mục này, ta sẽ làm rõ tại sao cách tiếp cận hướng tham số sẽ gặp phải vấn đề tràn bộ nhớ, đặc biệt khi xảy ra busrty data stream. Hầu như tất cả cách tiếp cận hướng tham số đều cần tham số lỗi *ϵ-accurate* và *ϵ-honest*. Hai tham số này có một ràng buộc trực tiếp tới kích thước table lưu trữ và chiều dài của một transaction.

Lemma 1: Cho D là table với kích thước k, S là stream với N transactions, và α là đối tượng muốn khai thác từ s (ở đây là itemset). Có tồn tại S và α thỏa supp(α) = $\frac{N*ave}{k+1}$ và α không được lưu trong table D.

Chứng minh: Cho stream S mà $n_{mo} = k + 1$ và mỗi mo(T_i) có cùng số đối tượng, ave. Tổng số lần xuất hiện của k+1 đối tượng là N*ave. Giả sử mỗi đối tượng O_i (1 $\leq$ I $\leq$ k+1) xảy ra độc lập trong N transactions. Vậy, support của mỗi đối tượng O_i là $\frac{N*ave}{k+1}$. Cho bất kì minimal support σ, chúng ta đều có thể tìm được giá trị cho ave thỏa $\sigma \leq \frac{ave}{k+1} < 1$. Do đó, mỗi đối tượng O_i đều là phổ biến theo σ vì supp(O_i) $\geq \sigma N$. Do table chỉ chứa k đối tượng nhưng có tới k+1 đối tượng là phổ biến, vậy còn 1 đối tượng là α mà supp(α) = $\frac{N*ave}{k+1}$ và α không được lưu trong table D.

Theorem 1: k ít nhất là min(n_{mo},$\frac{ave}{\epsilon} - 1$) cho thuật toán ϵ-accurate và ϵ-honest

Chứng minh: với $k = n_{mo}$ thì table D có thể chứa tất cả đối tượng có thể có của stream S, n_{mo} phải là chặn dưới của k. Bởi Lemma 1, có stream S và α mà supp(α) = $\frac{N*ave}{k+1}$ và α không được lưu trong table D. Ngoài ra, thuật toán là ϵ-honest khi α mà không được lưu trong table thỏa c(α) $\leq$ ϵN. Bên cạnh đó, thuật toán là ϵ-accurate nếu supp(α) $\leq$ c(α). Từ tất cả điều trên, ta có $\frac{N*ave}{k+1} \leq$ ϵN. Biến đổi bất đẳng thức, ta được k $\geq \frac{ave}{\epsilon} - 1$. Vậy, chặn dưới của k là min(n_{mo},$\frac{ave}{\epsilon} - 1$).

Theorem 1 cho thấy giới hạn nghiệm trọng của cách tiếp cận hướng tham số trong việc khai thác itemset. Cho một tham số lỗi ϵ, ta yêu cầu ít nhất min(n_{io}, $\frac{ave}{\epsilon}$-1) entries;

tuy nhiên, $ave = \frac{\Sigma\left(2^{|T_i|}-1\right)}{N}$ tăng theo hàm số mũ tương ứng với chiều dài T_i. Điều này chỉ ra rằng bất kì phương pháp hướng tham số nào có thể thất bại vì bỏ quên tài nguyên đặc biệt cho việc khai thác itemsets với bursty transaction.

6.5. Kết hợp thuật toán Lossy Counting và Space Saving

6.5.1. Thuật toán LC-SS

Thuật toán LC-SS là sự kết hợp của 2 thuật toán LC và SS, đặc biệt là SS. Thuật toán LC-SS lấy ý tưởng từ SS và áp dụng ý tưởng đó cho itemset. Thuật toán SS chỉ giành cho item nên tại 1 thời điểm chỉ cần thay thế 1 item khác có frequent count thấp nhất trong table bằng item mới, trong trường hợp table đã đầy. Điểm khác biệt của LC-SS so với SS ở chổ, tại 1 thời điểm không chỉ đơn thuần chỉ thay thế 1 item mà là 1 tập các itemset được lấy ra từ 1 transaction. Vấn đề của LC-SS phức tạp hơn nhiều so với thuật toán SS và cần thêm khái niệm error count Δ của thuật toán LC.

Tại sao khái niệm frequent count của thuật toán SS lại không thể áp dụng đối với thuật toán LC-SS cho itemset? Như đã nói ở trên, tại 1 thời điểm LC-SS thay thế 1 tập itemset chứ không phải là 1 item. Nếu áp dụng nguyên si thuật toán SS vào thì sai ngay vì frequent count tự tăng sau mỗi lần thay thế. Thuật toán SS sẽ coi 1 tập itemset là chuỗi tuần tự các item và frequent count sẽ tăng rất nhanh dù thời gian là không đổi. Lấy ý tưởng error count Δ từ thuật toán LC, thuật toán LC-SS tách rời khái niệm frequent count thành error count và frequent. Frequent là số đếm trực tiếp số lần xuất hiện của itemset mà table đang lưu trữ và error count là ước lượng số đếm của những itemset có thể đã được lưu và bị xóa đi vì có frequent count thấp nhất. Vấn đề được giải quyết, khi 1 tập itemset xuất hiện, ta chỉ cập nhật frequent còn error count tại thời điểm t chỉ giữ 1 giá trị.

Thuật toán 5: thuật toán baseline (LC-SS)

Input: Kích thước tối đa table k, minimal support σ, data stream S = (T_1, T_2, …, T_N).

Output: Một tập xấp xỉ các frequent itemsets từ S

1 $i \leftarrow 1$; ⊳ *i là thời điểm hiện tại*

2 *Khởi tạo* $D, \Delta(1)$ *và* θ ($D \leftarrow \emptyset, \Delta(1) = 0, \theta = 0$);

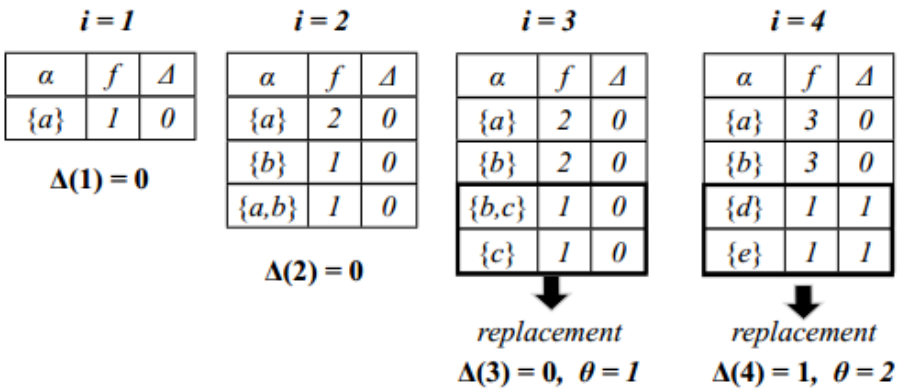
```
3    while i ≤ N do
4        đọc T_i;
5        foreach α ⊆ T_i mà α đã được theo dõi trong D do
6            f(α) + +;
7        endfor
8        foreach α ⊆ T_i mà α chưa được theo dõi trong D do
9            if |D| < k then
10               Thêm entry mới ⟨α, 1, Δ(i)⟩;
11           else
12               Tìm minimum entry ME = ⟨β, f(β), Δ_β⟩;
13               Thay thế ME với ⟨α, 1, Δ(i)⟩;
14               if θ < f(β) + Δ_β then
15                   θ ← f(β) + Δ_β;
16               ▻ θ được dùng để cập nhật Δ(i) sau này
17               endif
18           endif
19       endfor
20       if Δ(i) < θ then
21           Δ(i + 1) ← θ và θ ← 0;        ▻ Cập nhật Δ và θ
22       else
23           Δ(i + 1) ← Δ(i);
24       endif
25       i + +;                            ▻ Cập nhật thời điểm i
26   endwhile
27   for α ∈ D mà f(α) + Δ_α ≥ σN do
28       output α ▻ α là tập phổ biến ràng buộc σ và N
29   endfor
```

Thuật toán 5: thuật toán LC-SS [3]

Trong thuật toán LC-SS bên trên, giống với thuật toán LC, một entry cho itemset α được thể hiện dưới dạng tuple { α,f(α), Δ(t) }, nơi f(α) là số lần xuất hiện của a sau thời điểm t và Δ(t) là error count tại thời điểm t, với t là thời điểm gần nhất mà α được lưu.

Khi transaction T_i tới. Ta tìm tất cả tập con itemset của transaction T_i, dòng 4. Tiến hành cập nhật lại frequent của các tập con itemset mà chúng đã nằm trong table, dòng 5 tới 7. Những tập con itemset chưa nằm trong table sẽ thêm vào table trong 2 trường hợp sau. Nếu table chưa đầy |D| < k thì thêm mới, dòng 9 tới 10. Nếu table đã đầy |D| = k thì thay thế itemset có frequent count nhỏ nhất trong table, dòng 12 tới 13. Ở trường hợp hai, có những thao tác thay thế bị thừa. Vấn đề này sẽ được giải quyết với thuật toán cải tiến skip LC-SS. Θ là biến tạm lưu tạm thời error count hiện tại, dòng 14 và 15. Sau khi đã cập những tất cả itemset con của 1 transaction, ta mới bắt đầu cập nhật lại error count, dòng 19 tới 23. Dòng 26 và 27 là xử lý lấy tập frequent itemset từ table ra ngoài.

6.5.2. Ví dụ minh họa LC-SS



Hình 6-2: Quá trình cập nhật table trong stream S từ T_1 tới T_4 [3]

Cho stream S chứa 4 transactions {a}, {a,b}, {b,c} và {a,b,c,d,e}. Cho miminal support là 0.6 và kích thước tối đa của table là 4. *Hình 6-3* minh họa cách table được cập nhật theo quá trình xử lý từng transactions. Tại thời điểm i = 3, table đã đầy, và itemset có frequent count thấp nhất bị thay thế bởi 1 itemset ứng viên (candidate itemset). Δ(4) được cập nhật theo frequent count của itemset bị thay thế (Δ(4) = 1). Tại thời điểm t = 4, mỗi itemset được lưu trữ trong table đều là con của T_4, mỗi f được cộng thêm 1. Tiếp theo, 2 itemset ứng viên sẽ thay 2 entry thấp nhất. Có 27 tập ứng viên (2^{27} – 4 -1) từ T_4, phép toán thay thế được hiện 27 lần. Cuối cùng Δ(5) được cập nhật là 2. Sau T_4, kết quả trả về là {a} và {b} vì chúng có frequent count lớn σN = 0.6*4.

6.5.3. Tính chất của LC-SS

Lemma 2: Cho $\Delta(t)$ là error count tại thời điểm t. $\Delta(t)$ là chặn trên của support của những itemset trong khoảng thời gian [1,t) mà bị thay thế trước thời điểm t.

Chứng minh: Lemma 2 được chứng minh bằng phương pháp toán học quy nạp. Khi t = 1, $\Delta(1) = 0$. Vì không có itemset nào xuất hiện trước thời gian t = 1, lemma 2 đúng. Giả sử lemma 2 đúng với $t \leq i$, ta chứng minh lemma 2 đúng với t+1. Cho itemset β mà đã được thay thế tại thời điểm [1,t). Ta chia làm 2 trường hợp: Trường hợp 1: β được thay thế tại thời điểm t<i. Vì lemma 2 đã đúng với t <= i, support β tại thời điểm [1,t) nhỏ hơn hay bằng t. Trường hợp 2: β được thay thế tại thời điểm t = i. $f(\beta) + \Delta_\beta$ là chặn trên của support β trong khoảng thời gian [1,t+1). Tại dòng 20 của thuật toán LC-SS, $\Delta(i+1)$ được cập nhật tới $f(\beta) + \Delta_\beta$ nếu $f(\beta) + \Delta_\beta >= \Delta(i)$. Như vậy, $\Delta(i+1)$ là chặn trên của $\sup(\beta)$. Vậy lemma 2 được giữ.

Lemma 3: Cho S là 1 stream chứa N transactions. Với mỗi itemset α, nếu $\sup(\alpha) > \Delta(N + 1)$ thì α còn lại trong table.

Chứng minh: Giả sử có 1 itemset α thỏa $\sup(\alpha) > \Delta(N + 1)$ và α không nằm trong table. Vì α đã bị thay thế trong khoảng thời gian [1,N], do đó $\sup(\alpha) \leq \Delta(N + 1)$ theo *lemma 2*. Tuy nhiên, điều này mâu thuẫn với điều kiện của α, $\sup(\alpha) > \Delta(N + 1)$.

Sử dụng *Lemma 3*, ta tìm được theorem đảm bảo tính hiệu lực của output bởi thuật toán LC-SS.

Theorem 2: Cho S là 1 stream chứa N transactions và σ là minimal support. Nếu $\Delta(N + 1) < \sigma N$ thì mỗi frequent itemset ràng buộc theo S và σ được output bởi thuật toán LC-SS. Bên cạnh đó, mỗi output α thỏa $\sigma N - \Delta(N + 1) \leq \sup(\alpha)$.

Chứng minh: α là frequent itemset ràng buộc theo S và σ khi và chỉ khi $\sup(\alpha) \geq \sigma N$ (1). Từ (1) với giả thiết $\Delta(N + 1) < \sigma N$, α phải có support thỏa $\sup(\alpha) > \Delta(N + 1)$. Theo *lemma 3*, mỗi itemset mà có support cao hơn $\Delta(N + 1)$ thì vẫn còn trong table. Vậy, α luôn được lưu trong table và sẽ là output của dòng 27 của thuật toán LC-SS. Bên cạnh đó, mỗi output α thỏa $\sigma N \leq f(\alpha) + \Delta_\alpha$. Vì $f(\alpha) \leq \sup(\alpha)$ và $\Delta_\alpha \leq \Delta(N + 1)$, ta được $\sigma N \leq \sup(\alpha) + \Delta(N + 1)$.

Theorem 2 đảm bảo thuật toán LC-SS trả về kết quả no false-negative với điều kiện $\Delta(N + 1) < \sigma N$. Thỉnh thoảng $\Delta(N + 1)$ có thể vượt qua ngưỡng $\Delta(N + 1)$ nếu k là quá nhỏ. Ngay cả trong trường hợp này, thuật toán LC-SS vẫn đảm bảo output của thuật toán, như được mô ta bên dưới.

Theorem 3: Cho S là 1 stream chứa N transactions và support σ_Δ là $\frac{\Delta(N + 1)+1}{N}$. Mỗi frequent itemset ràng buộc theo S và σ_Δ có thể được output bởi thuật toán LC-SS.

Chứng minh: Mỗi itemset α mà có $sup(\alpha) \geq \Delta(N + 1) + 1$ đều là frequent itemset ràng buột theo S và σ_Δ vì $sup(\alpha) \geq \sigma_\Delta N$ với $\sigma_\Delta = \frac{\Delta(N + 1)+1}{N}$. Theo lemma 3, itemset α còn nằm trong table. Vậy, mỗi itemset α có thể được output bởi thuật toán LC-SS.

Theorem 3 xác định được chặn dưới của σ là σ_Δ, đảm bảo thuật toán trả về kết quả no false-negative. Biết được σ_Δ khá là giá trị vì người sử dụng có thể điều chỉnh σ trong quá trình chạy thuật toán để đạt được kết quả như ý mà không ảnh hưởng đến tính đảm bảo của thuật toán.

6.6. Thuật toán skip LC-SS

6.6.1. Hạn chế của thuật toán LC-SS

Thuật toán LC-SS yêu cầu $2^{|T_i|} - 1$ phép toán cập nhật và thay thế cho bất kì transaction T_i. Nếu bursty stream tới, như là T_4 trong ví dụ 2, thời gian truy cập table sẽ tăng theo hàm mũ. Như vậy, mặc dù thuật toán LC-SS có thể ngăn tràn bộ nhớ nhưng thời gian xử lý còn là một vấn đề lớn, đặc biệt là khi bursty stream tới. Do đó, thuật toán skip LC-SS được đề xuất để giải quyết vấn đề này. Ý tưởng là loại bỏ đi những phép toán thay thế thừa từ LC-SS để tăng tốc độ xử lý thuật toán.

Nhắc lại ví dụ. Tại thời điểm i = 4, bursty transaction cung cấp 27 tập ứng viên. Vì table đã đầy, mỗi tập ứng viên được thay thế với itemset có frequent count thấp nhất được lưu trong table. Thuật toán LC-SS đã thực hiện 27 phép toán thay thế. Kết quả cho thấy chỉ có 2 tập ứng viên {d} và {e} được lưu thay cho vị trí có frequent count thấp nhất, được lưu trước đó, trong table là {b,c} và {c}. Còn 25 tập ứng viên còn lại đã được lưu nhưng lại bị thay thế bởi tập ứng viên khác và kết quả là chúng

không nằm trong table. Vì thứ tự của các tập ứng viên là tùy ý nên thay vì {d} và {e}, kết quả của T_4 cũng có thể {a,c} và {b,e}. như vậy, chỉ cần thay thế 2 tập bất kì trong 27 tập ứng viên là đủ. Do đó, ta không cần phải thực hiện 27 phép toán thay thế trong ví dụ này.

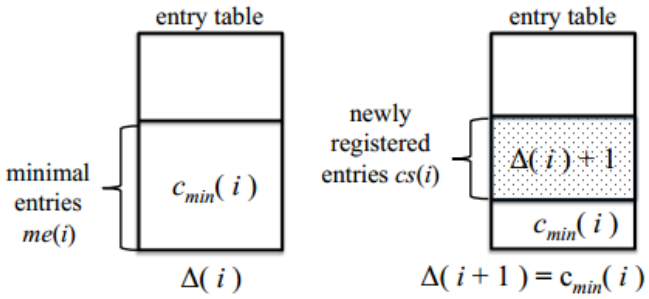
6.6.2. Thuật toán cải tiến skip LC-SS

Định nghĩa: Ta gọi entry mà lưu tập rỗng là empty entry. Với định nghĩa đó, ta giả sử rằng lúc khởi tạo entry table chứa k empty entry với kích thước tối đa của table là k. Do đó, nếu table chứa các empty entry thì minimal entry là các empty entry và có frequent count là 0. Thêm nữa, cs(i), me(i) và $c_{min}(i)$ lần lượt là số tập ứng viên, số minimal entry và frequent count của minimal entry.

Từ định nghĩa ở trên, phép toán thay thế của thuật toán LC-SS có thể được gom lại thành 3 trường hợp.

Trường hợp A: me(i) > cs(i). Sau khi thay thế cs(i) minimal entries với tập ứng viên cs(i), minimal entries trước đó vẫn tồn tại trong table. Lưu ý rằng mỗi entry mới có cùng frequent count là $\Delta(N) + 1$.

$$Cmin(i+1) = \min(c_{min}(i), \Delta(N) + 1),$$
$$\Delta(i+1) = cmin(i)$$

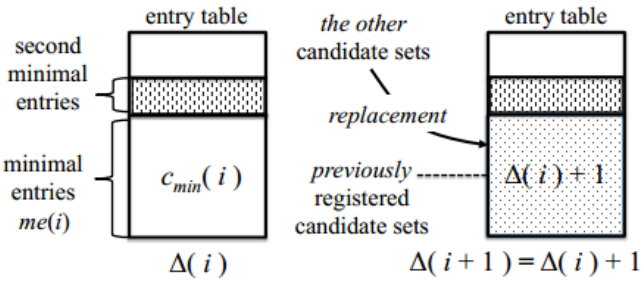


Hình 6-3: Trường hợp A: me(i) > cs(i) [3]

Trường hợp B: me(i) = cs(i). Tất cả minimal entries được thay bởi tập ứng viên. Frequent count của tập ứng viên là $\Delta(N) + 1$. $C_{min}(i+1)$ có thể là $\Delta(N) + 1$ hay là entry có frequent count thấp thứ 2 sau me(i), có frequent count tại thời điểm i là $c_{min}(i) + r$ với r (r>=1).

$$Cmin(i+1) = \min(\Delta(N) + 1, c_{min}(i) + r),$$

$$\Delta(i+1) = cmin(i)$$

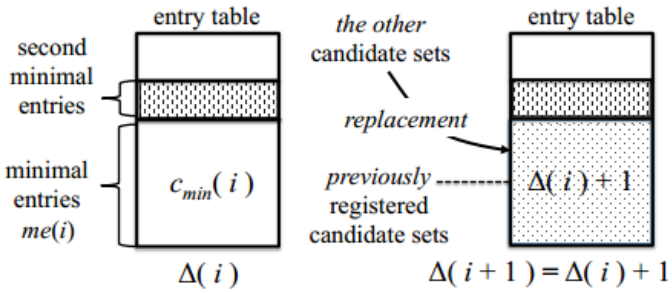


Hình 6-4: Trường hợp B: me(i) = cs(i) [3]

Trường hợp C: me(i) < cs(i). Ban đầu, mỗi minimal entry được thay thế với 1 số tập ứng viên. Sau đó, các tập ứng viên vừa được thêm vào table lại bị thay thế bởi những tập ứng viên khác. Θ đã được cập nhật thành Δ(N) + 1 do 1 tập ứng viên có frequent count Δ(N) + 1 đã bị xóa. Ở dòng , Δ(i+1) được cập nhật lại bằng Θ.

$$C_{min}(i+1) = \min(c_{min}(i) + r), \Delta(N) + 1)$$

$$\Delta(i+1) = \Delta(i) + 1$$



Hình 6-5: Trường hợp C: me(i) < cs(i) [3]

Từ những trường hợp trên, ta đạt được một mối quan hệ giữa Δ(t) và $c_{min}(t)$. Có thể dễ dàng chứng minh điều này bằng phương pháp quy nạp toán học.

Theorem 4 : $\Delta(t) \leq c_{min}(t) \leq \Delta(t)+1$ cho mỗi thời điểm t

Dựa vào theorem 4, ta xác định công thức tính cmin(t) và Δ(t) cho từng trường hợp

$$\begin{cases} Case\ A: me(i) > cs(i) & c_{min}(i+1) = c_{min}(i), \\ & \Delta(\mathrm{i}+1) = c_{min}(i). \\ Case\ B: me(i) = cs(i) & c_{min}(i+1) = \Delta(\mathrm{i})+1, \\ & \Delta(\mathrm{i}+1) = c_{min}(i). \\ Case\ C: me(i) < cs(i) & c_{min}(i+1) = \Delta(\mathrm{i})+1, \\ & \Delta(\mathrm{i}+1) = \Delta(i)+1. \end{cases} \quad (1)$$

Việc bỏ đi 1 số phép toán thay thế sẽ ảnh hưởng tới 1 số tham số khác của thuật toán như là Δ(t). Việc tính cmin(i+1) và Δ(t+1) là cần thiết để duy trì tính đùng đắn của thuật toán LC-SS.

Thuật toán skip LC-SS được đề xuất, là thuật toán LC-SS đã bỏ đi những phép toán thay thế thừa.

Thuật toán 6 : thuật toán Skip LC-SS

Input: Kích thước tối đa table k, minimal support σ, data stream S = (T_1, T_2, …, T_N).

Output: Một tập xấp xỉ các frequent itemsets từ S

1 $i \leftarrow 1$ ▻ *i là thời điểm hiện tại*

2 *D chứa k empty sets*, $\Delta(1) \leftarrow 0$;

3 $r(1) \leftarrow 0$; ▻ $r(i)$: *số entries thuộc* T_i

4 $c_{min}(1) \leftarrow 0, me(1) \leftarrow l, cs(1) \leftarrow 0$; ▻ *4 biến mới*

5 ***while*** $i \leq N$ ***do***

6 Đọc T_i;

7 ***foreach*** α ⊆ T_i mà α đã được theo dõi trong D **do**

8 $f(\alpha)++$ và $r(i)++$;

9 ***endfor***

10 Cập nhật $c_{min}(i)$ và $me(i)$ *bằng kiểm tra D*

11 *Cập nhật* $cs(i) \leftarrow 2^{|T_i|} - r(i) - 1$;

12 ***if*** $cs(i) < me(i)$ ***then***

13 *Lựa chọn tất cả tập ứng viên* ($cs(i)$) *từ* T_i;

14 ***else***

15 *Lựa chọn ngẫu nhiên* tập ứng viên (me(i)) từ T_i;

16 ***endif***

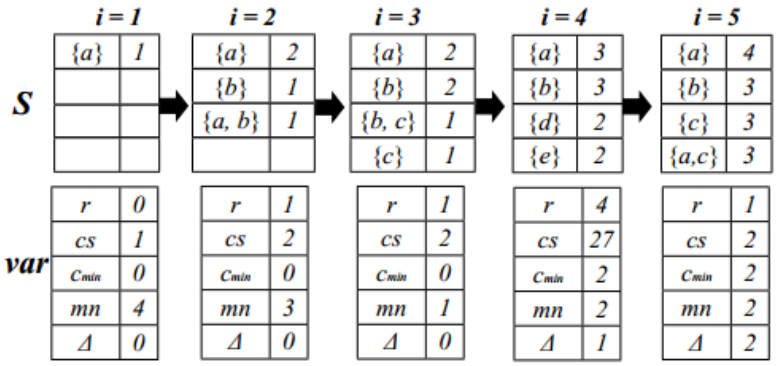
17 *Thay thế minimal entries bằng tập được lựa chọn trong D*

18	*mà có frequent count được gán là* $\Delta(i)+1$;
19	*Cập nhật* $c_{min}(i+1)$ *và* $\Delta(i+1)$ *bởi phương trình* (1);
20	$i++$; ▻ *cập nhật thời điểm i*
21	$r(i) \leftarrow 0$; ▻ *làm mới* $r(i)$
22	***endwhile***
23	***for*** α ∈ D mà f(α) + $\Delta_\alpha \geq \sigma N$ ***do***
24	*output* α ▻ α là tập phổ biến ràng buộc σ *và* N
25	***endfor***

Thuật toán 6: thuật toán Skip LC-SS [3]

Trong thuật toán skip LC-SS. Đầu tiên, thay vì một lúc duyệt tất cả itemset con của transaction T_i như LC-SS thì thuật toán chỉ cập nhật lại frequent của các itemset con mà chúng đã nằm trong table. r(i) cho biết số lượng itemset con của transaction T_i đã nằm trong table, dòng 7 và 8. Tại dòng 11, cs(i) = $2^{|T_i|} - r(i) - 1$. Tại dòng 12 tới dòng 16, thao tác lựa chọn tập itemset sẽ thay thế từ tập ứng viên theo cs(i) và me(i). Dòng 17 và 18, thay thế minimal entries bằng tập đựa lựa chọn với frequent count là Δ(N) + 1. Cuối cùng, dòng 19, cmin(i+1) và Δ(N+1) được tính bởi phương trình (1).

6.6.3. Ví dụ minh họa



Hình 6-6 : Cập nhật table từ T1 tới T5 [3]

Cho stream S chứa 5 transactions {a}, {a,b}, {b,c}, {a,b,c,d,e} và {a,c}. Cho miminal support là 0.6 và kích thước tối đa của table là 4. Hình minh họa cách table D và tập các thêm số bao gồm r,cs,cmin,me và Δ được cập nhật từ khoảng thời gian i = 1 tới 5 bởi thuật toán skip LC-SS. Tại thời điểm i = 4, mỗi entry trong table là con

của transaction Ti, ta có r(i) = 4. Cmin(4) và me(4) được cập nhật là 2 tại dòng số 10. Số tập ứng viên được tính tại dòng 11, $c(4) = 2^5 – 1 –r(4) = 27$. Vì cs(4) > me(4), ta cập nhật cmin(5) = $\Delta(4)+1 = 2$ và $\Delta(5) = \Delta(4) + 1$ tương ứng với phương trình (1). Tại thời điểm i = 5, chỉ có entry {α} là tập con của T_5, r(5) = 1. Tiếp, ta tính được $c(5) = 2^2\text{-}1\text{-}1 = 2$ và min(5) = me(5) = 2. Vì c(5) = me(5) nên min(6) = $\Delta(5) + 1$ và $\Delta(6) = \min(5) = 2$ theo phương trình (1). Output là tất cả entry trong table vì chúng đều thỏa frequent count lớn hơn hay bằng $\sigma N = 0.6*5 = 0.3$.

Thuật toán skip LC-SS là cải tiến của LC-SS ở phương diện hiệu năng nên vẫn giữ nguyên được tính đúng đắn mà thuật toán LC-SS đã có. Sau đâu là đánh giá về thời gian xử lý của thuật toán.

6.6.4. Tính chất của skip LC-SS

Theorem 6. Cho T là transaction với chiều dài L và k là kích thước của table. Thuật toán skip LC-SS xử lý T trong O(kL) bước.

Chứng minh : Trong thuật toán skip LC-SS, có 2 xử lý tốn thời gian nhất : Cập nhật table tại dòng 7-9 và lựa chọn me(i) tập ứng viên từ Ti tại dòng 12-16. Với xử lý cập nhật, thuật toán yêu cầu k*O(L) bước vì ít nhất O(L) lần kiểm tra transaction T với k entires được lưu trong table. Với xử lý lựa chọn tập ứng viên, thuật toán yêu cầu 2k*O(L) bước. Giả sử ta phát sinh 2k tập con của itemset từ transaction T. Nếu như table đã chứa k tập con ta vừa phát sinh thì vẫn còn k tập con còn lại cho tập ứng viên. Vì me(i) <= k, k tập ứng viên là đủ cho phép toán thay thế. Vậy, cả hai phép toán thay thế có thể được thực hiện trong O(kL) bước cho T.

6.7. Cải tiến thuật toán skip LC-SS

Vấn đề về bộ nhớ đã được giải quyết một cách thỏa đáng bởi thuật toán LC-SS. Thuật toán skip LC-SS được đề xuất cải tiến hiệu năng nhưng vẫn đảm bảo được những tính chất của thuật toán LC-SS. Sau đây, ta sẽ bàn luận về những hướng cải tiến cho thuật toán skip LC-SS về mặt hiệu năng

6.7.1. r-skip và t-skip

t-skip và *r-skip* cải tiến thuật toán skip LC-SS về mặt hiệu năng bằng cách loại bỏ thêm những phép toán thay thế hay cập nhật mà khi bỏ chúng có thể cải thiện hiệu

năng của thuật toán. *T-skip* và *r-skip* không còn duy trì được thuật toán LC-SS như skip LC-SS nhưng hiệu năng được cải thiện.

Trong trường hợp C của phương trình (1), error count Δ(i) phải tăng 1 dù phép toán thay thế có thể thực hiện được hay không. Nói cách khác, ta có thể bỏ tất cả phép toán thay trong trường hợp C. Điều này dễ dàng được hiện bằng việc kiểm tra me(i) > cs (i) trước dòng 17 của thuật toán skip LC-SS. Ta gọi thao tác trên là cải tiến ***r-skip***.

Đánh giá cải tiến *r-skip*. Theo Lemma 3, itemset α mà có sup(α) > Δ(N+1) thì α được lưu trong table. Lemma 3 không ràng buộc cho sup(α) = Δ(N+1). Vậy những itemset mới được thay thế trong trường hợp C có sup(α) = Δ(N+1) có nằm trong table hay không, điều đó không ảnh hưởng tới Lemma 3 và không ảnh hưởng đến kết quả của thuật toán. Vậy, cải tiến *r-kip* không không làm thay đổi kết quả của thuật toán skip LC-SS.

Lemma 4 : Cho T_i là 1 transaction tại thời điểm I và k là kích thước của table. Nếu $2^{|T_i|-1}$ > k, thì me(i) < cs(i) đúng.

Chứng minh: Theo định nghĩa, ta có cs(i) = $2^{|T_i|} - r(i) - 1$. Vì k ≥ r(i) nên cs(i) ≥ $2^{|T_i|} - k - 1$ (1). Theo giả thiết, ta có $2^{|T_i|-1}$ > k. Biến đổi giả thiết ta được $2^{|T_i|}$ > 2k + 2 (2). Từ (1) và (2), ta có cs(i) ≥ k+1. Vì k ≥ me(i), cs(i) > me(i) được giữ.

Trường hợp Ti thỏa $2^{|T_i|-1}$ > k thì cs(i) > me(i). Cải tiến *r-skip* chắc chắn được thực hiện trước khi xác định me(i) và cs(i). Vẫn còn thao tác cập nhật tốn O(kL) thời gian xử lý. Cải tiến *t-skip* bỏ luôn cả 2 phép toán cập nhật và thay thế để tăng tốc độ xử lý. Nói cách khác, cải tiến *t-skip* bỏ xử lý transaction T_i nếu chiều dài của nó lớn hơn log(k) + 1.

Trong **t-skip**, frequent count của 1 entry có thể thấp hơn support thật của nó vì một số thao tác cập nhật đã đã bị bỏ qua trong khoảng thời gian trước. Nói cách khác, t-skip có thể tạo ra kết quả false-negatives ngay cả khi thỏa điều kiện Δ(N+1) < σN. Để giải quyết vấn đề này, ta xem xét 2 cách tiếp cận bên dưới.

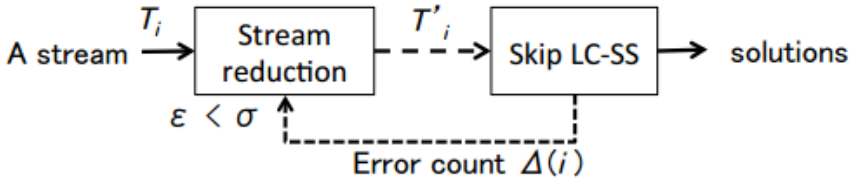
Tiếp cận 1: Thay đổi điều kiện của kết quả. Ta thêm 1 biến X mới cho biết số lần thực hiện t-skip. Sau đó, ta kiểm tra xem $f(\alpha) + \Delta_\alpha + X > \sigma N$ cho mỗi entry α là output của thuật toán. Ta gọi cách tiếp cận này là *t_1-skip*.

Tiếp cận 2: Tăng frequent count của mỗi entry cho mỗi lần t-skip được thực hiện. Khác *t_1-skip*, ta không thay đổi điều kiện của output. Ta gọi tiếp này là *t_2-skip*.

Đánh giá cải tiến t-skip. T-skip không còn duy trì được kết quả như r-skip. Mỗi cách tiếp cận t-skip có nhược điểm riêng. Về khía cạnh xử lý, cách tiếp cận 2 cần duyệt tất cả entries khi t2-skip được thực hiện. Ngược lại, cách tiếp cận 1 có thể làm giảm độ chính xác của thuật toán vì làm tăng số lượng non-frequent itemsets trong ouput. Dẫn đến điều đó là vì cách tiếp cận 1 phải cộng thêm tham số lỗi X cho mỗi entry α ngay cả khi α đó không bị ảnh hưởng bởi phép toán t-skip trước đó.

6.7.2. Stream reduction

Ở đây, ta cải tiến hiệu năng của thuật toán skip LC-SS bằng cách sử dụng *stream reduction* để nén động mỗi transaction. Chiều dài L của một transaction là nhân tố chính cho việc xác định hiệu năng của thuật toán skip LC-SS, theo *Theorem 6* và *Lemma 4*. Ý tưởng của stream reduction nằm ở một sự thật rằng hầu hết các item trong bursty transactions là non-frequent. Theo nguyên lý non-monotonicity, mỗi itemset có bất kì item nào là non-frequent thì nó không frequent. Như vậy, ta có thể loại bỏ những non-frequent items từ mỗi transaction.



Hình 6-7: Mô hình kết hợp giữa stream reduction với skip LC-SS [3]

Hình 6-9 là minh họa cho mô hình kết hợp giữa stream reduction với thuật toán skip LC-SS. Transaction T_i sau khi qua stream reduction sẽ loại bỏ đi những item là non-frequet thành transaction con T'_i. Sau đó, transaction T'_i sẽ đi vào thuật toán skip LC-SS để xử lý. Để loại bỏ các phần tử là non-frequent, stream reduction không

hoạt động độc lập mà phụ thuộc vào error count Δ(i) và minimal support σ từ thuật toán skip LC-SS.

Thuật toán SS-ST được đề xuất cho thành phần stream reduction. Thuật toán SS-ST đạt được bởi việc mở rộng thuật toán SS từ khai thác dòng item thành khai thác dòng transaction. Thuật toán SS-ST tính frequent count c_i cho mỗi item trong transaction T_i tại thời điểm t. Cho Δ(i) là error count tại thời điểm i của thuật toán skip LC-SS. Nếu $c_i \leq \Delta(i)$ thì e nên được xóa khỏi Ti, vì cho mỗi tập con α_e của T_i mà chứa e, $c(\alpha_e) \leq \Delta(i)$ được giữ. Ta không nên lưu những tập con mà có frequent nhỏ hơn Δ(i).

Thuật toán 7: Stream reduction với thuật toán SS-ST

Input: Kích thước table khởi tạo k_{im}, tham số lỗi ϵ, data stream $S = \langle T_1, T_2, \ldots, T_N \rangle$, các error counts $\Delta(1), \ldots, \Delta(N)$.

Output: transaction được giảm $T_1', T_2', \ldots, T_N'$

$i \leftarrow 1, D \leftarrow \emptyset, L_{ave} \leftarrow 0;$
while $i \leq N$ ***do***
    Đọc T_i và $\Delta(i)$;
    $C \leftarrow C + |T_i|;$
    $L_{ave} \leftarrow \frac{C}{i};$
    if $k_{im} < \frac{L_{ave}}{\epsilon}$ ***then***
        $k_{im} \leftarrow \left\lceil \frac{L_{ave}}{\epsilon} \right\rceil;$
    endif
    foreach *item* $e \in T_i$ ***do***
        if $\langle e, c(e) \rangle \in T_i$ ***do***
            $c(e) + +;$
        elseif $|D| < k_{im}$ ***then***
            Thêm entry mới $\langle e, 1 \rangle$ *trong* D
        else
            Thay thế minimal entry với $\langle e, 1 + c(m) \rangle;$
        endif

```
17          endfor
18          Xóa từ mỗi item e mà c(e) ≤ Δ(i);
19          Output transaction được giảm T_i';
20          i++;
21      endwhile
```

Tính đúng đắng của thuật toán SS-ST được chứng minh bên dưới.

Theorem 7: Nếu $k_{im} \geq \frac{L_{ave}}{\epsilon}$, mỗi frequent item e mà sup(e) $\geq \epsilon$N được lưu trong table D.

Chứng minh: Ta dùng phản chứng, giả sử tồn tại một item e mà $k_{im} \geq \frac{L_{ave}}{\epsilon}$ (1) và sup(e) $\geq$ eN mà e không được lưu trong table D. Theo Lemma 2 của bài báo [13] thì min < $\frac{C}{k}$ (2). Từ (1) và (2) ta được min $\leq \frac{C\epsilon}{L_{ave}}$. Mà $L_{ave} = \frac{C}{N}$, ta được min $\leq \epsilon N$. Theo theorem 1 của bài báo [24] thì sup(e) < min nên sup(e) < ϵN mâu thuẩn với giả thiết sup(e) $\geq \epsilon$N. Vậy không tồn tại item e thỏa điều kiện nhưng không nằm trong table D.

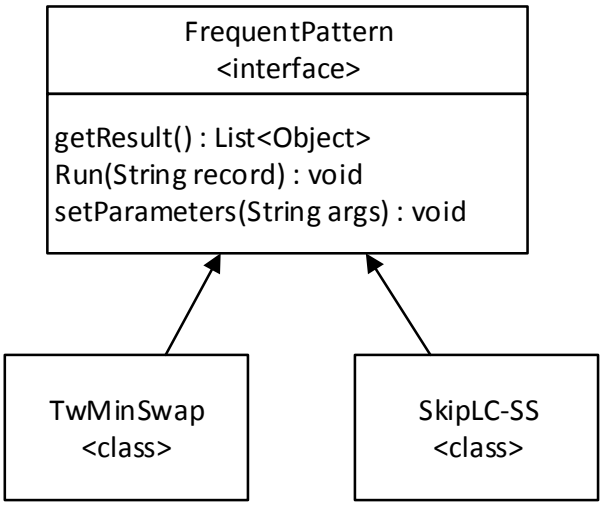
Theo theorem 7, thuật toán SS-ST vẫn duy trì những item mà có độ hổ trợ lớn hơn ϵN và do đó cũng lớn hơn σN; chúng sẽ không bị loại bỏ sớm bởi thuật toán SS-ST. Việc chọn tham số cho SS-ST, ϵ có thể chọn tùy ý nhưng phải thỏa $\epsilon < \sigma$; kích thước ban đầu k_{im} được gán cùng giá trị với kích thước table D của thuật toán Skip LC-SS.

6.8. Vận dụng

Trong mục này, chúng tôi sẽ trình bày cách mà thuật toán Skip LC-SS được tích hợp như thế nào trong hệ thống phân tích Data Stream mà chúng tôi đang xây dựng. Hình 6-8 cho biết interface mà hệ thống cung cấp cho một thuật toán để tích hợp vào hệ thống.

Theo interface FrequentPattern như *hình 6-8 bên dưới* sẽ có function mà thuật toán cần implement cho lớp Skip LC-SS:

- **setParameters**: là nơi thiết lập tham số cho thuật toán. Vì thuật toán yêu cầu 2 tham số là minimum support $\sigma, 0 < \sigma \leq 1$ và $k, k > 0$ cho biết kích thước tối đa của table D, Input của function kiểu String có dạng "σ, k".
- **Run**: là nơi chạy thuật toán xử lý cho mỗi transaction tới. Thuật toán được cài đặt trong function này. Input của function là transaction chứa một chuỗi các IdItems cách nhau bằng dấu phẩy ",";
- **getResults**: là nơi lấy kết quả của thuật toán. Thuật toán trả về kiểu String có định dạng JSon trong đó có danh sách các itemset phổ biến trên ngưỡng σ.



Hình 6-8: Lớp interface tích hợp thuật toán

Lớp Skip LC-SS sẽ được tích hợp vào hệ thống và hoạt động như là thành phần lõi chuyên xử lý dữ liệu của hệ thống. Kết quả chuỗi Json được lưu vào Redis Cluster (Chương 3) và phân tán cho người dùng.

6.9. Kết luận

Trong chương này, chúng ta bàn luận về một phương phá khai thác FIM-DS dựa trên các tiếp cận hướng tài nguyên. Điểm khó khăn của FIM-DS nằm ở chổ làm cách nào để quản lý sự bùng nổ tổ hợp của tập các ứng viên được lưu trữ. Mặc dù có nhiều phương pháp nổi tiếng dựa cách tiếp cận tham số, mỗi cách tiếp cận này đều có nhược điểm của chính nó đó chính là gây ra tràn bộ nhớ đặc biệt là đối với dữ liệu thật khi có sự kiện bursty stream xảy ra. Trong chương này, chúng ta đầu tiên trình bày chặn dưới sử dụng bộ nhớ của các phương pháp tiếp hướng tham số và sau đó đề xuất một

thuật toán hướng LC-SS tài nguyên chỉ yêu cầu O(k) không gian bộ nhớ, mà đạt được bằng việc kết hợp hai thuật toán Lossy Count và Space Saving trong ngữ cảnh của khai thác itemset. Thuật toán Skip LC-SS là cải tiến từ thuật toán LC-SS tăng tốc độ xử lý, chỉ yêu cầu O(kLN) thời gian với k, L và N lần lượt là kích thước tối đa của table D, chiều dài tối đa của transaction trong stream và số lượng transactions. Chúng ta nhấn mạnh rằng thuật toán Skip LC-SS có thể trả ra kết quả mà không bị tràn bộ nhớ, chạy nhanh hơn nhưng vẫn giữ được tính đảm bảo của thuật toán LC-SS. Chúng ta còn nói tới một số cải tiến để tăng hiệu năng của thuật toán Skip LC-SS như các kỹ thuật skip hay dùng stream reduction để giảm kích thước của transactions trước khi xử lý thuật toán. Bên cạnh giới thiệu thuật toán Skip LC-SS, chúng tôi đã trình bày cách triển khai thuật toán vào hệ thống phân tích Data Stream.

CHƯƠNG 7: THỰC NGHIỆM

Trong chương này, luận văn trình bày các thực nghiệm để kiểm tra kết quả và hiệu năng khai thác bằng việc chạy 2 thuật toán TwMinSwap và Skip LC-SS đã được trình bày bên trên, bằng mô hình khai thác dữ liệu dòng được trình bày chương 3.

7.1. Bộ dữ liệu thử nghiệm

Luận văn sẽ thử nghiệm trên 2 loại dữ liệu là item và itemset, với từng loại sẽ dùng 1 bộ dữ liệu riêng. Bộ dữ liệu *food-review* sẽ được dùng cho item và *retail data* dùng cho itemset, những bộ dữ liệu trên được tải về từ 2 nguồn dữ liệu uy tín là *Stanford* [34] và *fimi* [35] . Các thông tin trong *bảng 7.1 và 7.2* là đặc điểm các tập dữ liệu, trong đó bao gồm tổng số review (*#review*), tổng số item phân biệt (*#items*), tổng số giao dịch (*#Trans*), và một số thông tin khác.

Đặc điểm	**Số liệu**
Tổng số dòng (*#review)*	568.454
Số lượng items khác nhau (*#items*)	74.258
Kích thước của một record (*byte*)	70

Bàng 7-1: Đặc điểm của tập dữ liệu movie-review cho loại dữ liệu item

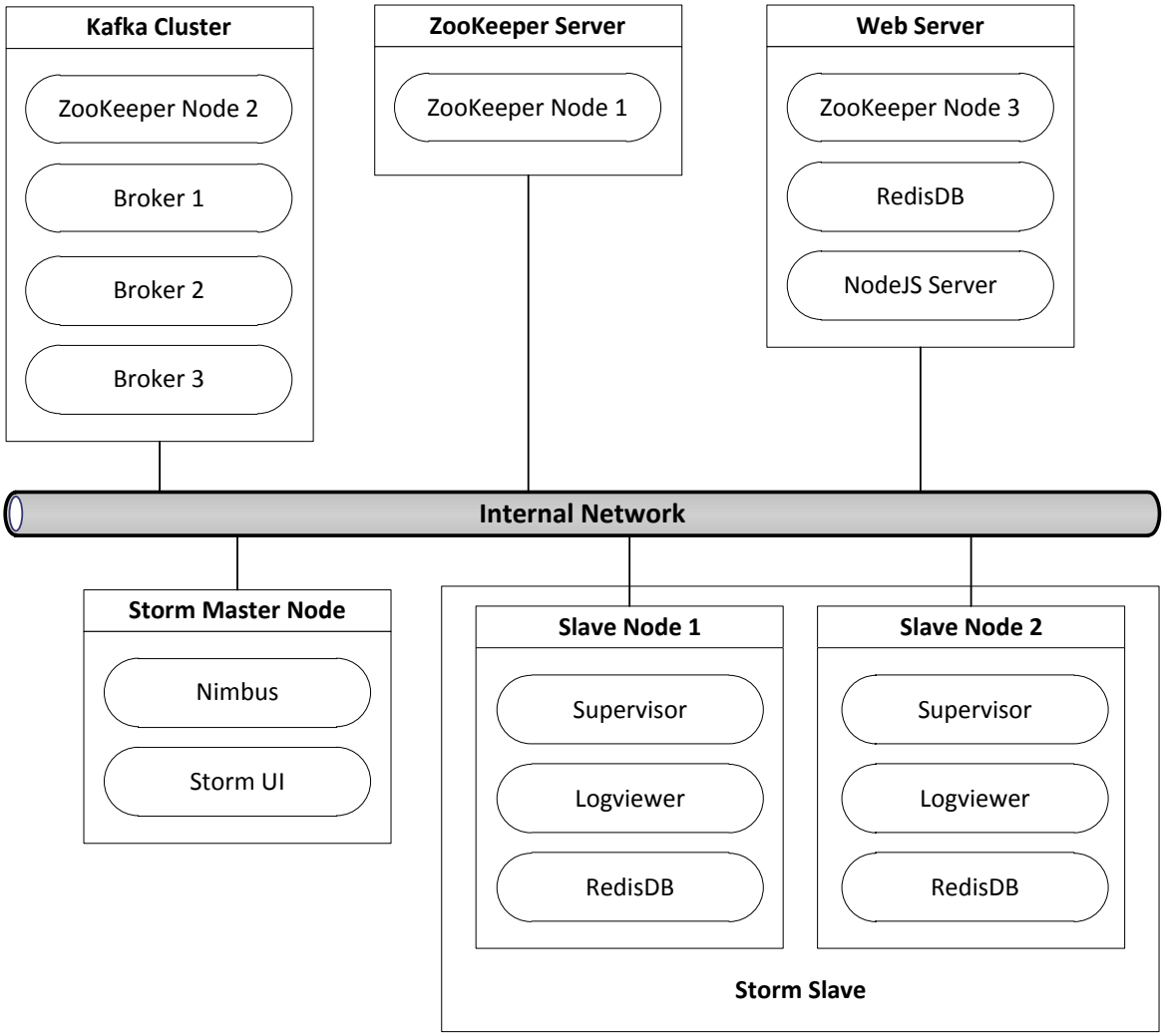
Đặc điểm	**Số liệu**
Số lượng giao dịch (#trans)	88.163
Số lượng items khác nhau (#items)	16.470
Độ dài tối đa của 1 giao dịch (*item*)	76

Bàng 7-2: Đặc điểm của tập dữ liệu retail-data cho loại dữ liệu itemset

Tất cả các thử nghiệm đều được triển khai dựa trên mô hình hệ thống khai thác Data Stream đã được trình bày chi tiết trong *chương 3*, và được thiết lập môi trường thử nghiệm được đề cập chi tiết trong mục tiếp theo.

7.2. Thiết lập môi trường thử nghiệm

Hệ thống server được xây dựng giả lập trên phần mềm VMware workstation 9, bao gồm *6 server vật lý* chạy trên hệ điều hành Ubuntu Sever 14.04, với cấu hình Intel 1 core và 1GB bộ nhớ cho từng server riêng biệt. Mô hình server vật lý của toàn hệ thống được mô như *hình 7-1*.



Hình 7-1: Hệ thống khai thác Data Stream (*physical layer)*

Một server có thể chạy một hoặc nhiều các frameworks khác nhau, điều này giúp ta tiết kiệm được chi phí triển khai hệ thống mà vẫn đạt hiệu năng cao, bởi vì những thành phần tốn nhiều chi phí xử lý, như *Supervisor, Brokers, NodeJS, Nimbus*, được chúng tôi chia sẽ đều ra các server khác nhau. Như ta đã thấy, thành phần ZooKeeper

được chia chạy trên 3 server vật lý riêng biệt, việc này nhằm đảm bảo ZK cluster luôn hoạt động và không bị mất kết nối với toàn hệ thống. Chúng ta phải làm vậy vì ZK cluster là thành phần quan trọng nhất của toàn hệ thống chi tiết đã được để cập trong chương 2.

Trong những phần còn lại chúng tôi sẽ thử nghiệm hệ thống với việc tích hợp 2 thuật toán TwMinSwap và Skip LC-SS chạy trên *Apache Storm version 0.9.5* với 1 worker trên 1 supervisor.

7.3. Khai thác dữ liệu trực tuyến

Mục này trình bày những thông tin hữu ích tìm ẩn trong Data Stream được khai thác bằng 2 thuật toán TwMinSwap và Skip LC-SS. Kết quả của việc khai thác này được thực hiện real-time và được hiển thị thông qua 2 công nghê visualization là *NodeJS* và *D3.js* [36].

7.3.1. Khai thác trên bộ dữ liệu food-review

❖ ***Chủ đề khai thác***

Hiển thị top 10 food hay được review nhất trong thời điểm hiện tại.

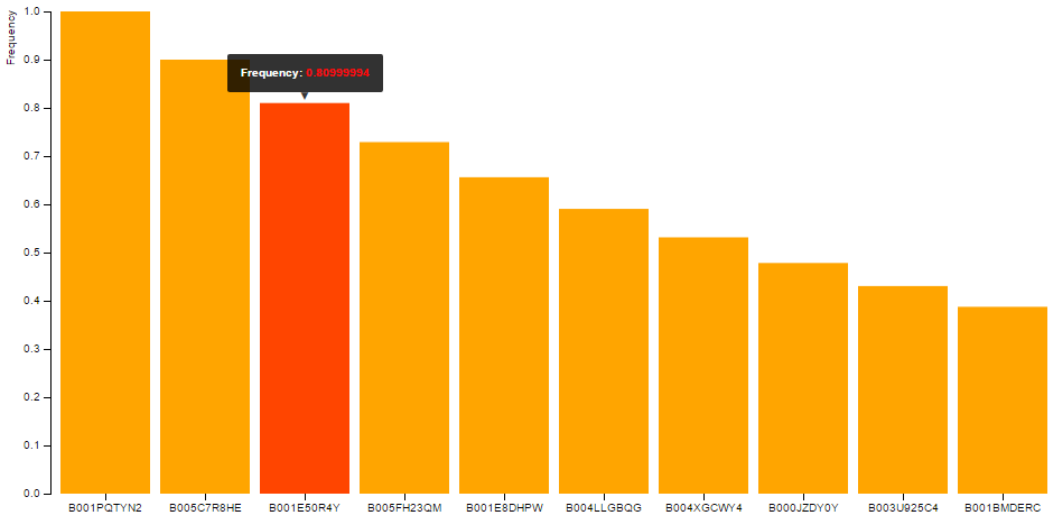
❖ ***Ngữ cảnh:***

Ngữ cảnh real-time report nằm trong một cửa hàng cung cấp thực phẩm. Họ sở hữu một trang web để giới thiệu những thực phẩm mà họ đang bán. Hàng ngày có rất nhiều khách hàng vào trang web của họ để để xem hay để mua hàng trực tuyến. Mỗi lần một người review một món hàng thì họ lại lưu lại thông tin của món hàng được review ấy. Những thông tin ấy rõ ràng không giúp được gì cho họ cho mục đích kinh doanh. Họ cần một real-time report có thể cho họ biết tại thời điểm này món hàng nào hay được review nhất. Với thông tin ấy, họ có điều chỉnh chính sách của mình tập trung vào món hàng đang được review ấy như là khuyến mãi khi mua số lượng, mua 2 tặng 1. Kết quả là họ thu được nhiều lợi nhuận hơn với real-report đó.

❖ ***Hiển thị kết quả khai thác dưới dạng real-time report:***

Kết quả khai thác được hiển thị dưới dạng biểu đồ cột. Có 10 cột tương ứng với 100 thực phẩm được review nhiều nhất kèm theo con số thể hiện mức độ review của

chúng. Các cột được sắp theo thứ tự tăng dần mức độ quan tâm của người dùng. Hình 7-2 thể hiện kết quả của real-time report thể hiện 10 food hay được review nhất.



Hình 7-2: Real-tiem report - Top 10 food hay được review nhất

7.3.2. Khai thác trên bộ dữ liệu retail

❖ ***Chủ đề khai thác***

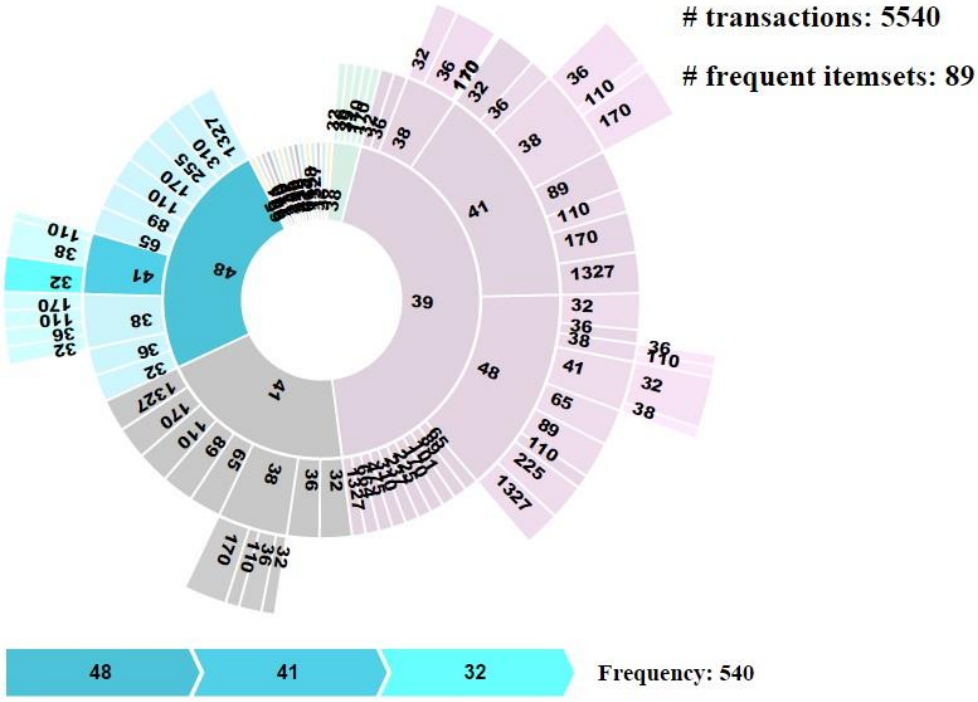
Hiển thị các nhóm món hàng hay được mua chung trong các giao dịch.

❖ ***Ngữ cảnh:***

Ngữ cảnh real-time report nằm trong cửa hàng bán lẻ. Mỗi ngày, có rất nhiều khác mua hàng tới mua hàng của họ. Sau khi thanh toán, họ thu được rất nhiều đơn hàng mà khách hàng của họ đã mua. Với thông tin giao dịch đã ghi nhận, cuối tháng họ report được số lượng hàng hóa đã được mua. Thông tin giúp họ biết được lượng hàng nào cần được nhập thêm hay hàng nào hay được mua. Thông tin ấy giúp ích cho mục đích kinh doanh. Liệu còn thông tin nào mà cửa hàng đã bỏ sót không khai thác. Có một số quy luật tiềm ẩn trong các đơn hàng mà bình thường không nhận ra được bằng mắt thường. Có một số món hàng rất hay được mua chung với nhau. Nếu biết được thông tin, cửa hàng có thể còn tăng doanh thu hơn vì để chung những vật hay được mua gần với nhau. Vi dụ như là áo sơ mi hay được mua chung với cà vạt, sẽ thật tốt khi cả hai ở gần nhau và người mua có thể mua ngay cả hai. Họ sẽ cần một report cho họ biết những thông tin này.

❖ ***.Hiển thị kết quả khai thác dưới dạng real-time report:***

Kết quả khai thác được hiển thị dưới dạng biểu đồ quạt. Biểu đồ có nhiều lớp không đều. Mỗi lớp tương ứng với một item. Một đường đi từ lớp đầu tới lớp cuối tạo thành một itemset phổ biến. Hình 7-3 thể hiện kết quả của real-time report thể hiện nhóm món hàng hay được mua chung trong các giao dịch.



Hình 7-3: Nhóm món hàng hay được mua chung trong các giao dịch

7.4. Hiệu năng sử lý của hệ thống với thuật toán TwMinSwap

Thuật toán được chạy với 2 siêu tham số *k = 100*, và *decay = 0.9* trên bộ dữ liệu *food-review*. Bằng việc cho tốc độ truyền tải dữ liệu thay đổi với những cường độ khác nhau trong từng lần chạy, ta sẽ đo đạt lại số lượng dữ liệu chưa được xử lý còn tồn động bên trong Kafka, để từ đó ước lượng khả năng sử lý tối đa của toàn hệ thống. Dữ liệu đầu vào của 3 lần chạy này tương đương nhau, và được chạy độc lập trên cùng một môi trường.

❖ ***Cách thực hiện thử nghiệm***

Chúng tôi thực hiện thử nghiệm 3 lần riêng biệt, với tốc độ phun dữ liệu lần lược là 12.000, 14.000, 15.000 record/s. Để tính số lượng dữ liệu còn tồn động lại trong Kafka mà chưa xử lý, chúng tôi viết một con *bot* để thực hiện việc thu thập dữ liệu

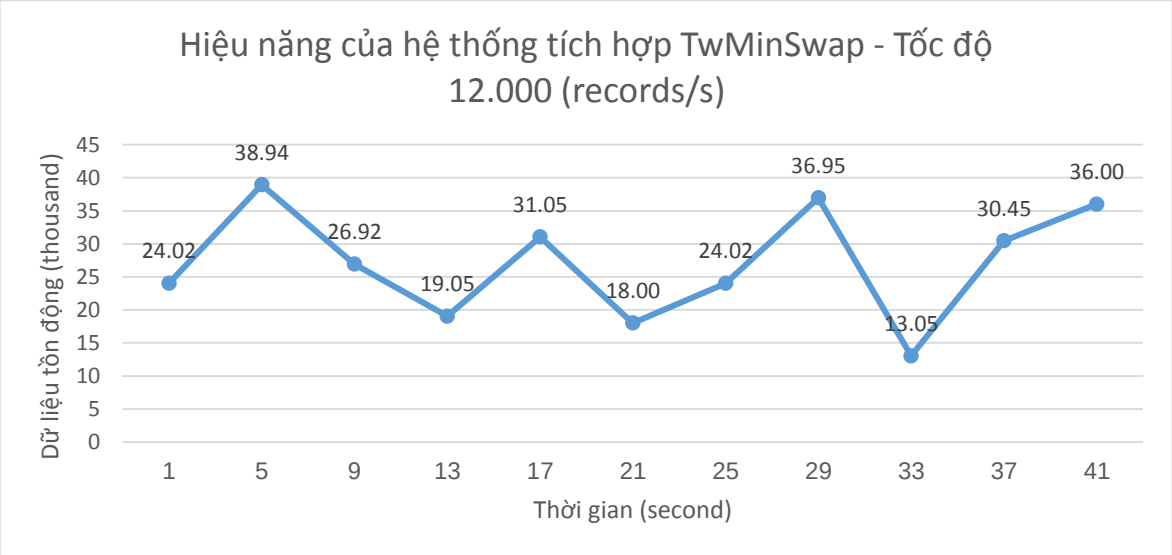
tại Kafka và Storm trong mỗi 4 giây một lần, và ghi Log lại lượng dữ liệu tồn động trong mỗi 4 giây.

❖ ***Mục tiêu của việc thử nghiệm***

Mục tiêu của quá trình thực nghiệm này nhằm ước lượng khả năng xử lý tối đa có thể của thuật toán TwMinSwap khi được tích hợp vào hệ thống bên trên, để có thể cảnh bảo người quản trị khi hệ thống gần trạm tới ngưỡng quá tải. Việc này giúp người quản lý hệ thống có những biệt pháp nâng cập hệ thống hoặc ngăn chặn việc quá tải trước khi xảy ra.

❖ ***Kết quả thu được***

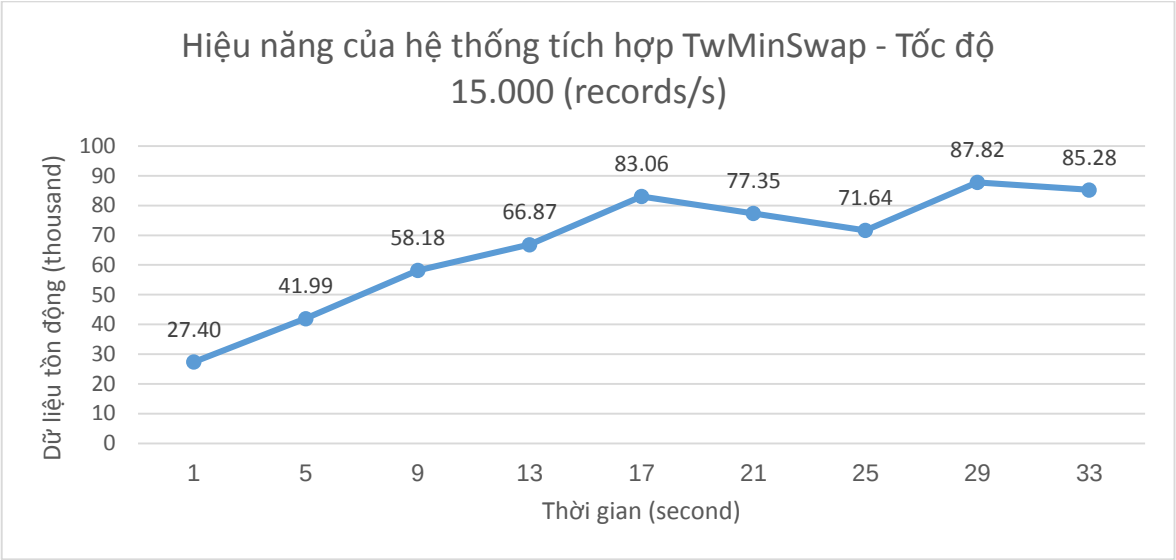
Hình 7-2 được phun với tốc độ 12.000 thì giá trị *Max* là 38.94, và sau đó lượng dữ liệu tồn động không vượt qua được ngưỡng này nữa, mà chỉ giao động lên xuống trong khoảng từ 13 tới 40. Ta có, sự giao động này bởi vì hệ thống Storm xử lý dữ liệu theo từng *small batch*[24], chứ không xử lý từng record một, kích thước của một small batch được Storm tự tính toán sao cho đạt hiệu năng tốt nhất, và ta không thể cấu hình cho việc này được.



Hình 7-4: Hiệu năng của hệ thống tích hợp TwMinSwap (tốc độ 12.000)

[24] Small Batch: một lô gồm nhiều records được gôm lại với nhau để xử lý

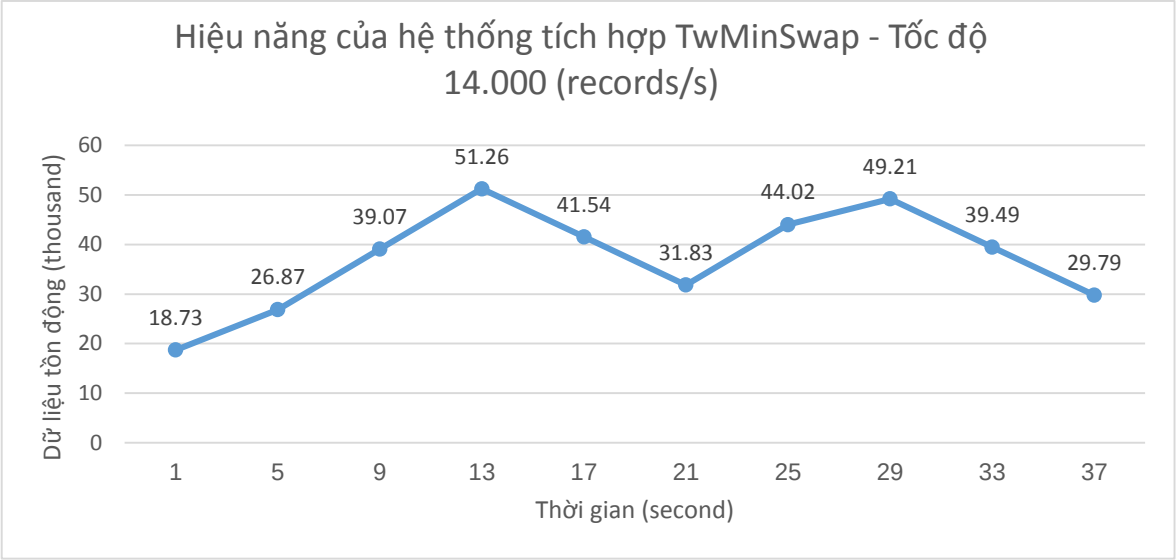
Qua *hình 7-2* cho thấy, với tốc độ này hệ thống vẫn hoạt động tốt, vì biểu đồ không có xu hướng tăng theo đường tuyến tính.



Hình 7-5: Hiệu năng của hệ thống tích hợp TwMinSwap (tốc độ 15.000)

Tuy nhiên, tại *hình 7-3* khi ta tăng tốc độ lên 15.000 thì có một sự thay đổi rõ rệt. Lượng dữ liệu tồn đọng có xu hướng tăng liên tục theo đường tuyến tính. Điều này cho thấy, nếu tốc độ này vẫn không thay đổi thì dữ liệu tồn động sẽ càng ngày càng tăng, và độ trễ cũng tăng dần theo.

Chúng tôi giảm tốc độ này xuống mức 14.000 records/s ra thu được kết quả như *hình 7-4*. Ta có thể thấy, ngưỡng *Max* là 51.26 và cao hơn ngưỡng *Max* khi ta chạy với tốc độ 12.000, điều này cho thấy với mỗi tốc độ khác nhau ta có một ngưỡng Max khác nhau. Tuy nhiên, với tốc độ phun dữ liệu như vậy thì hệ thống vẫn hoạt động bình thường, bởi vì lượng dữ liệu dồn chỉ giao động quanh mức 18 tới 52, và không có xu hướng tăng theo đường tuyến tính.



Hình 7-6: Hiệu năng của hệ thống tích hợp TwMinSwap (tốc độ 14.000)

❖ ***Kết luận***

Qua 3 thử nghiệm trên, ta thấy được với hệ thống cấu hình thấp như trên và tích hợp thuật toán TwMinSwap thì ngưỡng xử lý tối đa sấp xỉ 14.000 record/s. Với tốc độ này trở xuống thì hệ thống vẫn hoạt động tốt, độ dồn ứ dữ liệu và độ trễ sẽ không tăng dần theo thời gian.

Dựa vào những số liệu này, ta có thể xây dựng một hệ thống giám sát để cảnh báo khi hệ thống sắp bị quá tải. Bên cạnh đó, ta có thể cải thiện hiệu năng sử lý của hệ thống bằng cách khai báo thêm nhiều workers khác cùng nhau xử lý song song cho một ứng dụng, hoặc tăng cấu hình phần cứng cho những server xử lý của Storm.

CHƯƠNG 8: KẾT QUẢ VÀ HƯỚNG PHÁT TRIỂN

Trong chương này, chúng tôi sẽ trình bày về kết quả đạt được của khóa luận, và hướng phát triển trong tương lại của hệ thống cũng như việc tích hợp thêm thuật toán vào hệ thống. Chương này còn chỉ ra một số hạn chế của hệ thống và thuật toán tích hợp.

8.1. Kết quả khóa luận

8.1.1. Kết quả thu được từ để tại khóa luận

- Trong khóa luận đã trình bày về mô hình tổng quát cần có của một hệ thống khai thác dữ liệu thời gian thực. Dựa vào bộ khung và những tính chất cần có của một hệ thống khai thác dữ liệu thời gian thực *chúng tôi đã đề xuất một mẫu kiến trúc hệ thống để khai thác Data Stream, và triển khai thực nghiệm hệ thống.*
- Bên cạnh đó, bằng việc *tích hợp công nghệ mới về lưu trữ dữ liệu phân tán trên Ram là Redis Cluster*, tốc độ lưu trữ và chí phí truyền tải dữ liệu qua những nơi lưu trữ bên ngoài cũng được cải thiện rất nhiều. Thành phần này đặc biệt thể hiện ưu điểm trong việc xây dựng các ứng dụng real-time như là *Dashboard*, vì tốc độ nhanh mà *vẫn đảm bảo tính nhất quán của dữ liệu*.
- Hơn thế nữa, *chúng tôi còn xây dựng một bộ thư viện* dựa trên những API của Storm, Redis, và Kafka *để tích hợp những thuật toán hiện đại* về khai thác mẫu phổ biến vào trong hệ thống chúng tôi đã đề xuất.
- Để giải quyết 2 bài toán lớn về khai thác item phổ biến và itemset phổ biến trên Data Stream, chúng tôi đã *nghiêng cứu 2 paper* [2] [3], và *tích hợp chúng vào hệ thống* để giải quyết những vấn đề thực tế khai triển khai. Những paper này được đánh giá rất cao trên KDD và mới vừa được công bố gần đây.
- Trong phần thực nghiệm*, chúng tôi trình bày 3 report khác nhau*. Report 1 trình bày *kết quả khai thác real-time, report 2 chúng tôi đo hiệu năng về ngưỡng xử lý ổn định của toàn hệ thống* với những tốc độ phun dữ liệu khác nhau.

8.1.2. Một số hạn chế của khóa luận

- Hệ thống chưa đề cập tới việc lưu trữ toàn bộ Data Stream vào các kho dữ liệu để thực hiện việc khai thác sau này khi cần. Vấn đề đặt ra ở đây là quá trình ETL thường lâu hơn nhiều so với tốc độ xử lý của Storm.
- Hệ thống hiện tại chỉ thực hiện khai thác trực tiếp trên Data Stream, chưa đề cập tới việc áp dụng những thuật toán máy học.
- Thuật toán được tích hợp vào hệ thống chưa có khả năng sử lý song song trên nhiều workers khác nhau, điều này gây hạn chế về hiệu năng toàn hệ thống trong trường hợp ta cần phát triển hệ thống lơn hơn, điều này sẽ là một vấn đề lớn.
- Mặc dù Storm có hổ trợ Storm UI để giám sát Storm, tuy nhiên đây chỉ là một ứng dụng, nó chỉ có thể giám sát các ứng dụng chạy trong Storm cluster. Còn để giám sát về hiệu năng toàn hệ thống và các nodes con của cluster thì cần một hệ thống khác hổ trợ làm việc này.
- Thuật toán khai thác tập itemset phổ biến, tuy giải quyết được bài toàn Bursty hiệu quả, nhưng bù lại tốc độ xử lý không cao.

8.2. Hướng phát triển

8.2.1. Hướng phát triển hệ thống

- Trong tương lại, hệ thống cần cải thiện hơn về khả năng áp dụng những thuật toán về máy học, nhờ vào bộ thư viện TridentML và cơ chế truy vấn trực tuyến trên Data Stream của Storm là DRPC.
- Hệ thống nên cân nhắc tới việc tích hợp thêm thành phần giám sát hệ thống (*Monitoring System*) để có thể thực hiện cảnh báo, và hiển thị thông tin từng server một cách chi tiết hơn, một trong những hệ thống nổi bật là Ganglia.
- Việc lưu trữ toàn bộ Data Stream vào kho dữ liệu thương không đảm bảo hiệu năng do quá trình ETL mất quá nhiều thời gian. Để làm được việc này hệ thống cần nâng cấp khả năng lưu trữ dữ liệu lớn như là hệ thống lưu trữ phân tán HDFS của Hadoop.

8.2.2. Hướng phát triển thuật toán

- Trong tương lai, chúng tôi sẽ tiếp tục tìm hiểu thêm một số thuật toán về những hướng khác như là clustering, classification để tích hợp vào hệ thống.
- Nâng cấp thuật toán TwMinSwap để nó có khả năng xử lý song song, nhằm giải quyết những hạn chế về hiệu năng của hệ thống khi tích hợp TwMinSwap.

TÀI LIỆU THAM KHẢO

[1] Byron Ellis, Real-Time Analytics Techniques to Analyze and Visualize Streaming Data, John Wiley & Sons , 2014.

[2] Y.Lim, J.Choi and U.Kang, "Fast, Accurate, and Space-efficient Tracking of Time-weighed Frequetn Items from Data Streams," 2014, pp. 1109-1118.

[3] Y.Yamamoto, K.Iwanuma and S.Fukuda, "Resource-oriented Approximation for Frequent Itemset Mining from Bursty Data Streams," vol. 14, 2014, pp. 205-216.

[4] Charu C. Aggarwal, Jiawei Han, "Chapter 9 Frequent Pattern Mining in Data Streams," in *Frequent Pattern Mining*, Springer International Publishing Switzerland, 2014, pp. 199-224.

[5] M. M. G. J.Gama, Learning from Data Streams, Springer, 2007.

[6] Apache Storm, [Online]. Available: http://storm.apache.org/documentation/.

[7] Apache Hadoop, [Online]. Available: https://hadoop.apache.org/.

[8] Nishant Garg, Apache Kafka, PACKT , 2014.

[9] NodeJs, [Online]. Available: https://nodejs.org/.

[10] Rohit Rai, Socket.IO Real-time Web Application Development, 2013.

[11] J.Han, M.Kamber and J.Pei, Data Mining Concepts and Techniques, 3rd, Morgan Kaufmann, 2012.

[12] R.Agrawal and R.Srikant, "Fast algorithms for mining association rules in large databases," 1994, pp. 487–499.

[13] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," 2000, pp. 13–23.

[14] W.Li, J.Han, and J.Pei, "Cmar: Accurate and efficient classification based on multiple class-association rules," 2001, pp. 369–376.

[15] Mohammed J. Zaki and Charu C. Aggarwal, "Xrules: an effective structural classifier for xml data," 2003, pp. 316–325.

[16] A.Mala and F.R.Dhanaseelan, "Data Stream Mining Algorithms - A Review of Issues and Existing Approaches," 2011, pp. 2726 – 2732.

[17] J.Gama, Knowledge Discovery from Data Streams, 2010.

[18] P. B. Gibbons and Y. Matias, "New sampling-based summary statistics for improbing approximate query answer," 1998.

[19] M. J. Fischer and S. L. Salzberg, "Finding a majority among n votes: Solution to problem 81-5 (Journal of Algorithms, june 1981)," 1982, pp. 362-380.

[20] J. Misra and D. Gries, "Finding repeated elements. Science of Computer Programming," 1982, pp. 143-152.

[21] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," 2002.

[22] R. M. Karp, S. Shenker, and C. H. Papadimitriou, " A simple algorithm for finding frequent elements in streams and bags," 2003, pp. 51-55.

[23] G. S. Manku and R. Motwani, "Approximate frequent counts over data streams," 2002, pp. 346–357.

[24] A.Metwally, D.Agrawal and A.E.Abbadi, "Efficient Computation of Frequent and top-k Elements in Data Streams," 2005, pp. 398-412.

[25] G. Cormode and S. Muthukrisnan, "An improved data stream summary: The count-min sketch and its applications," 2003, pp. 51-55.

[26] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," 2004.

[27] C. Jin, W. Gian, C. Sha, J. X. Yu, and A. Zhou, "Dynamically maintaining frequent items over a data stream," 2003.

[28] J. S. Vitter, "Random sampling with a reservoir," 1985, pp. 37-57.

[29] S.Zhang, L.Chen, and L.Tu, "Frequent items mining on data stream based on time fading factor," 2009.

[30] L. Chen and Q. Mei, "Mining frequent items in data stream using time fading model. Information Sciences," 2014, pp. 54–69.

[31] R. Jin and G. Agrawal, "An algorithm for in-core frequent itemset mining on streaming data," 2005, pp. 210-217.

[32] H.-F. Li, M.-K. Shan and S.-Y. Lee, "DSM-FI: an efficient algorithm for mining frequent itemsets in data streams," 2008, pp. 79–97.

[33] B.Lin, W.S.Ho, B.Kao and C.K.Chui, "Adaptive frequence counting over bursty data streams," 2007, pp. 516-523.

[34] Stanford Source, [Online]. Available: https://snap.stanford.edu/data/ .

[35] Fimi Source, [Online]. Available: http://fimi.ua.ac.be/data/ .

[36] D3Js, [Online]. Available: http://d3js.org/ .