ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

ỨNG DỤNG KIẾN TRÚC DỮ LIỆU LỚN TRONG XỬ LÝ DỮ LIỆU NGÀNH VIỄN THÔNG

Lê Đức Tài

tai.ld195163@sis.hust.edu.vn

Chuyên ngành: Toán Tin

Chuyên sâu: Tin học

Giảng viên hướng dẫn:	ThS. Nguyễn Tuấn Dũng	
Bộ môn:	Toán Tin	Chữ ký của GVHD
Viện:	Viện Toán ứng dụng và Tin học	

HÀ NỘI, 2023

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC
——————o0o——————

ỨNG DỤNG KIẾN TRÚC DỮ LIỆU LỚN TRONG XỬ LÝ DỮ LIỆU NGÀNH VIỄN THÔNG

ĐỒ ÁN TỐT NGHIỆP

Chuyên ngành: Toán Tin
Chuyên sâu: Tin học

Giảng viên hướng dẫn:	ThS. Nguyễn Tuấn Dũng
Sinh viên thực hiện:	Lê Đức Tài
Mã sinh viên:	20195163
Lớp:	CTTN Toán Tin K64

HÀ NỘI, 2023

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

1. Mục tiêu và nội dung của đồ án

(a)

(b)

(c)

2. Kết quả đạt được

(a)

(b)

(c)

3. Ý thức làm việc của sinh viên:

(a)

(b)

(c)

Hà Nội, ngày ... tháng ... năm 2023

Giảng viên hướng dẫn

ThS. Nguyễn Tuấn Dũng

Lời cảm ơn

Lời đầu tiên, em xin gửi lời cảm ơn chân thành nhất đến thầy cô viện Toán ứng dụng và Tin học đã giúp đỡ và dạy dỗ em trong suốt những năm tháng học tập tại Đại học Bách khoa Hà Nội. Trải qua bốn năm học tập, nhờ có sự đồng hành của các thầy cô, em đã tiếp thu được những kiến thức quý báu và tích lũy những kỹ năng quan trọng để bước chân vào cuộc sống.

Em xin chân thành cảm ơn các anh chị trong dự án nPm - Công ty cổ phần Telsoft đã cung cấp dữ liệu và tài liệu để em thực hiện đồ án này.

Đặc biệt, em xin gửi lời cảm ơn tới thầy Nguyễn Tuấn Dũng - giảng viên hướng dẫn đồ án tốt nghiệp của em. Cảm ơn thầy vì đã hướng dẫn và chỉ bảo em tận tình trong quá trình nghiên cứu và thực hiện đồ án. Cảm ơn thầy vì những kiến thức và kinh nghiệm quý báu đã chia sẻ với em, đó là nguồn động lực vô cùng quý giá cho sự phát triển và tiến bộ của em trong học tập và sự nghiệp sau này.

Và cuối cùng, em xin cảm ơn bố mẹ và gia đình luôn tin tưởng em. Xin cảm ơn những người bạn trong lớp CTTN Toán Tin K64 đã luôn bên cạnh, giúp đỡ em trong học tập.

Hà Nội, ngày ... tháng ... năm 2023

Sinh viên thực hiện

Lê Đức Tài

Mục lục

Danh sách hình vẽ

Danh sách bảng

Danh mục từ viết tắt

Ký hiệu chữ viết tắt	Chữ viết đầy đủ
EMS	Hệ thống quản lý phần tử
HDFS	Hệ thống lưu trữ phân tán
KPI	Chỉ số hiệu suất chính
FTP	Giao thức truyền tải tập tin
SFTP	Giao thức truyền tải tập tin an toàn

Mở đầu

Viễn thông là một phương tiện quan trọng để kết nối mọi người trên toàn cầu và loại bỏ các rào cản giao tiếp. Tuy nhiên, ngành này đối mặt với nhiều khó khăn trong việc cải thiện kết nối và mất nhiều năm để loại bỏ những rào cản đó. Sự ứng dụng của dữ liệu lớn trong viễn thông đã giải quyết các vấn đề này và mở ra kỷ nguyên mới cho ngành viễn thông.

Dữ liệu lớn mang đến nhiều cơ hội mới cho tổ chức, doanh nghiệp và cơ quan thông qua việc tạo ra giá trị thông qua cải thiện chất lượng dịch vụ, quyết định đúng đắn và kịp thời, cạnh tranh hiệu quả và theo kịp xu hướng phát triển. Việc coi trọng và đầu tư hợp lý vào dữ liệu lớn được coi là một tài sản chiến lược.

Dữ liệu lớn trong viễn thông đem lại nhiều lợi ích quan trọng, bao gồm:

1. Tối ưu hóa mạng lưới kết nối: Nhờ phân tích dữ liệu lớn, các công ty viễn thông theo dõi và điều chỉnh lưu lượng mạng một cách hiệu quả. Điều này giúp họ đưa ra quyết định phát triển mạng, tối ưu hóa chi phí mở rộng mạng và cải thiện chất lượng dịch vụ.

2. Trải nghiệm người dùng: Dữ liệu lớn cho phép các doanh nghiệp viễn thông điều chỉnh và đáp ứng đúng nhu cầu của khách hàng. Thông qua các báo cáo thống kê và hệ thống hỗ trợ khách hàng trực tuyến, việc tương tác và giải quyết vấn đề của người dùng trở nên hiệu quả hơn.

Việc ứng dụng dữ liệu lớn trong ngành viễn thông đã mang lại nhiều lợi ích đáng kể, giúp cải thiện hiệu năng, trải nghiệm người dùng và đảm bảo an toàn cho ngành này. Điều này đã thúc đẩy sự phát triển và chuyển đổi số trong lĩnh vực viễn thông, tạo ra nhiều tiềm năng và cơ hội mới cho ngành này trong tương lai.

Chương 1

Giới thiệu

1.1 Đặt vấn đề

Tập đoàn Bưu chính Viễn thông Việt Nam (VNPT), một trong những công ty hàng đầu cung cấp dịch vụ viễn thông tại Việt Nam, quản lý hơn 118.000 trạm phát sóng trải rộng khắp cả nước. Để đáp ứng các yêu cầu ngày càng cao từ khách hàng, VNPT đang tập trung vào việc nâng cao chất lượng mạng thông qua việc quản lý hiệu năng vô tuyến, nhằm xác định và khắc phục các vấn đề ảnh hưởng đến chất lượng mạng, từ đó cải thiện trải nghiệm người dùng.

Mỗi trạm phát sóng trong mạng viễn thông được trang bị các counter (thiết bị đếm) để ghi nhận và đếm các chỉ số quan trọng về hoạt động của trạm. Các counter này thu thập thông tin chi tiết và theo dõi sự thay đổi trong hệ thống. VNPT sử dụng dữ liệu từ các counter thu thập được từ trạm trên toàn quốc để tính toán các KPI để đánh giá hiệu suất hoạt động của từng trạm và từng khu vực. Thông qua việc phân tích dữ liệu KPI, VNPT có thể tìm ra các trạm gặp vấn đề và thực hiện các biện pháp sửa chữa.

Hiện nay, VNPT sử dụng hệ thống vô tuyến di động 2G/3G/4G từ các nhà cung cấp hàng đầu như Ericsson, Nokia, Huawei, Alcatel và Motorola. Với lượng dữ liệu từ các counter được tạo ra từ các trạm khá là lớn, VNPT cần có một hệ thống phân tích dữ liệu lớn mạnh mẽ và toàn diện để hiệu quả thu thập,

lưu trữ và xử lý dữ liệu từ các trạm phát sóng.

Hơn nữa, VNPT cần cung cấp một giao diện người dùng cuối trực quan và tích hợp bản đồ số. Giao diện này giúp người dùng dễ dàng truy cập và tương tác với thông tin quan trọng. Tích hợp bản đồ số cho phép người dùng hiển thị và định vị các trạm phát sóng trên một giao diện trực quan, giúp họ theo dõi và quản lý mạng viễn thông một cách thuận tiện. Sự tương tác với các trạm trên bản đồ số cũng hỗ trợ người dùng thực hiện các thao tác quản lý và xử lý sự cố nhanh chóng và hiệu quả. Điều này giúp đảm bảo thông tin về các đối tượng mạng lưới được đưa ra chính xác và kịp thời tới kỹ sư hiện trường, kỹ sư vận hành, đến các cấp lãnh đạo tham gia công tác điều hành và tối ưu hóa chất lượng mạng.

1.2 Mục tiêu và phạm vi của đề tài tốt nghiệp

Qua các thông tin được đưa ra bên trên, để giải quyết những yêu cầu đã đề cập, mục tiêu trong nội dung của đồ án này, em sẽ xây dựng một hệ thống đáp ứng các tiêu chí sau đây:

- Hệ thống bao gồm đầy dủ các thành phần thu thập, xử lý, lưu trữ và trực quan hóa dữ liệu.
- Đáp ứng được các yêu cầu mà một hệ thống xử lý dữ liệu lớn cần phải có, đó là khả năng mở rộng, tính tương thích với nhiều công nghệ dữ liệu lớn khác nhau và khả năng chịu lỗi của mô hình.

Trong phạm vi đồ án này, em sẽ xây dựng hệ thống xử lý với dữ liệu của hãng Nokia.

1.3 Bố cục của đồ án

Bố cục của bài báo cáo tốt nghiệp này được trình bày như sau đây:

Chương 2: Cơ sở lý thuyết và các khái niệm: Chương này sẽ cung cấp một cái nhìn tổng quan về dữ liệu lớn và các đặc trưng của nó, đồng thời so sánh dữ liệu lớn với dữ liệu truyền thống để hiểu rõ hơn về sự khác biệt giữa chúng. Sau đó, chương sẽ giới thiệu sự kết hợp giữa kho dữ liệu và hồ dữ liệu để xử lý dữ liệu lớn.

Chương 3: Khảo sát yêu cầu và phân tích: Chương này trình bày những hạn chế của hệ thống hiện tại. Sau đó, chương này xác định các yêu cầu của hệ thống mới, bao gồm cả chức năng mà hệ thống phải có và các yêu cầu phi chức năng hệ thống phải có.

Chương 4: Kiến trúc hệ thống: Trong chương này, em sẽ đề xuất kiến trúc tổng quan của hệ thống và đi vào trình bày chi tiết các công nghệ dùng trong từng thành phần của kiến trúc hệ thống.

Chương 5: Giải pháp kỹ thuật và kết quả thử nghiệm: Chương này, em sẽ trình bày giải pháp kỹ thuật của hệ thống và thực hiện xây dựng hệ thống bao gồm xây dựng metadata các luồng xử lý của các thành phần trong kiến trúc hệ thống. Và cuối cùng kết quả thử nghiệm chạy trên môi trường kiểm thử.

Và cuối cùng chính là kết luận và hướng phát triển của đề tài

Chương 2

Cơ sở lý thuyết và các khái niệm

2.1 Dữ liệu lớn

2.1.1 Khái niệm dữ liệu lớn

- Theo wikipedia [12]: Dữ liệu lớn là một thuật ngữ chỉ bộ dữ liệu lớn hoặc phức tạp mà các phương pháp truyên thống không đủ các ứng dụng để xử lý dữ liệu này.
- Theo Gartner [3]: Dữ liệu lớn là những nguồn thông tin có đặc điểm chung khối lượng lớn, tốc độ nhanh và dữ liệu định dạng dưới nhiều hình thức khác nhau, do đó muốn khai thác được đòi hỏi phải có hình thức xử lý mới để đưa ra quyết định, khám phá và tối ưu hóa quy trình.

2.1.2 Đặc trưng của dữ liệu lớn

Dữ liệu lớn có 5 đặc trưng cơ bản sau:

- **Tốc độ (Velocity)**: Trong thời đại ngày nay, tốc độ dữ liệu được tạo ra và xử lý để đáp ứng nhu cầu truy xuất của khách hàng và các công ty công nghệ. Dữ liệu lớn thường được xử lý trong thời gian thực.
- **Độ tin cậy/chính xác (Veracity)**: do dữ liệu lớn đa dạng về các kiểu dữ liệu nên sự không thống nhất của các tập dữ liệu có thể cản trở các quy trình xử

lý và quản lý dữ liệu lớn. Do đó, độ chính xác của dữ liệu lớn càng cao thì càng đảm bảo độ chính xác trong các truy xuất dữ liệu.

- **Giá trị (Value)**: Khái niệm này liên quan đến chất lượng của dữ liệu khai thác được, điều này ảnh hưởng rất lớn đến việc phân tích dữ liệu. Tính chất này cũng là yếu tố mà doanh nghiệp hay nhà nghiên cứu quan tâm khi xử lý dữ liệu lớn.
- **Đa dạng (Variety)**: Khái niệm này nói về kiểu dữ liệu và tính chất của dữ liệu. Điều này giúp những người phân tích sử dụng hiệu quả thông tin chi tiết về dữ liệu. Dữ liệu được tập hợp từ những văn bản, hình ảnh, âm thanh, video.
- **Khối lượng dữ liệu (Volume)** : Khái niệm về khối lượng dữ liệu được tạo và lưu trữ. Kích thước của dữ liệu sẽ được đánh giá xem liệu nó có thể được coi là dữ liệu lớn hay không.

Hình 2.1: Đặc trưng dữ liệu lớn (Nguồn [6]).

2.1.3 Sự khác biệt giữa dữ liệu lớn và dữ liệu truyền thống

Dữ liệu lớn khác với dữ liệu truyền thống ở 4 điểm cơ bản: dữ liệu đa dạng hơn; lưu trữ dữ liệu lớn hơn; truy vấn dữ liệu nhanh hơn; độ chính xác cao hơn.

- **Dữ liệu đa dạng hơn**: Dữ liệu lớn thường bao gồm nhiều nguồn và định dạng khác nhau, bao gồm cả dữ liệu cấu trúc và không cấu trúc. Trong khi đó, dữ liệu truyền thống thường tập trung vào dữ liệu cấu trúc từ các nguồn cố định như cơ sở dữ liệu quan hệ.
- **Lưu trữ dữ liệu lớn hơn**: Dữ liệu lớn đòi hỏi các phương pháp lưu trữ và xử lý dữ liệu khác biệt vì khối lượng dữ liệu lớn hơn. Các công nghệ như Hadoop và HDFS được sử dụng để lưu trữ và quản lý dữ liệu lớn. Trong khi đó, dữ liệu truyền thống thường được lưu trữ trong các cơ sở dữ liệu truyền thống như hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) hoặc kho dữ liệu.
- **Truy vấn dữ liệu nhanh hơn**: Dữ liệu lớn thường được xử lý bằng các công nghệ xử lý phân tán và song song như MapReduce và Spark, giúp tăng tốc độ truy vấn dữ liệu. Các hệ thống dữ liệu lớn cũng thường sử dụng các công nghệ lưu trữ bộ nhớ đám mây (in-memory) để cải thiện hiệu suất truy vấn. Trong khi đó, dữ liệu truyền thống thường sử dụng truy vấn cơ sở dữ liệu truyền thống, có thể không hiệu quả với khối lượng dữ liệu lớn.
- **Độ chính xác cao hơn**: Với dữ liệu lớn, có thể thu thập và xử lý một lượng lớn dữ liệu từ nhiều nguồn khác nhau. Điều này tạo ra cơ hội để có được sự chính xác cao hơn khi phân tích và xử lý dữ liệu. Tuy nhiên, dữ liệu truyền thống thường tập trung vào một số lượng hạn chế nguồn dữ liệu, có thể dẫn đến độ chính xác thấp hơn.

2.2 Sự kết hợp kho dữ liệu và hồ dữ liệu

2.2.1 Kho dữ liệu

Theo William H. Inmon [4]: Kho dữ liệu được hiểu là một tập hợp các dữ liệu tương đối ổn định (không hay thay đổi), cập nhật theo thời gian, được tích hợp theo hướng chủ đề nhằm hỗ trợ quá trình tạo quyết định về mặt quản lý .

Kho dữ liệu về bản chất là một cơ sở dữ liệu bình thường, các hệ quản trị cơ sở dữ liệu quản lý và lưu trữ nó như các cơ sở dữ liệu thông thường. Tuy nhiên, nó có thể quản lý dữ liệu lớn và hỗ trợ truy vấn. Nên điểm khác biệt giữa kho dữ liệu và cơ sở dữ liệu là ở quan niệm, cách nhìn nhận vấn đề.

Một kho dữ liệu điển hình thường bao gồm các yếu tố sau:

- Cơ sở dữ liệu quan hệ để lưu trữ và quản lý dữ liệu.
- Giải pháp trích xuất, biến đổi và lưu trữ (ETL) để chuẩn bị dữ liệu cho phân tích.
- Khả năng phân tích thống kê, báo cáo và khai thác dữ liệu.
- Các công cụ phân tích để trực quan hóa và trình bày dữ liệu cho người dùng doanh nghiệp.
- Các ứng dụng phân tích phức tạp hơn có thể tạo ra tri thức và thực hiện các hành động thông qua việc sử dụng các thuật toán học máy và trí tuệ nhân tạo.

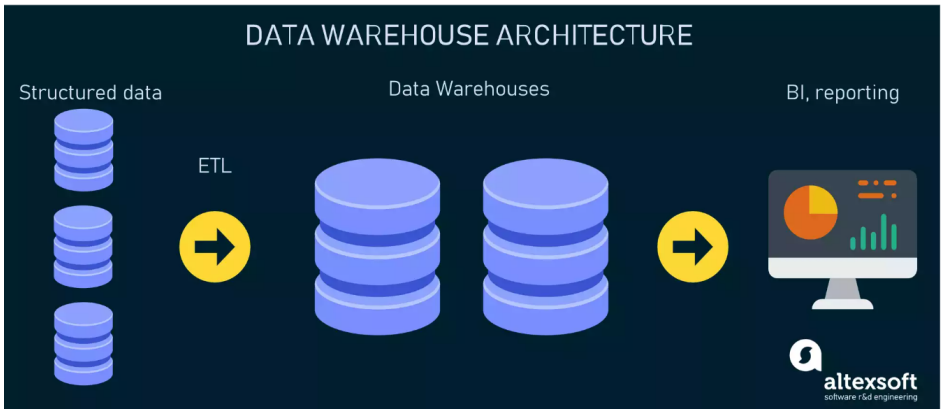
Kho dữ liệu chính là cơ sở dữ liệu chuyên dùng cho tạo báo cáo và phân tích dữ liệu. Nó vừa hỗ trợ các truy vấn phức tạp, vừa là điểm tập trung dữ liệu của các nguồn khác nhau để có được thông tin phân tích đầy đủ nhất. Theo đó, kho dữ liệu là một tập dữ liệu hướng chủ đề, toàn vẹn, không bị rỏ rỉ mất mát và có giá trị lịch sử. Cụ thể các tính chất đó như sau:

- **Tính hướng chủ đề (Subject-oriented)**: Mục đích của Kho dữ liệu là phục vụ các yêu cầu phân tích, hoặc khai phá cụ thể được gọi là chủ đề. Ví dụ với chủ đề phân tích nhân sự thì có thể bao gồm các độ đo về doanh thu của từng người, số ngày nghỉ trong tháng, số dự án tham gia trong tháng, theo các chiều phân tích: thời gian, chi nhánh, sản phẩm,. . . Tập trung vào việc mô hình hóa và phân tích dữ liệu cho các nhà đưa ra quyết định, mà không tập trung vào các hoạt động hay các xử lý giao dịch hàng ngày. Ngoài ra

còn cung cấp một khung nhìn đơn giản và sức tích xung quanh các sự kiện của các chủ đề.

- **Tính toàn vẹn (Integrated)**: Giải quyết các khó khăn trong việc kết hợp dữ liệu từ nhiều nguồn dữ liệu khác nhau, giải quyết các sai khác về tên trường dữ liệu (dữ liệu khác nhau nhưng tên giống nhau), ý nghĩa dữ liệu (tên giống nhau nhưng dữ liệu khác nhau), định dạng dữ liệu (tên và ý nghĩa giống nhau nhưng kiểu dữ liệu khác nhau).
- **Tính bất biến (Nonvolatile)**: Quy định rằng dữ liệu phải thống nhất theo thời gian (bằng cách hạn chế tối đa sửa đổi hoặc xóa dữ liệu), từ đó làm tăng quy mô dữ liệu lên đáng kể so với hệ thống nghiệp vụ (5-10 năm so với 2 đến 6 tháng như cơ sở dữ liệu thông thường).
- **Giá trị lịch sử (Time-varying)**: Gắn thời gian và có tính lịch sử: Dữ liệu trong Kho dữ liệu bao gồm cả quá khứ và hiện tại. Yếu tố thời gian được lưu trữ trong cơ sở dữ liệu.

Kiến trúc tổng quan kho dữ liệu



Hình 2.2: Kiến trúc tổng quan của kho dữ liệu (Nguồn: [1]).

Kho dữ liệu có những hạn chế sau:

- Không linh hoạt trong việc xử lý dữ liệu không cấu trúc: Kho dữ liệu truyền thống thường không thể hiện sự linh hoạt khi xử lý dữ liệu không cấu trúc

như dữ liệu văn bản không được cấu trúc, hình ảnh, video, và các dạng dữ liệu đa phương tiện khác.

- Chi phí cao và hiệu suất không cao: Xây dựng và duy trì các kho dữ liệu truyền thống có thể rất tốn kém và không hiệu quả đối với việc xử lý lượng dữ liệu ngày càng tăng.
- Không phù hợp với xử lý dữ liệu phức tạp: Cấu trúc của kho dữ liệu thường không đáp ứng tốt cho các tác vụ xử lý dữ liệu phức tạp như học máy và phân tích dữ liệu lớn, nơi các tác vụ này thường đòi hỏi sự truy cập và tích hợp dữ liệu từ nhiều nguồn khác nhau.
- Không thể mở rộng dễ dàng: Kho dữ liệu truyền thống có thể gặp khó khăn trong việc mở rộng và thích ứng với nhu cầu dữ liệu ngày càng tăng của tổ chức.

2.2.2 Hồ dữ liệu

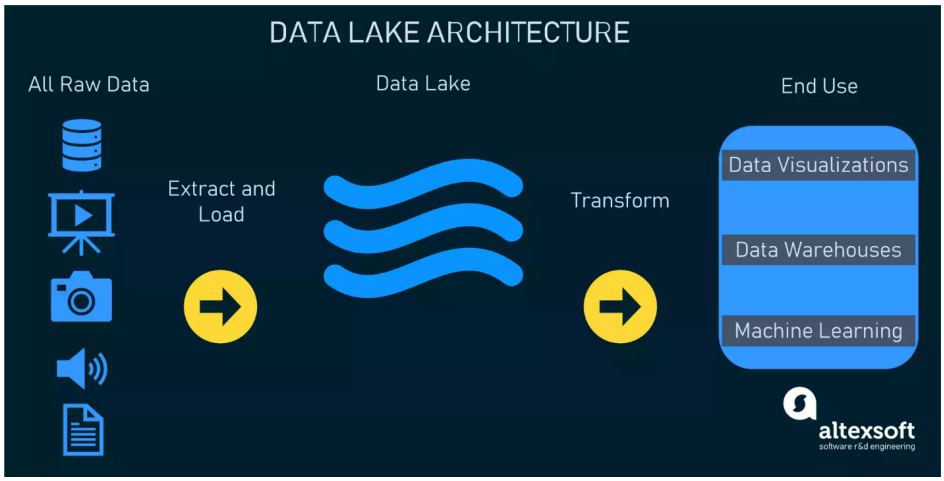
Hồ dữ liệu ra đời nhằm giải quyết những hạn chế của kho dữ liệu và các giải pháp lưu trữ dữ liệu truyền thống.

Hồ dữ liệu là một kiểu kiến trúc lưu trữ dữ liệu , được thiết kế để lưu trữ lượng lớn dữ liệu có cấu trúc, không cấu trúc và bán cấu trúc từ nhiều nguồn khác nhau. Trong hồ dữ liệu, dữ liệu được lưu trữ trong định dạng nguyên thủy của nó, tức là không cần phải tiến hành việc trình bày hay biến đổi dữ liệu trước khi lưu trữ. Điều này cho phép hồ dữ liệu chứa rất nhiều dữ liệu đa dạng, bao gồm dữ liệu từ các ứng dụng kinh doanh, thiết bị IoT, trang web, mạng xã hội, tập tin văn bản, hình ảnh, video, và nhiều nguồn dữ liệu khác.

Trong hồ dữ liệu, dữ liệu không được sắp xếp hoặc xác định trước lược đồ. Thay vào đó, các lược đồ và cấu trúc dữ liệu được xác định khi dữ liệu được truy xuất hoặc yêu cầu (schema-on-read). Điều này cho phép người dùng định

dạng và hiểu cấu trúc dữ liệu dễ dàng theo nhu cầu thực tế của họ, giúp linh hoạt và tiết kiệm thời gian khi phân tích dữ liệu.

Hồ dữ liệu thường được sử dụng trong các ứng dụng phân tích dữ liệu lớn, học máy, phân tích thời gian thực và phân tích đa nguồn, cho phép các tổ chức khai thác dữ liệu một cách hiệu quả và tìm ra thông tin giá trị từ lượng lớn dữ liệu đa dạng.



Hình 2.3: Kiến trúc tổng quan của hồ dữ liệu (Nguồn: [1]).

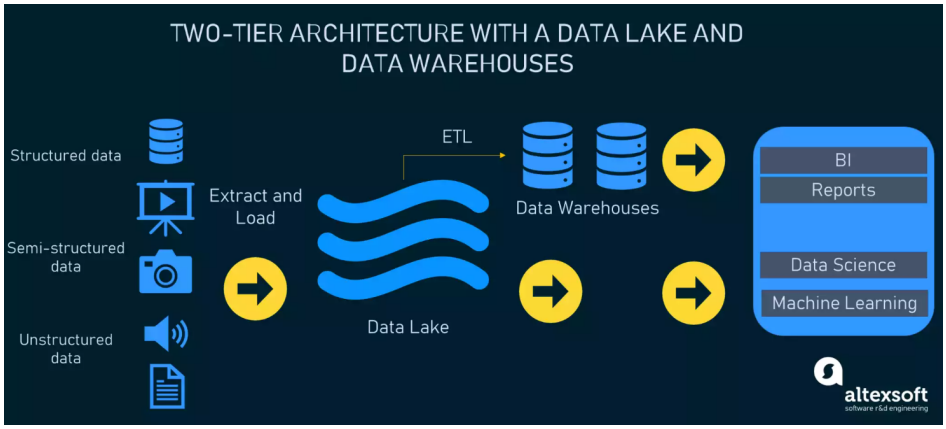
Tuy vậy, hồ dữ liệu có những hạn chế sau:

- Thiếu lược đồ hoặc metadata mô tả có thể làm cho dữ liệu truy vấn.
- Thiếu tính nhất quán về ý nghĩa của dữ liệu có thể làm cho việc phân tích trên dữ liệu trở nên khó khăn.
- Khó đảm bảo chất lượng dữ liệu trước khi đưa vào hồ dữ liệu.
- Thiếu quản lý chính sách, việc kiểm soát truy cập và các vấn đề về quyền riêng tư có thể gây ra vấn đề. Thông tin nào được lưu vào hồ dữ liệu, ai có thể truy cập dữ liệu đó và dùng cho mục đích gì?
- Hồ dữ liệu có thể không phải là cách tốt nhất để tích hợp dữ liệu đã có cấu trúc.

- Một mình hồ dữ liệu không cung cấp cái nhìn tích hợp hoặc toàn diện về tổ chức.
- Hồ dữ liệu có thể trở thành một nơi đổ dữ liệu không bao giờ được phân tích hoặc khai thác để tìm thông tin giá trị.

2.2.3 Kiến trúc dữ liệu hai tầng

Kiến trúc dữ liệu hai tầng nhằm kết hợp được những ưu điểm của kho dữ liệu, kho dữ liệu và giải quyết được những được những nhược điểm của nó. . Nó bao gồm hai thành phần chính: hồ dữ liệu và kho dữ liệu . Mỗi thành phần có chức năng và mục tiêu riêng, nhưng khi hoạt động cùng nhau, chúng tạo ra một hệ thống mạnh mẽ để xử lý và phân tích dữ liệu. Kiến trúc tổng quan dữ liệu hai tầng như sau:



Hình 2.4: Kiến trúc tổng quan dữ liệu hai tầng (Nguồn: [1]).

Hồ dữ liệu là nơi tổ chức lưu trữ các tập dữ liệu lớn, có cấu trúc, không cấu trúc và bán cấu trúc. Dữ liệu được lưu trữ trong hồ dữ liệu dưới dạng nguyên bản, mà không cần phải tiến hành biến đổi hoặc cấu trúc trước. Điều này cho phép lưu trữ linh hoạt và giữ cho dữ liệu không bị mất thông tin, dù cho có thể chưa biết trước mục đích cụ thể của việc sử dụng dữ liệu này. Hồ dữ liệu cũng hỗ trợ các loại phân tích đa dạng, bao gồm xử lý dữ liệu lớn, phân tích thời gian

thực, truy vấn SQL, và các tác vụ học máy.

Kho dữ liệu, mặt khác, là một cơ sở dữ liệu tối ưu hóa cho phân tích dữ liệu quan hệ từ các hệ thống giao dịch và ứng dụng kinh doanh. Dữ liệu trong kho dữ liệu đã được biến đổi và tối ưu hoá để hỗ trợ truy vấn SQL nhanh chóng, đáp ứng nhu cầu của việc thực hiện các báo cáo và phân tích hoạt động trong tổ chức. Kho dữ liệu thường được xây dựng dựa trên một lược đồ dữ liệu đã được xác định trước, giúp đảm bảo tính nhất quán và tin cậy của dữ liệu khi sử dụng để đưa ra các quyết định kinh doanh.

Mối quan hệ giữa hồ dữ liệu và kho dữ liệu trong kiến trúc hai tầng cho phép tổ chức tận dụng tốt cả hai loại dữ liệu: dữ liệu nguyên bản và dữ liệu đã được tiền xử lý. Dữ liệu nguyên bản được lưu trữ trong hồ dữ liệu, cho phép các tác vụ khám phá dữ liệu và phân tích đa dạng. Khi dữ liệu được xác định cụ thể và cần phải được truy vấn và phân tích một cách nhanh chóng và tin cậy, dữ liệu này được sao chép và biến đổi thành cấu trúc thích hợp để lưu trữ trong kho dữ liệu.

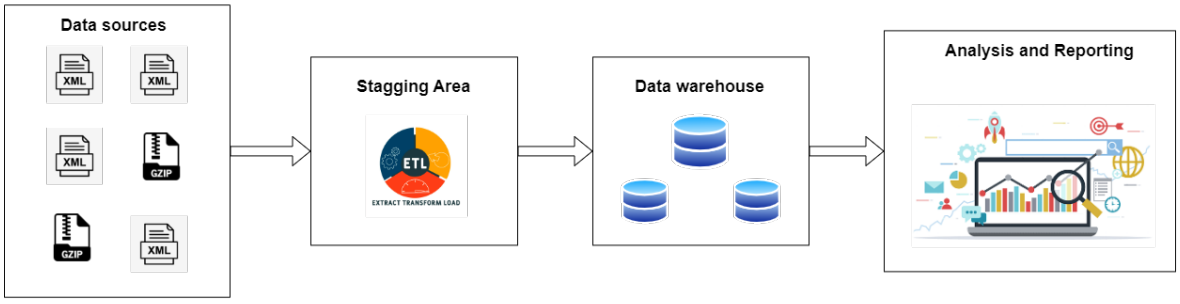
Chương 3

Khảo sát yêu cầu và phân tích

3.1 Khảo sát hệ thống

Các máy chủ EMS cung cấp các file raw counter với định dạng, tần xuất khác nhau. Đối với hãng Nokia, thì cứ sau mỗi chu kỳ 15 phút máy chủ EMS lại xuất ra file raw counter của các trạm với định dạng file XML dạng nén. Tổng dung lượng file raw counter hãng Nokia chứa dữ liệu counter của trạm cần thu thập, xử lý trong một ngày là gần 1 TB.

Hệ thống hiện tại đang sử dụng được thiết kế theo kiến trúc như sau



Hình 3.1: Kiến trúc hệ thống hiện tại.

Với kiến trúc hệ thống cũ, các file raw counter được đổ vào vùng đệm (Staging Area) sẽ có bước bóc tách riêng để trích xuất ra các thông tin counter để thực hiện quá trình tổng hợp dữ liệu và tính toán KPI, sau đó đưa dữ liệu KPI

vào vào kho dữ liệu để phục vụ cho việc phân tích và đưa ra các báo cáo thống kê.

Tuy nhiên, dữ liệu raw counter thu thập từ nguồn dữ liệu về không được lưu trữ lâu dài, sau khi tính toán đưa vào kho dữ liệu xong sẽ bị xóa đi trong vùng đệm. Điều này mang đến một vấn đề dữ liệu counter sẽ mất không truy xuất lại sau khi đưa dữ liệu KPI vào kho dữ liệu dẫn đến bất cập khi nhà mạng VNPT muốn khai thác một KPI mới thì dữ liệu raw counter không có sẵn sàng để sử dụng.

Ngoài ra, dữ liệu trong kho dữ liệu được lưu trữ trên cơ sở dữ liệu Oracle, khi lượng dữ liệu lưu trữ ngày càng tăng lên dẫn đến việc truy vấn và xử lý dữ liệu có thể trở nên chậm hơn do cần thực hiện các thao tác phức tạp trên dữ liệu lớn. Ngoài ra, lưu trữ một lượng lớn dữ liệu trong cơ sở dữ liệu Oracle có thể yêu cầu phần cứng và hạ tầng mạnh mẽ hơn, điều này dẫn đến tăng chi phí đầu tư vào hệ thống.

3.2 Mô hình nghiệp vụ

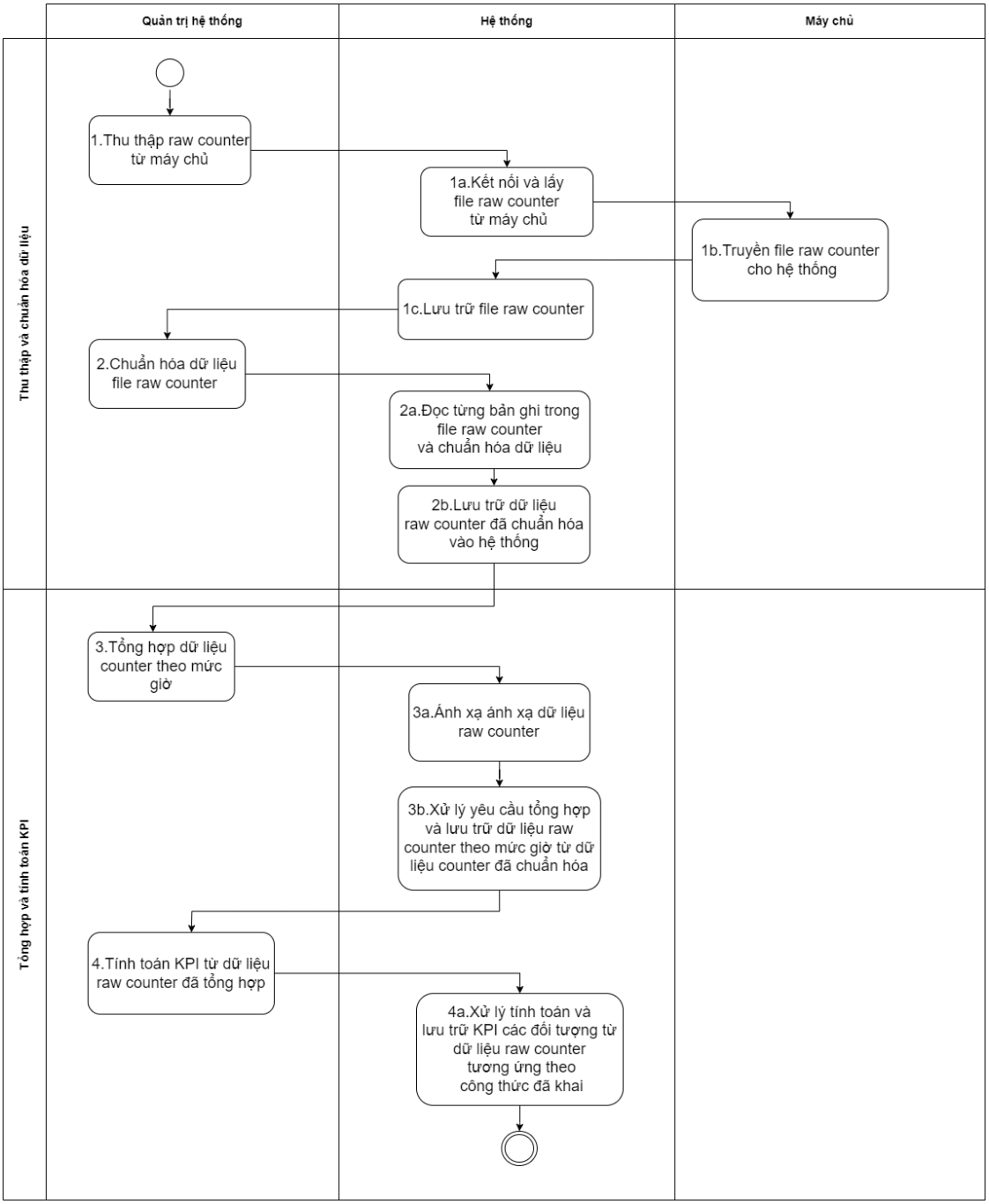
Mô hình nghiệp vụ gồm có 3 quy trình:

- Thu thập, xử lý dữ liệu raw counter và tính toán tổng hợp dữ liệu KPI.
- Quản lý cấu trúc metadata của file raw counter.
- Báo cáo thống kê.

Chi tiết các quy trình nghiệp vụ được mô tả sau đây.

Thu thập, xử lý dữ liệu raw counter và tính toán tổng hợp dữ liệu KPI:

Quy trình xử lý:



Hình 3.2: Quy trình thu thập, xử lý dữ liệu raw counter và tổng hợp tính toán KPI.

Mô tả quy trình:

- **Bước 1:** Quản trị hệ thống thực hiện thu thập và lưu trữ dữ liệu raw counter từ máy chủ.
  - **Bước 1a**: Hệ thống thực hiện:
    + Kết nối đến máy chủ tương ứng.
    + Lấy danh sách file dữ liệu và lọc ra những file mới xuất hiện trong thư mục được cấu hình.
    + Lấy các file dữ liệu mới xuất hiện.
  - **Bước 1b**: Hệ thống máy chủ truyền file dữ liệu raw counter về cho hệ thống.
  - **Bước 1c**: Hệ thống lưu trữ file dữ liệu được truyền về vào hệ thống lưu trữ theo cấu hình. Phục vụ cho các bước xử lý tiếp theo và sao lưu dữ liệu.
- **Bước 2**: Quản trị hệ thống thực hiện chuẩn hóa dữ liệu raw counter theo metadata.
  - **Bước 2a**: Hệ thống thực hiện:
    + Đọc từng bản ghi trong file raw counter đã thu thập.
    + Chuẩn hóa dữ liệu raw counter theo metadata đã cấu hình tương ứng với từng loại đối tượng, từng hãng, từng công nghệ.
  - **Bước 2b**: Hệ thống lưu trữ dữ liệu raw counter đã chuẩn hóa vào hệ thống theo cấu hình đặt trước.
- **Bước 3**: Quản trị hệ thống thực hiện tổng hợp dữ liệu counter trạm hàng giờ từ dữ liệu raw counter đã thu thập và chuẩn hóa từng vendor, từng công nghệ.
  - **Bước 3a**: Hệ thống thực hiện ánh xạ đối tượng trạm, trạm cho các bản ghi raw counter.

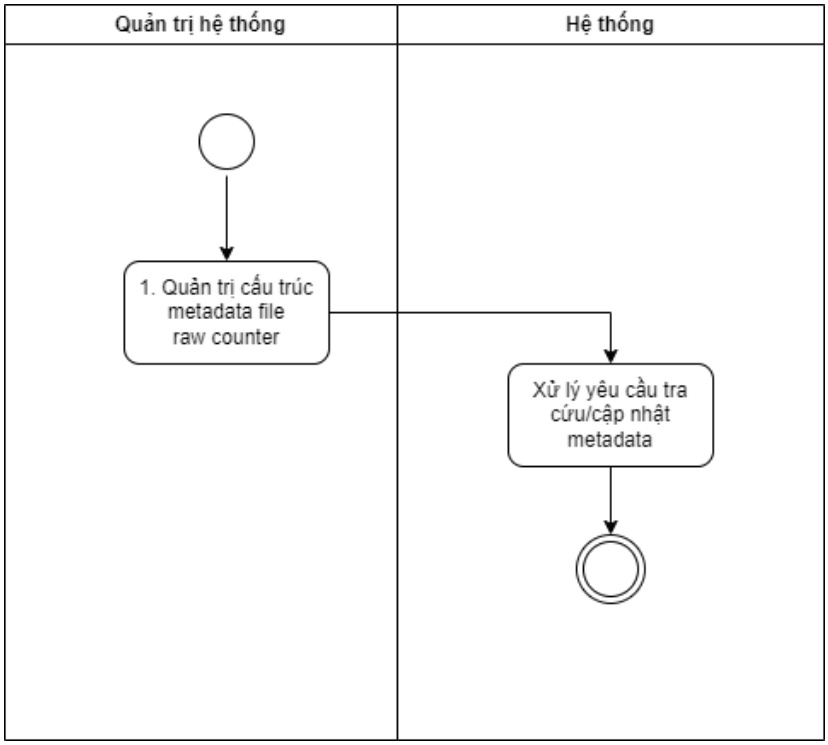
– **Bước 3b**: Hệ thống xử lý yêu cầu tổng hợp và lưu trữ dữ liệu counter mức giờ từ dữ liệu raw counter đã thu thập và chuẩn hóa của trạm.

- **Bước 4**: Quản trị hệ thống thực hiện tính toán KPI từ dữ liệu counter đã tổng hợp tương ứng.

  – **Bước 4a**: Hệ thống xử lý yêu cầu tính toán KPI từ dữ liệu counter đã tổng hợp tương ứng theo công thức đã khai báo.

Quản lý cấu trúc metadata file raw counter:

Quy trình xử lý:



Hình 3.3: Quy trình quản lý cấu trúc metadata của file raw counter

Mô tả quy trình:

- **Bước 1**: Quản trị hệ thống thực hiện các công việc cấu hình, quản lý cấu trúc metadata của file raw counter khi có nhu cầu thay đổi theo các nâng cấp thay đổi từ phía máy chủ.

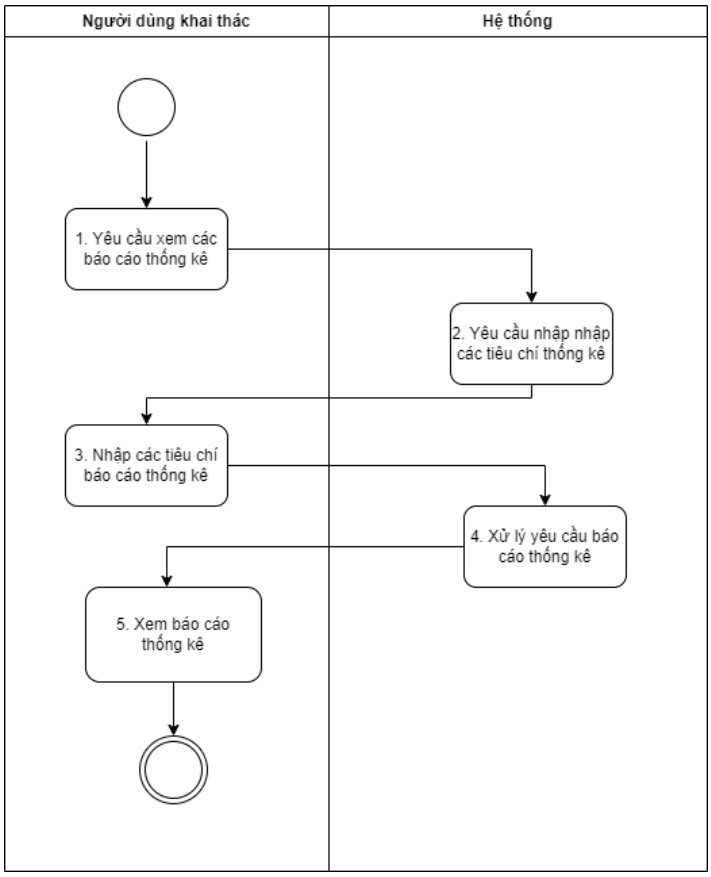
- **Bước 2**: Hệ thống thực hiện xử lý các yêu cầu liên quan:

  + Tra cứu metadata.

  + Cập nhật metadata.

Quy trình báo cáo thống kê:

Quy trình xử lý:



Hình 3.4: Quy trình báo cáo, thống kê.

Mô tả quy trình:

- **Bước 1**: Người dùng khai thác yêu cầu xem các báo cáo, thống kế.
- **Bước 2**: Hệ thống yêu cầu người dùng chọn các tiêu chí báo cáo, thống kê.
- **Bước 3**: Sau khi chọn tiêu chí báo cáo thống kê, tiến hành yêu cầu tra cứu báo cáo, thống kê.

- **Bước 4**: Hệ thống tổng hợp báo cáo, thống kê theo yêu cầu được gửi đến.
- **Bước 5**: Người dùng khai thác xem các kết quả báo cáo, thống kê hiển thị trên màn hình ứng dụng.

3.3 Các yêu cầu chức năng

Các yêu cầu chức năng của hệ thống như sau:

- Thu thập file raw counter trạm.
- Chuẩn hóa dữ liệu raw counter trạm.
- Tổng hợp dữ liệu counter trạm mức giờ từ raw counter trạm.
- Tính toán KPI trạm ở mức giờ.
- Xem thống kê số liệu trung bình KPI của đối tượng trạm.
- Xem bản đồ trạm.

3.4 Các yêu cầu phi chức năng

Lưu trữ dữ liệu

- Khả năng mở rộng: Hệ thống phân tán phải có khả năng mở rộng để có thể xử lý một lượng lớn dữ liệu và đáp ứng với việc gia tăng tải công việc.
- Độ tin cậy: Hệ thống phân tán nên có khả năng chống chịu lỗi và phục hồi sau khi xảy ra sự cố. Dữ liệu được sao lưu hoặc phân tán để đảm bảo tính sẵn sàng và không mất mát dữ liệu quan trọng.
- Hiệu suất: Hệ thống phân tán cần đạt được hiệu suất cao trong việc truy cập và xử lý dữ liệu. Điều này bao gồm tốc độ truy cập dữ liệu, thời gian đáp ứng và khả năng xử lý các yêu cầu cùng lúc.

- Tính nhất quán: Hệ thống phân tán cần đảm bảo tính nhất quán của dữ liệu, tức là dữ liệu truy cập từ các nút khác nhau phải cho kết quả nhất quán.
- Quản lý dữ liệu: Hệ thống phân tán nên cung cấp các công cụ và giao diện để quản lý, giám sát và điều khiển dữ liệu trên các nút trong hệ thống.
- Tích hợp: Hệ thống phân tán cần có khả năng tích hợp với các công nghệ và ứng dụng khác trong môi trường hệ thống tổng thể.

Thời gian xử lý và đảm bảo thông lượng

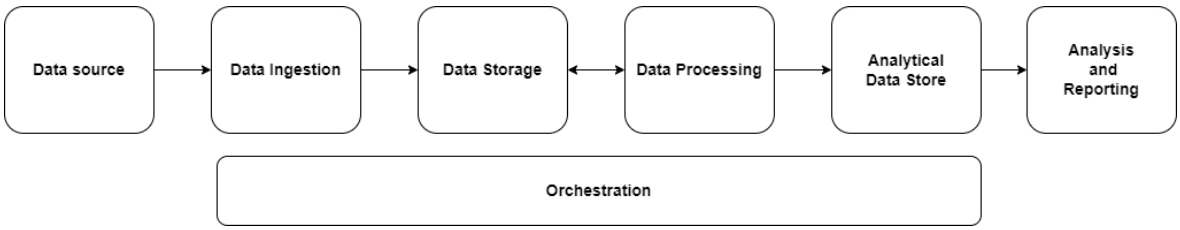
- Dữ liệu KPI mức giờ được tính toán, xử lý không quá 1 giờ sau khi có file raw counter trên máy chủ.
- Hệ thống đảm bảo đáp ứng sử dụng đồng thời 200 người dùng.
- Thời gian phản hồi nhanh.

Chương 4

Kiến trúc hệ thống

4.1 Kiến trúc tổng quan

Để khắc phục những nhược điểm của hệ thống hiện tại, kiến trúc hệ thống mới được đề xuất dựa kiến trúc dữ liệu hai tầng được trình bày ở phần 2.2.3. Tổng quan kiến trúc sẽ có các thành phần như sau:



Hình 4.1: Kiến trúc hệ thống dữ liệu lớn.

- **Nguồn dữ liệu (Data Sources)**: Nguồn dữ liệu, mỗi hệ thống dữ liệu lớn có thể lấy dữ liệu từ một hoặc nhiều nguồn khác nhau, dữ liệu có thể có định dạng có cấu trúc, bán cấu trúc hoặc phi cấu trúc. Một số nguồn dữ liệu tiêu biểu có thể kể đến như các cơ sở dữ liệu quan hệ, các tệp tin như CSV, JSON, XML...
- **Thu thập dữ liệu (Data Ingestion)**: Quá trình thu thập dữ liệu từ các nguồn và chuyển đổi nó thành các lô dữ liệu để truyền đi vào các khoảng

thời gian được xác định trước. Lớp xử lý dữ liệu theo lô có thể sử dụng lịch trình đơn giản, các sự kiện kích hoạt hoặc bất kỳ trình tự logic nào khác để thu thập dữ liệu. Khi doanh nghiệp cần thu thập các điểm dữ liệu cụ thể hàng ngày hoặc không cần dữ liệu để đưa ra quyết định theo thời gian thực, quá trình thu thập dựa trên lô trở nên hữu ích.

- **Lưu trữ dữ liệu (Data Storage)**: Lưu trữ dữ liệu, trong mô hình xử lý dữ liệu theo lô (batch processing), dữ liệu từ các nguồn dữ liệu sẽ được đẩy vào một nơi lưu trữ dữ liệu chung thường được gọi là một hồ dữ liệu (data lake). Khối lượng dữ liệu được đẩy vào trong hồ dữ liệu sẽ là rất lớn với đa dạng các định dạng dữ liệu. Dữ liệu lớn kết hợp với việc tạo các bản sao của file dữ liệu (replica) để đảm bảo không bị mất dữ liệu trong trường hợp hệ thống gặp lỗi khiến cho dữ liệu đã lớn lại càng lớn. Do đó, hồ dữ liệu thường được thiết kế lưu trữ phân tán file dữ liệu trên các cụm (cluster) nhiều máy tính để tăng khả năng lưu trữ dữ liệu và giảm chi phi nâng cấp thiết bị phần cứng.
- **Xử lý dữ liệu (Data Processing)**: Quá trình xử lý dữ liệu theo từng lô hoặc nhóm dữ liệu. Dữ liệu được nhập vào hệ thống và xử lý một cách tuần tự hoặc song song. Quá trình này thích hợp khi không cần phản hồi và xử lý dữ liệu theo thời gian thực. Sau khi dữ liệu được nhập, nó được chuyển đổi, tiền xử lý và phân tích để trích xuất thông tin.
- **Kho dữ liệu phân tích (Analytical Data Store)**: Lưu trữ dữ liệu phân tích, dữ liệu sau khi được xử lý sẽ được lưu lại dưới dạng có cấu trúc trong OLAP để phục vụ cho việc phân tích và báo cáo.
- **Phân tích và báo cáo (Analysis and Reporting)**: Phân tích và báo cáo, thành phần này nhận trách nhiệm trực quan hóa, phân tích, kết xuất dash-board,... từ dữ liệu trong kho dữ liệu

- **Điều phối (Orchestration)**: Quá trình quản lý và tổ chức các công việc xử lý dữ liệu theo lô. Nó đảm bảo các công việc được thực hiện theo thứ tự và đồng bộ, xác định sự phụ thuộc và quản lý tài nguyên. Thành phần điều phốigiúp tự động hóa quy trình xử lý bằng cách sử dụng các công cụ và hệ thống quản lý công việc. Thành phần điều phốicung cấp tính linh hoạt, đơn giản hóa và tính nhất quán cho việc xử lý dữ liệu theo lô, giúp tăng hiệu suất và sự tự động hóa trong quá trình xử lý.

Hệ thống mong muốn sẽ cần phải giải quyết các vấn đề về thu thập, xử lý, lưu trữ và phân tích dữ liệu lớn. Hệ thống bắt bắt từ nguồn dữ liệu nơi lưu trữ dữ liệu thu thập từ các nguồn khác nhau. Dữ liệu trong nguồn dữ liệu được thu thập và chuẩn hóa bằng thành phần thu thập dữ liệu và được đẩy vào thành phần lưu trữ dữ liệu.

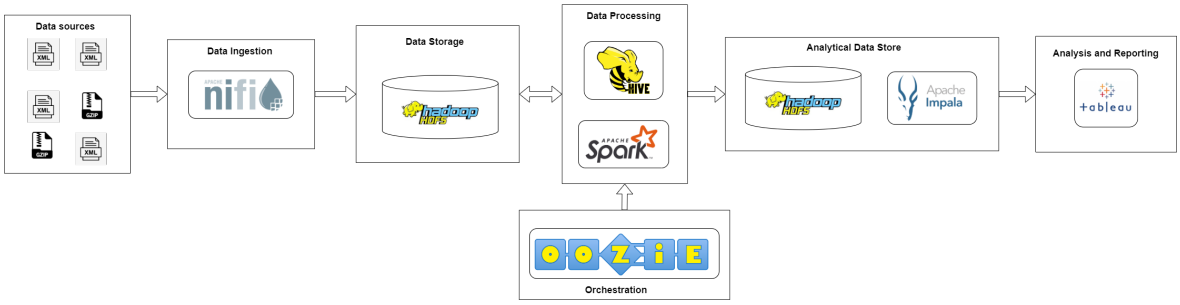
Sau đó, dữ liệu được chuyển đến thành phần xử lý dữ liệu, nơi mà dữ liệu trải qua các bước biến đổi, tính toán tổng hợp và trích xuất thông hữu ích. Quá trình này giúp tạo ra dữ liệu đã được xử lý và sẵn sàng cho việc lưu trữ.

Thành phần xử lý dữ liệu được điều phối bởi thành phần điều phối đảm bảo cho sự đồng bộ, sắp xếp đúng thứ tự công việc và tính nhất quán của quá trình xử lý theo lô.

Dữ liệu sau khi được xử lý được lưu trữ kho dữ liệu phân tích, quản lý dữ liệu được tối ưu hóa cho việc truy vấn và phân tích. Kho dữ liệu phân tích cung cấp cấu trúc và tổ chức dữ liệu, giúp tạo ra các báo cáo, biểu đồ, thông tin hữu ích.

Dữ liệu sau khi được tổ chức trong trong kho dữ liệu phân tích sẽ được trực quan và tương tác, hiển thị thông tin một cách trực quan và dễ hiểu.

Như vậy, em đã trình bày cái nhìn tổng quan về kiến trúc của hệ thống và các thành phần của nó. Trong phần tiếp theo, em sẽ trình bày về công nghệ áp dụng cho từng thành của hệ thống. Chi tiết về thiết kế hệ thống được cho như hình dưới đây:



Hình 4.2: Kiến trúc chi tiết hệ thống.

4.2 Nguồn dữ liệu

Về cơ bản, có thể hiểu nguồn dữ liệu là nơi mà dữ liệu trong hệ thống được sinh ra. Dữ liệu có thể đa dạng về mặt cấu trúc, từ dữ liệu có cấu trúc (cơ sở dữ liệu quan hệ), đến dữ liệu bán cấu trúc (file XML, JSON,...) và cả dữ liệu phi cấu trúc (văn bản, video,...).

Nguồn dữ liệu trong đồ án là máy chủ EMS chịu trách nhiệm quản lý và điều khiển các thiết bị mạng và các thành phần liên quan từ các nhà cung cấp khác nhau.

Trong đồ án này, ta sẽ có nguồn dữ liệu raw counter của nhà hãng Nokia định dạng file XML. Cứ mỗi 15 phút trạm sẽ xuất ra dữ liệu dạng file XML được lưu trữ trong máy chủ EMS. Dữ liệu trong máy chủ EMS được thu thập bằng công cụ Nifi thông qua giao thức FTP hoặc SFTP.

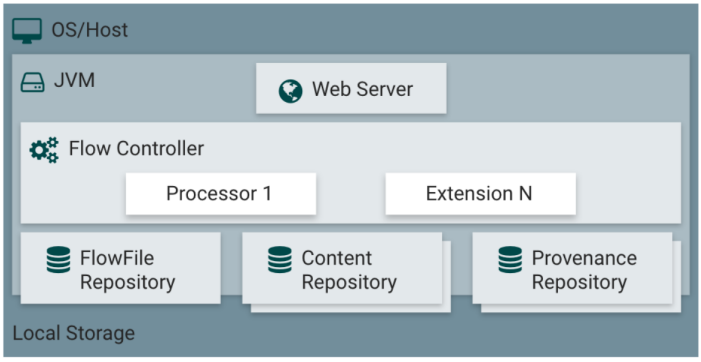
4.3 Thu thập dữ liệu

Như đã đề cập của phần 4.1, đây là phần trung gian giữa các nguồn dữ liệu với thành phần lưu trữ dữ liệu. Nó đóng vai trò thu thập, chuẩn hóa dữ liệu từ nguồn dữ liệu. Việc thu thập, chuẩn hóa dữ liệu được thu thập bởi công cụ Apache Nifi. Tổng quan về Nifi như sau:

Apahe Nifi là một công cụ mã nguồn mở được sử dụng để tự động hóa và

kiểm soát các luồng thu thập dữ liệu. Nó cung cấp giao diện người dùng trực quan và dễ sử dụng giúp người dùng thiết kế và triển khai một cách dễ dàng. Ngoài ra Nifi cung cấp theo dõi xuất xứ dữ liệu (data provenance tracking), cho phép người dùng theo dõi các sự kiện và hoạt động liên quan đến dữ liệu.

Kiến trúc của Nifi được thể hiện như hình (4.3)



Hình 4.3: Kiến trúc Nifi (Nguồn: [2]).

Nifi thực thi trong một JVM (Java Virtual Machine) trên hệ điều hành máy chủ. Nifi bao gồm các thành chính như sau:

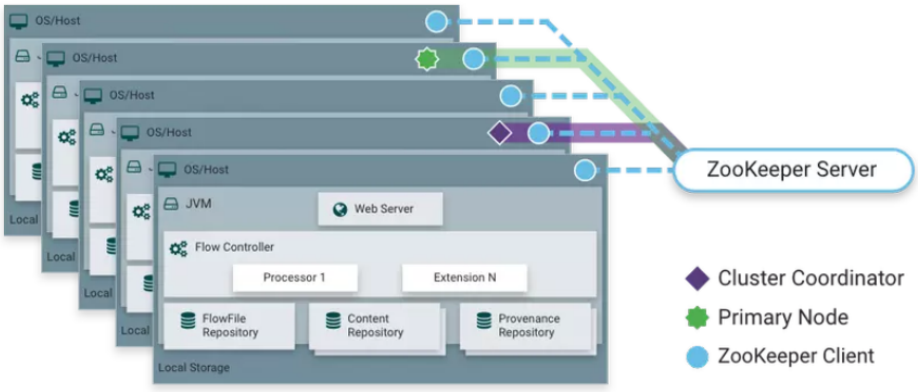
- Web Server: Mục đích của máy chủ web là lưu trữ API điều khiển và lệnh dựa trên HTTP của Nifi.
- Flow Controller: Quản lý và điều khiển luồng dữ liệu. Nó đảm bảo chuyển dữ liệu qua các Processor và Relationship theo quy trình đã được định nghĩa. Flow Controller quản lý tài nguyên, đồng thời kiểm soát việc bắt đầu và kết thúc xử lý dữ liệu, đảm bảo sự liên tục và tin cậy của luồng dữ liệu. Nó cũng quản lý tốc độ xử lý và đồng bộ hóa việc xử lý đồng thời. Flow Controller là trung tâm điều khiển quan trọng trong hệ thống Nifi, đóng góp vào sự linh hoạt và tin cậy của quá trình xử lý dữ liệu
- Flow File: Đơn vị dữ liệu khi di chuyển qua hệ thống. Nó chứa dữ liệu thô và thuộc tính liên quan. Flow file đại diện cho dữ liệu trong quá trình xử lý và chuyển đổi trong luồng dữ liệu. Mỗi Flow File có một ID duy nhất và

có thể chứa bất kỳ loại dữ liệu nào. Thuộc tính cung cấp thông tin về dữ liệu và có thể được sửa đổi bởi các Processor. Flow File được quản lý bởi Flow Controller và di chuyển qua các Processor trong luồng dữ liệu. Nó giúp Nifi xử lý dữ liệu linh hoạt và tự động, theo dõi và quản lý dữ liệu khi di chuyển qua hệ thống.

- Content Repository: Dùng để lưu trữ và quản lý dữ liệu của hệ thống. Nó được sử dụng để lưu trữ Flow File và cung cấp khả năng truy cập và quản lý dữ liệu tạm thời và vĩnh viễn. Content Repository có thể được cấu hình để sử dụng các hình thức lưu trữ khác nhau và đóng vai trò quan trọng trong việc quản lý và truy xuất dữ liệu trong quá trình xử lý luồng dữ liệu của Nifi.

- Provenance Repository: Thành phần lưu trữ và quản lý thông tin về lịch sử và nguồn gốc của dữ liệu. Nó ghi lại các sự kiện quan trọng liên quan đến xử lý, chuyển đổi và di chuyển dữ liệu trong hệ thống. Provenance Repository cho phép truy xuất và tìm kiếm lịch sử dữ liệu, cung cấp thông tin chi tiết về các sự kiện và quá trình xử lý. Nó hỗ trợ lưu trữ dữ liệu lâu dài hoặc tạm thời và cung cấp thông tin về định danh, thời gian, thuộc tính và quan hệ của các sự kiện liên quan đến dữ liệu. Provenance Repository giúp theo dõi, giám sát và quản lý hiệu suất của quá trình xử lý dữ liệu trong Nifi.

Nifi có thể hoạt động theo một cụm:



Hình 4.4: Nifi Cluster (Nguồn: [2]).

Mô hình hoạt động của Nifi Cluster là mô hình hoạt động Zero-Master Clustering. Nó là một mô hình triển khai cụm Nifi mà không có một nút chủ duy nhất. Thay vào đó, mỗi node trong cụm thực hiện các tác vụ và có khả năng chịu lỗi đảm bảo tính sẵn sàng và khả năng mở rộng của hệ thống.
Cụm Nifi bao gồm các thành phần:

- Cluster Coordinator: Là một node trong cụm Nifi chịu trách nhiệm duy trì thông tin về trạng thái của cụm.
- Primary node: Là node chịu trách nhiệm quản lý và điều phối hoạt động của toàn bộ cluster. Nó quản lý tài nguyên, điều phối hoạt động của các node thành viên, và quản lý Metadata và Provenance Repository. Primary Node cũng xử lý sự cố và có khả năng chuyển giao quyền kiểm soát khi cần thiết. Vai trò của Primary Node đảm bảo tính tin cậy và khả năng mở rộng của hệ thống.
- Mỗi node trong cụm Nifi thực hiện các tác vụ tương tự trên dữ liệu, nhưng mỗi node hoạt động độc lập trên một tập dữ liệu riêng biệt.

Apache Nifi có khả năng quản lý luồng dữ liệu và sử dụng dễ dàng:

- Đảm bảo an toàn dữ liệu: Nifi sử dụng đối tượng FlowFile để biểu diễn mỗi đơn vị dữ liệu trong luồng. FlowFile ghi lại thông tin về dữ liệu đang được xử lý bởi các khối và được chuyển đến đâu. Lịch sử xử lý của một FlowFile được lưu trữ trong Provenance Repository, cho phép việc truy vết dữ liệu. Cơ chế Copy-on-Write giúp Nifi lưu trữ dữ liệu trước khi xử lý, giúp dễ dàng chạy lại dữ liệu khi cần.
- Data Buffering: Tính năng này giúp giải quyết vấn đề tốc độ ăn chậm hơn tốc độ nhả giữa hai hệ thống khác nhau. Nó hoạt động dựa theo cơ chế Queue giữa hai khối xử lý trong luồng. Dữ liệu này sẽ được giữ trên RAM, nhưng nếu nó vượt qua ngưỡng mình cài thì dữ liệu sẽ được đưa xuống ổ cứng.

Việc sử dụng Nifi cũng rất dễ dàng nhờ vào các tính năng:

- Tạo luồng dữ liệu: Việc tạo luồng dữ liệu được thực hiện hoàn toàn trên giao diện web, bạn có thể kéo và thả các khối để tạo ra một luồng dữ liệu đơn giản.
- Tính tái sử dụng: Nifi hỗ trợ tái sử dụng luồng dữ liệu thông qua việc tạo mẫu (template). Có thể tạo một mẫu chứa một luồng cơ bản và sử dụng lại nó khi cần

Ngoài ra, Apache Nifi cũng có những khả năng và tính năng khác, bao gồm:

- Kết nối với máy chủ : Nifi hỗ trợ kết nối và truyền dữ liệu qua giao thức FTP và SFTP. Điều này cho phép thu thập dữ liệu từ các máy chủ thông qua FTP và SFTP gửi dữ liệu đến các máy chủ khác.
- Khả năng mở rộng: Mỗi processor trong Nifi là một đơn vị xử lý dữ liệu độc lập, thực hiện một công việc cụ thể trên dữ liệu khi chúng đi qua hệ thống. Processor có thể thực hiện các hoạt động như trích xuất, biến đổi,

lọc, mã hóa, giải mã, kiểm tra lỗi, ghi log và nhiều tác vụ xử lý dữ liệu khác.

Với khả năng tùy chỉnh và mở rộng, người dùng có thể viết mã Java để tạo ra các processor tùy chỉnh hoặc sử dụng công cụ ExecuteScript processor để viết mã xử lý dữ liệu bằng các ngôn ngữ như Groovy, Python, JavaScript. Điều này cho phép Nifi linh hoạt và đáp ứng được các yêu cầu xử lý dữ liệu đặc biệt và phức tạp.

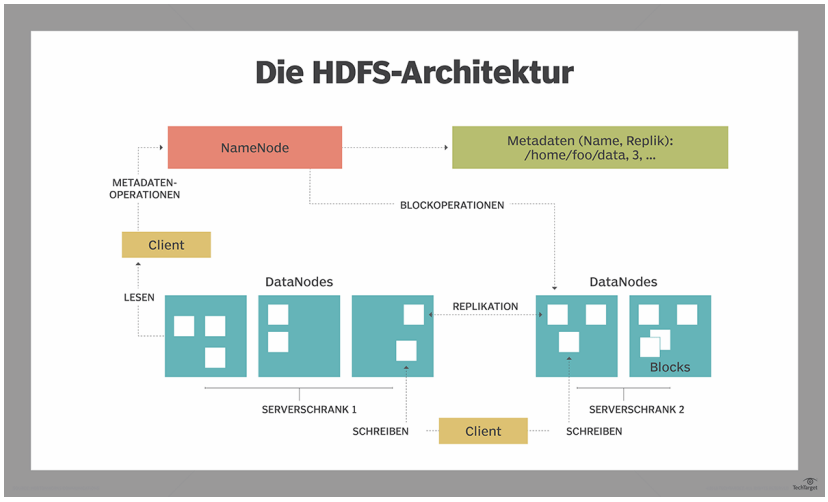
4.4 Lưu trữ dữ liệu

Có hai loại dữ liệu được thu thập bởi thành phần thu thập dữ liệu đó là file raw gốc ban đầu và dữ liệu của file raw được chuẩn hóa. File raw gốc ban đầu lưu trữ trên HDFS để sao lưu, phục vụ cho việc tra cứu dữ liệu khi gặp sự cố mà không cần lấy lại file từ nguồn dữ liệu. Dữ liệu file raw được chuẩn hóa để dễ dàng cho việc tính toán tổng hợp của thành phần xử lý dữ liệu. Tổng quan về công nghệ lưu trữ HDFS như sau:

HDFS là một hệ thống tệp phân tán được thiết kế để lưu trữ và xử lý dữ liệu lớn trên một cụm máy tính. Nó là một phần quan trọng của nền tảng Apache Hadoop, một hệ sinh thái phần mềm mã nguồn mở được sử dụng rộng rãi để xử lý và phân tích dữ liệu lớn.

HDFS chia nhỏ dữ liệu thành các khối nhỏ hơn (block) và lưu trữ chúng trên nhiều máy tính trong một cụm Hadoop. Dữ liệu được phân tán trên các node trong cụm, giúp tăng khả năng lưu trữ lớn và xử lý dữ liệu đồng thời. HDFS tự động sao lưu dữ liệu bằng cách lưu nhiều bản sao của mỗi khối dữ liệu trên các node khác nhau. Việc sao lưu dữ liệu giúp đảm bảo tính sẵn có và độ tin cậy của dữ liệu, ngăn ngừa mất mát dữ liệu trong trường hợp node hoặc đĩa bị hỏng. HDFS hỗ trợ việc đọc và ghi dữ liệu tuần tự, tập trung vào việc lưu trữ và xử lý các tệp dữ liệu lớn. Điều này cho phép xử lý dữ liệu theo các khối liên tiếp và

hiệu quả, phù hợp với kiểu xử lý dữ liệu phân tán.



Hình 4.5: Kiến trúc HDFS (Nguồn: [5]).

HDFS sử dụng mô hình kiến trúc master-slave. Cụ thể, nó có một namenode (master) quản lý metadata và nhiều datanode (slave) lưu trữ dữ liệu thực tế. Namenode theo dõi vị trí và thông tin về các khối dữ liệu, trong khi datanode chịu trách nhiệm lưu trữ và quản lý các khối dữ liệu. HDFS có khả năng mở rộng tốt, cho phép mở rộng số lượng node và dung lượng lưu trữ dựa trên nhu cầu của ứng dụng. Điều này cho phép xử lý dữ liệu lớn và đáp ứng tải công việc ngày càng tăng. HDFS có tính kháng lỗi cao. Với việc lưu trữ nhiều bản sao của dữ liệu và quản lý sao lưu tự động, nó giúp đảm bảo tính sẵn có và tin cậy của hệ thống, ngay cả khi có sự cố xảy ra trên các node hoặc đĩa. HDFS là một phần quan trọng trong cơ sở hạ tầng của Hadoop và phù hợp với việc xử lý dữ liệu lớn. Nó cung cấp khả năng lưu trữ và truy xuất dữ liệu hiệu quả, cho phép xử lý song song và phân tán trên cụm máy tính phân tán.

4.5 Xử lý dữ liệu

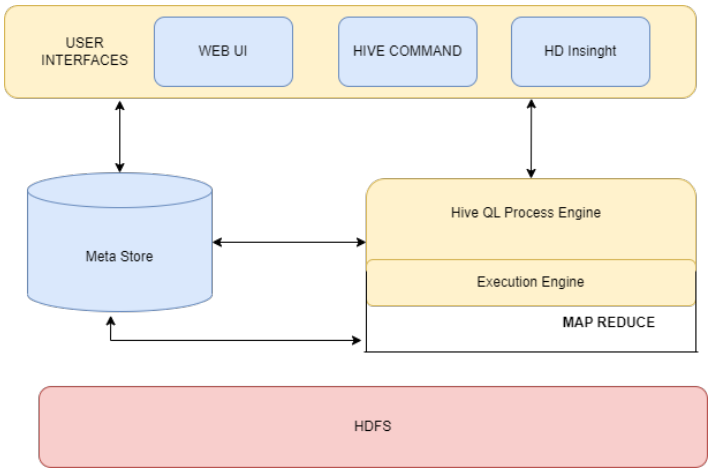
Để dễ dàng cho việc tính toán tổng hợp thì dữ liệu sau khi chuẩn hóa được lưu trong thành phần lưu trữ dữ liệu sẽ được ánh xạ lên các bảng của Hive và sử

dụng nền tảng tính toán phân tán Spark để tổng hợp, tính toán KPI, đáp ứng yêu cầu xử lý theo lô. Tổng quan về công nghệ được sử dụng trong xử lý dữ liệu.

Apache Hive:

Hive là một công cụ cở sở hạ tầng kho dữ liệu để xử lý dữ liệu có cấu trúc trong Hadoop. Nó được triển khai trên nền tảng Hadoop để tổng hợp dữ liệu lớn và đơn giản hóa quá trình truy vấn và phân tích.

Sơ đồ mô tả kiến trúc của Apache Hive



Hình 4.6: Kiến trúc Apache Hive (Nguồn: [10]).

Apache Hive bao gồm các thành phần:

- **User Interface**: Hive là một phần mềm cơ sở hạ tầng kho dữ liệu có thể tạo ra sự tương tác giữa người dùng và HDFS. Các giao diện người dùng mà Hive hỗ trợ là Hive Web UI, Hive command line và Hive HD Insight.
- **Meta Store**: Hive chọn các máy chủ cơ sở dữ liệu tương ứng để lưu trữ lược đồ hoặc metadata của các bảng, cơ sở dữ liệu, các cột trong một bảng, các loại dữ liệu của chúng và ánh xạ HDFS.
- **HiveQL Process Engine**: HiveQL tương tự như SQL để truy vấn thông tin lược đồ trên Metastore. Đây là một trong những thay thế của phương

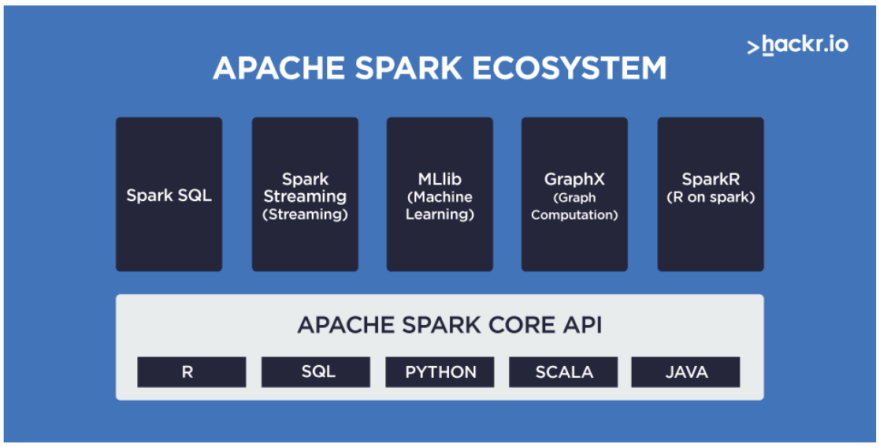
pháp truyền thống cho chương trình MapReduce. Thay vì viết chương trình MapReduce bằng Java thì có thể viết một truy vấn cho công việc MapReduce và xử lý nó.

- **Execution Engine**: Phần kết hợp của công cụ xử lý HiveQL và MapReduce là Công cụ thực thi Hive (Hive Execution Engine). Công cụ thực thi xử lý truy vấn và tạo kết quả giống như kết quả MapReduce.
- **HDFS**: Hệ thống tệp phân tán Hadoop là các kỹ thuật lưu trữ dữ liệu để lưu trữ dữ liệu vào hệ thống tệp.

Hive dễ sử dụng, với giao diện truy vấn dựa trên SQL. Người dùng không chỉ có thể nhanh chóng làm quen với công cụ này mà còn dễ dàng tương tác với dữ liệu và thực hiện các thao tác truy vấn. Hive tích hợp tốt với Hadoop, hệ sinh thái lưu trữ và xử lý dữ liệu lớn. Điều này cho phép người dùng tận dụng sức mạnh của Hadoop, bao gồm HDFS và các công cụ khác, để quản lý và xử lý dữ liệu một cách hiệu quả. Với Hive, người dùng có khả năng xử lý cả dữ liệu có cấu trúc và dữ liệu phi cấu trúc. Điều này mở ra nhiều khả năng cho việc truy vấn, biến đổi và phân tích dữ liệu theo các yêu cầu cụ thể. Hive tối ưu hóa truy vấn, sử dụng bộ tối ưu hóa để cải thiện hiệu suất xử lý. Các kỹ thuật tối ưu hóa, bao gồm chỉ mục và phân vùng, được tự động áp dụng để đảm bảo truy vấn được thực hiện một cách nhanh chóng và hiệu quả.

Apache Spark:

Apache Spark là một phần mềm mã nguồn mở được phát triển để hỗ trợ xử lý song song dữ liệu lớn trên các cụm máy tính phân tán. Spark được phát triển bằng ngôn ngữ lập trình Scala nhưng hỗ trợ rất nhiều ngôn ngữ phổ biến khác nhau (Python, Java, Scala, R,...) cùng với rất nhiều thư viện phục vụ cho các tác vụ khác nhau (truy vấn SQL, xử lý dữ liệu thời gian thực, máy học,...). Các thành phần chính trong hệ sinh thái Spark bao gồm:



Hình 4.7: Hệ sinh thái Spark (Nguồn: [8]).

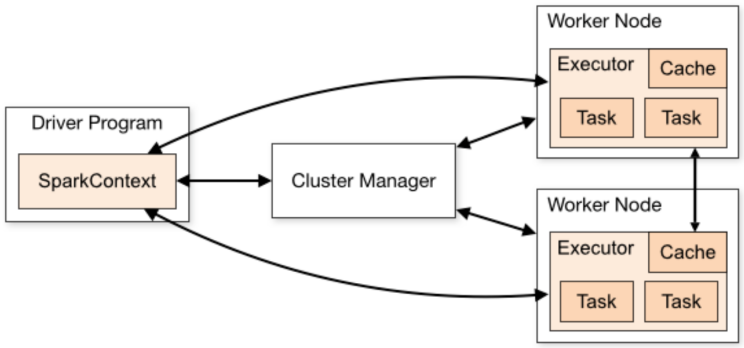
- **Spark Core**: Là nền tảng chung cho tất cả các thành phần của Spark. Cung cấp khả năng xử lý song song trên các cụm phân tán, khả năng tính toán trên bộ nhớ RAM và cả các bộ nhớ ngoài.
- **Spark SQL**: Cung cấp Dataset API, hỗ trợ xử lý dữ liệu có cấu trúc và bán cấu trúc.
- **Spark Streaming**: Gộp dữ liệu streaming thành các mini-batch và thực hiện các RDD transformation để xử lý dữ liệu, qua đó đạt được kết quả near real-time.
- **MLlib**: Tận dụng tốc độ tính toán cao (distributed memory-based) của Spark để thực hiện các tác vụ machine learning.
- **GraphX**: Hỗ trợ tính toán, mô hình hóa các đồ thị do người dùng xác định bằng các API được tối ưu sẵn.

Spark ban đầu được sinh ra để khắc phục các nhược điểm của Hadoop MapReduce, một công cụ lập trình song song rất mạnh mẽ và phổ biến trước đây cho phép xử lý song song dữ liệu trên cụm hàng ngàn server. MapReduce hoạt động dựa trên một chuỗi các thao tác [đọc - xử lý - ghi] dữ liệu. Dữ liệu đầu vào chia thành các block được xử lý song song bởi các hàm map() rồi tổng

hợp lại bằng các hàm reduce(). Vấn đề của MapReduce là mỗi lần [đọc - xử lý - ghi] dữ liệu như vậy, dữ liệu đầu ra lại được ghi vào file system của Hadoop là HDFS tức là được ghi vào ổ đĩa vốn có hiệu suất I/O không cao khiến cho việc xử lý dữ liệu bị chậm.

Để khắc phục nhược điểm của MapReduce, Spark phát triển một API gọi là RDD (Resilient Distributed Dataset) với mỗi RDD hoạt động như một tập dữ liệu phân tán. Điểm khác biệt của RDD là trong suốt quá trình dữ liệu được xử lý, nó chỉ được load lên từ file system trên ổ đĩa duy nhất 1 lần, các phép transformation sẽ không được thực hiện ngay mà sẽ được lên kế hoạch, tối ưu cho đến khi một action được gọi thì kết quả mới được lưu lại lên file system. Toàn bộ quá trình này được thực hiện trên bộ nhớ RAM cộng với việc tối ưu kế hoạch xử lý khiến cho tốc độ xử lý dữ liệu của Spark nhanh hơn MapReduce tới từ 10-100 lần.

Kiến trúc cơ bản của Apache Spark được hiển thị trong hình:



Hình 4.8: Kiến trúc Spark (Nguồn: [9])

Trong kiến trúc của Apache Spark, Driver Program được coi là thành phần chính của ứng dụng và tạo ra SparkContext. Một SparkContext bao gồm tất cả các chức năng cơ bản. Spark Driver bao gồm các thành phần khác nhau như DAG Sheduler, Task Sheduler, ... chịu trách nhiệm biên dịch code do người dùng biên soạn thành các công việc và được thực thi trên các cụm.

Spark Driver và SparkContext cùng giám sát việc thực hiện công việc trong các cum. Spark Driver làm việc với Cluster Manager để quản lý các công việc. Cluster Manager thực hiện việc phân bổ tài nguyên. Sau đó, công việc được chia thành nhiều tác vụ nhỏ hơn và được phân phối đến các Worker Node.
Khi một RDD được tạo trong SparkContext, nó có thể được phân phối trên nhiều Worker Node và cũng có thể được lưu trữ tại đó. Các Worker Node thực thi các tác vụ được giao bởi Cluster Manager và trả về cho Spark Context.

Một executor chịu trách nhiệm thực thi các tác vụ này. Tuổi thọ (lifetime) của các executor giống như thời gian chạy của ứng dung Spark. Nếu muốn tăng hiệu suất của hệ thống thì có thể tăng số lượng các worker để công việc có thể được chia thành các phần logic nhỏ hơn.

Trong đồ án, thành phần SparkSQL được dùng xuyên suốt trong quá trình xử lý dữ liệu. Vì vậy, em sẽ trình bày kỹ hơn về thành phần này:

Được xây dựng dựa trên nền Spark Core, Spark SQL thường được coi là thành phần quan trọng nhất trong hệ sinh thái của Spark. Nó cho phép ta thực hiện các câu lệnh SQL với các tập dữ liệu phân tán, có cấu trúc thông qua Dataset hoặc Dataframe. Thừa hưởng tất cả các tính năng của RDD từ Spark Core kết hợp với cơ chế Catalyst Optimizer khiến cho Spark SQL đạt được hiệu năng rất cao cùng khả năng mở rộng và chịu lỗi tốt. Ngoài ra, do cùng được xây dựng trên nền Spark Core nên Spark SQL tương thích rất tốt với các thành phần khác trong hệ sinh thái Spark, giúp ta xây dựng dựng được các ứng dụng Spark tối ưu, linh hoạt với đa dạng các chức năng.

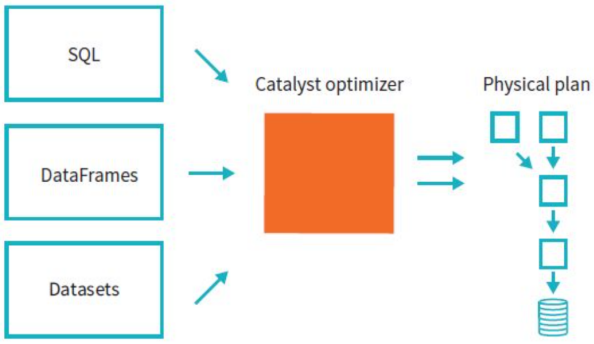
Thành phần SparkSQl có hai kiểu cấu trúc dữ liệu được sử dụng để xử lý và truy vấn:

- Dataframe là một tập dữ liệu phân tán và không thay đổi được, điểm khác là Dataframe được thiết kế tương tự như một bảng trong cơ sở dữ liệu quan hệ, tức là mỗi bản ghi sẽ được lưu dưới dạng một hàng với các cột có kiểu dữ

liệu được định nghĩa trước thông qua schema. Ta có thể tạo một Dataframe từ đa dạng các nguồn khác nhau có thể kể đến như các file CSV, JSON, XML, các hệ quản trị cơ sở dữ liệu quan hệ, NoSQL, Hive, HDFS,...

- Dataset là một cấu trúc dữ liệu kiểu mạnh được giới thiệu trong Spark 1.6, kế thừa từ DataFrame. Nó cung cấp tính năng của DataFrame, bao gồm tối ưu hóa truy vấn dựa trên Catalyst Optimizer, nhưng cũng hỗ trợ việc sử dụng kiểu dữ liệu cấp cao và truy cập dữ liệu qua các phương thức gõ chính xác (type-safe methods). Với Dataset, Spark SQL có thể áp dụng kiểm tra kiểu dữ liệu tại thời điểm biên dịch, giúp phát hiện lỗi logic trước khi chạy chương trình. Dataset kết hợp tính linh hoạt của RDD với tối ưu hóa của DataFrame và được coi là một API dựa trên kiểu dữ liệu mới cho Spark.

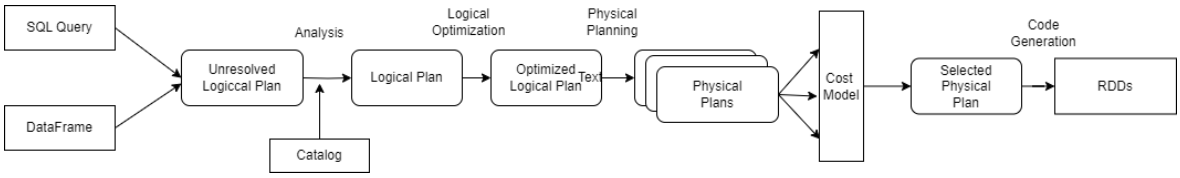
SparkSQL sử dụng cơ chế Catalyst Optimizer để nâng cao hiệu năng cho các ứng dụng Spark. Thông thường, khi xây dựng một ứng dụng, ta sẽ phải viết code để chỉ dẫn cho máy tính phải làm gì. Công đoạn này được thực hiện thủ công nên không thể tránh khỏi việc code của ta chưa tối ưu hoặc còn sót lại những tính toán dư thừa. Đối với những hệ thống dữ liệu lớn lên tới hàng tỷ bản ghi, một phép tính chưa tối ưu hoặc dư thừa có thể sẽ lặp lại đến hàng tỷ lần gây tốn kém về mặt chi phí và thời gian. Cơ chế Catalyst Optimizer được sinh ra để khắc phục vấn đề này. Về cơ bản, Catalyst Optimizer sẽ hoạt động như một thành phần trung gian giữa code mà người dùng submit với những công việc thực sự được thực hiện trên các máy tính trong cụm phân tán. Nó tiến hành đọc code, chỉnh sửa code để ta có được phiên bản code hiệu quả nhất mà vẫn đảm bảo cho ra kết quả tương đương với đoạn code ban đầu:



Hình 4.9: Catalyst Optimizer trong Spark SQL (Nguồn: [7]).

Khi người dùng gửi mã (submit code) cho Spark, mã đó sẽ được chuyển thành "unresolved logical plan". Được gọi là "unresolved" vì có thể xảy ra trường hợp mã của người dùng không gặp lỗi, nhưng có các cột hoặc bảng mà mã tham chiếu tới không tồn tại. Spark sử dụng một bộ sưu tập thông tin về các bảng gọi là "catalog" để phân tích "unresolved logical plan". Nếu có bảng hoặc cột mà mã tham chiếu tới không tồn tại, "unresolved logical plan" sẽ bị từ chối và hệ thống sẽ báo lỗi. Ngược lại, nếu tất cả các tham chiếu trong mã đều hợp lệ, "unresolved logical plan" sẽ chuyển thành "resolved logical plan" sẵn sàng để được tối ưu.

"Resolved logical plan" sau đó sẽ trải qua một loạt các luật được xác định sẵn trong Catalyst Optimizer và trở thành "physical plan". "Physical plan" là một chuỗi các phép biến đổi (transformation) mà sẽ được thực hiện trên các máy tính trong cụm phân tán. Quá trình này nhằm tối ưu hóa việc thực thi và phân phối công việc trên các node xử lý trong hệ thống Spark:



Hình 4.10: Chi tiết các bước trong Catalyst Optimer (Nguồn: [7]).

4.6 Điều phối

Quá trình tổng hợp dữ liệu counter và tính toán KPI từ dữ áp dụng quy tắc biến đổi phức tạp và yêu cầu tích hợp dữ liệu liên tục. Điều này dẫn đến việc phải xử lý các vấn đề như sự phụ thuộc dữ liệu, lịch trình thực hiện và sự đồng bộ giữa các bước. Vì vậy, Điều phối giúp tăng tính linh hoạt, đảm bảo sự nhất quán và tiết kiệm thời gian nhất. Trong đồ án này, công nghệ được em sử dụng là Apache Oozie. Tổng quan về Apache Oozie sẽ được trình bày ngay sau đây:

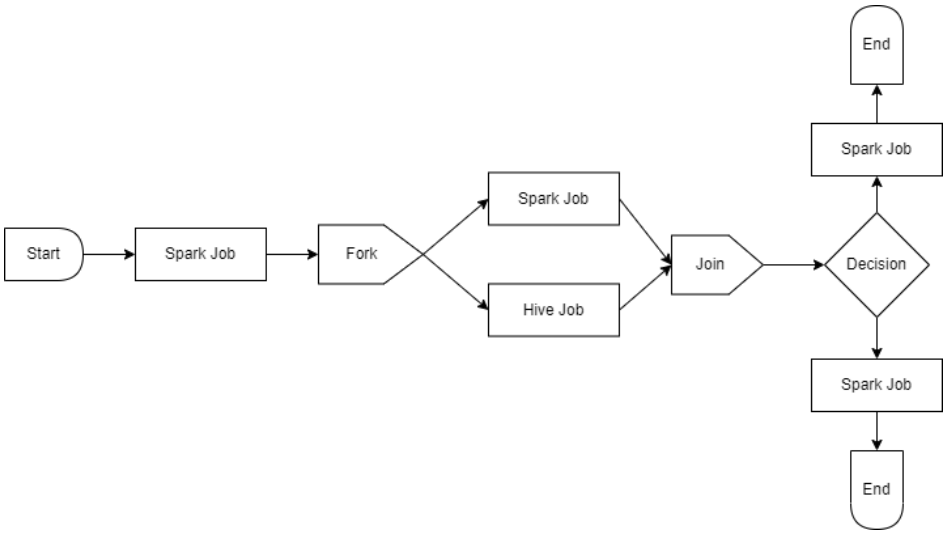
Apache Oozie là một công cụ lập lịch công việc (workflow scheduling) mã nguồn mở, giúp xử lý và tổ chức các nhiệm vụ xử lý dữ liệu trên cơ sở hạ tầng dựa trên Hadoop. Nó cho phép người dùng lập kế hoạch và thực hiện các luồng công việc xử lý dữ liệu phức tạp trong hệ sinh thái Hadoop, xử lý nhiều nhiệm vụ và hoạt động khác nhau. Người dùng Oozie có thể mô tả các phụ thuộc giữa các công việc và hoạt động khác nhau, chỉ định thứ tự mà chúng được thực thi và xử lý các vấn đề và khả năng thử lại. Nó hỗ trợ nhiều công nghệ liên quan đến Hadoop, bao gồm Pig, Hive, Sqoop và Hadoop MapReduce. Oozie cung cấp một API để tương tác với các công cụ và hệ thống khác và giao diện dựa trên web để quản lý và giám sát luồng công việc. Apache Oozie là một công cụ hiệu quả để lập kế hoạch và phối hợp các hoạt động dữ liệu quan trọng trong Hadoop.

Apache Oozie có hai thành phần chính: quản lý luồng công việc Oozie (Oozie Workflow Manager) và điều phối Oozie (Oozie Coordinators).

- **Quản lý luồng công việc Oozie**: Quản lý và thực thi các luồng công việc và chuỗi các hoạt động phải được tuân theo một thứ tự cụ thể.
- **Điều phối Oozie**: Chịu trách nhiệm tổ chức và giám sát các luồng công việc lặp đi lặp lại. Điều phối Oozie mô tả lịch trình để chạy các luồng công việc, dữ liệu đầu vào cho mỗi luồng công việc và sự phục thuộc giữa các

luồng công việc của quá trình. Điều phối Oozie hoạt động theo chu kỳ và lập lịch theo kế hoạch và dữ liệu cung cấp.

Quản lý luồng công việc Oozie và điều phối Oozie kết hợp với nhau để tạo ra luồng xử lý manh mẽ để kiểm soát và thực hiện các luồng công việc phức tạp trong môi trường Hadoop. Ví dụ luồng công việc với các nút điều khiển (Bắt đầu (Start), tách (Fork), quyết định (Decision), kết hợp(Join), kết thúc (End)) với các hoạt động của (Hive, Spark) sẽ có dạng như sơ đồ sau đây:



Hình 4.11: Ví dụ luồng công việc của Oozie

Một luồng công việc của Oozie là một tập hợp các hoạt động được sắp xếp theo một đồ thị có hướng (DAG). Các nút điều khiển xác định thứ tự công việc đặt quy tắc để bắt đầu và kết thúc một luồng công việc. Điều này cho phép Oozie điều khiển thực thi luồng công việc với các nút quyết định, tách và kết hợp . Các nút hoạt động kích hoạt thực thi các tác vụ.

Trong phạm vi đồ án, em thiết kế các tác vụ trong Oozie được tổ chức thành các luồng công việc, thể hiện chuỗi logic hoặc thứ tự các tác vụ cần được thực hiện, qua đó giúp việc giám sát vận hành và đảm bảo thứ tự tổng hợp dữ liệu và tính toán KPI theo yêu cầu. Mỗi tác vụ có thể là một yêu cầu tổng hợp dữ liệu và tính toán thực hiện bằng Spark.

4.7 Kho dữ liệu phân tích

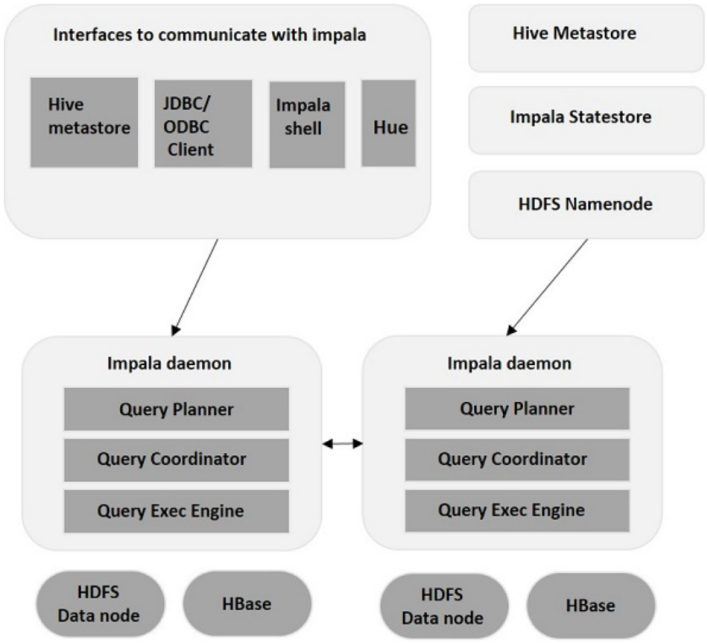
Dữ liệu KPI của các trạm được lưu trữ trong HDFS ánh xạ lên lên các bảng fact và dimension trong Impala để phục vụ cho các yêu cầu truy xuất dữ liệu hiển thị các báo cáo. Tổng quan về công nghệ sử dụng trong phần này:

Apache Impala

Impala là công cụ truy vấn SQL MPP (Massive Parallel Processing) được xử dụng để xử lý lượng dữ liệu lớn được lưu trữ trong cum Hadoop.

Với Impala, người dùng có thể truy xuất HDFS bằng cách sử dụng các truy vấn SQL một cách nhanh chóng hơn so với các công cụ truy vấn SQL khác như Hive. Không giống như Apache Hive, Impala không dựa trên các thuật toán MapReduce. Nó triển khai một kiến trúc phân tán dựa trên các quy trình daemon chịu trách nhiệm cho tất cả các khía cạnh của thực thi truy vấn chạy trên cùng các node. Do đó, Impala giảm độ trễ khi sử dụng MapReduce và điều này làm cho Impala nhanh hơn Apache Hive.

Khác với hệ thống lưu trữ truyền thống, Impala phụ thuộc trực tiếp vào công lưu trữ. Nó có ba thành phần chính là Impala daemon (Impalad), Impala State-store & Impala metadata hoặc metastore.



Hình 4.12: Kiến trúc Apache Impala (Nguồn: [11]).

- **Impala daemon (Impalad)**: Chạy trên mỗi node nơi Impala được cài đặ. Nó chấp nhận các truy vấn từ các giao diện khác nhau. Khi một truy vấn được gửi đến impalad trên một node cụ thể , node đó đóng vai trò là "node điều phối" truy vấn đó. Các truy vấn khác cũng được "phục vụ" bởi các impalad chạy trên các node khác trong cụm. Impalad đọc và ghi dữ liệu từ các tệp và phân tán công việc cho các impalad khác thực thi truy vấn song song. Kết quả các truy vấn từ các impalad khác nhau được trả về node điều phối trung tâm. Tùy thuộc vào yêu cầu, truy cấn có thể được gửi đến một impalad cụ thể hoặc cân bằng tải đến các impalad khác trong cum.
- **Impala State Store**: Có nhiệm vụ kiểm tra tình trạng của mỗi impalad và chuyển tiếp thông tin này cho các Impala daemon trong cụm một cách thường xuyên. Nó có thể chạy trên cùng một node server hoặc trên các node khác trong cụm đang chạy. Quá trình Impala State Store nhận thông báo về tình trạng hoạt động từ Impalad được gọi là State stored. Trong trường hợp một node gặp sự cố, Statestore cập nhật thông tin sự cố cho các Impala

daemon khác, đồng thời ngăn không cho các Impala daemon khác giao thêm truy vấn vào node bị ảnh hưởng

- **Impala Metadata & Meta Store**: Impala sử dụng cơ sở dữ liệu MySQL hoặc PostgreSQL để lưu trữ định nghĩa các bảng. Các thông tin quan trọng như thông tin bảng, cột và định nghĩa bảng được lưu trữ trong một cơ sở dữ liệu trung tâm gọi là meta store. Mỗi node Impala lưu trữ bộ đệm metadata cục bộ để truy xuất nhanh chóng thông tin liên quan đến các bảng. Khi có sự cập nhật định nghĩa bảng hoặc dữ liệu bảng, các Impala daemon khác phải cập nhật bộ đệm metadata của mình bằng cách truy xuất metadata mới nhất trước khi thực hiện truy vấn mới liên quan đến bảng đó.

4.8 Phân tích và báo cáo

Dữ liệu sau khi được đưa vào kho dữu liệu phân tích đổ vào công cụ trực quan hóa dữ liệu Tableau thông qua ODBC Impala. Với giao diện đồ họa dễ sử dụng, người dùng có thể kết nối và tổ chức dữ liệu từ nhiều nguồn khác nhau. Tableau cung cấp nhiều loại biểu đồ và đồ thị phức tạp, giúp tìm ra mẫu, xu hướng và thông tin quan trọng từ dữ liệu. Ngoài ra, nó còn hỗ trợ tính năng phân tích dự đoán và khai phá dữ liệu. Tableau cũng cho phép chia sẻ và xuất báo cáo dễ dàng. Với tính linh hoạt và mạnh mẽ, Tableau là công cụ hữu ích giúp doanh nghiệp và nhà quản lý nắm bắt thông tin tình trạng chất lượng của các trạm.

Chương 5

Giải pháp kỹ thuật và kết quả thử nghiệm

5.1 Giải pháp kỹ thuật

5.1.1 Thu thập và chuẩn hóa dữ liệu raw counter

Hệ thống cần thực hiện thu thập dữ liệu nhiều file raw counter từ nhiều nguồn máy chủ khác nhau với giao thực kết nối FTP hoặc SFTP. Tần suất dữ liệu suất dữ liệu 15 phút của các đối tượng trạm. Hệ thống xử lý với dữ liệu của hãng Nokia xuất ra có định dạng XML với hình thức nén. Vì vậy cần được chuẩn hóa về định dạng CSV để phục vụ người dùng có thể tra cứu, sử dụng làm nguồn dữ liệu cung cấp cho việc tính toán các chỉ số KPI.

Hệ thống cần đáp ứng tốc độ đảm bảo thu thập đầy đủ và kịp thời xử lý khối lượng dữ liệu lớn trước khi bắt đầu chu kỳ sinh dữ liệu mới:

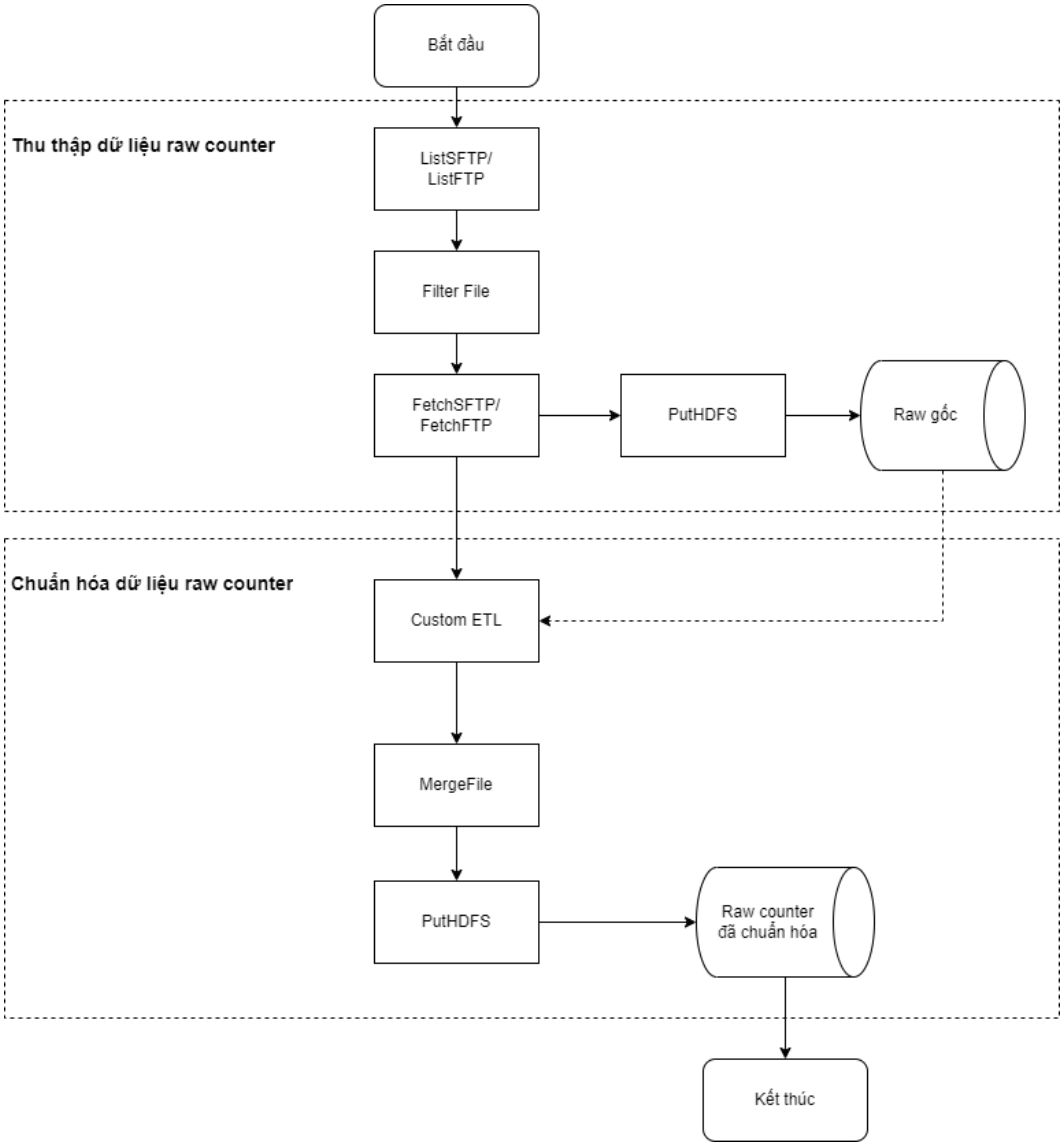
- Tổng dung lượng file raw counter chứa dữ liệu counter của trạm do hãng Nokia xuất ra cần xử lý trong một ngày gần 1TB.
- Số lượng trạm cần quản lý là hơn 118.000 trạm.

Phương án kỹ thuật được đưa ra là triển khai sử dụng Nifi làm công cụ thu thập dữ liệu từ máy chủ EMS. Nifi Hỗ trợ nhiều giao thức thu thập dữ liệu FTP, SFTP và có khả năng tùy biến cao. Giao diện trực quan, cho phép quản trị viên có thể theo dõi được luồng hoạt động thu thập dữ liệu. Có thể triển khai chạy

phân tán, song song hay đơn luồng tùy tác vụ, tăng tốc độ xử lý thu thập mà vẫn đảm bảo tính chính xác.

Việc chuẩn hóa thông tin dữ liệu đầu vào giúp quy chuẩn về một một thể thống nhất, dễ dàng tra cứu, giảm khối lượng dữ liệu bước tổng hợp phía sau. Và nó được thực hiện trên Nifi.

Các bước xử lý theo luồng hoạt động liên tục và xuyên suốt bao gồm các bước chính sau đây:



Hình 5.1: Luồng hoạt động các bước thu thập và chuẩn hóa.

Thu thập dữ liệu

Khai báo tạo các luồng xử lý trên Nifi thu thập dữ liệu raw counter cho từng nguồn dữ liệu.

- Sử dụng processor ListFTP hoặc ListSFTP và FetchFTP hoặc FetchSFTP để đáp ứng lấy file theo giao thức FTP hoặc SFTP.
- FilterFile: Danh sách file trước khi lấy về sẽ được lọc nhằm loại bỏ các trường hợp lấy nhầm và trùng lặp.
- PutHDFS: File sau khi lấy về thông qua các bước FetchFTP/FetchSFTP được đẩy vào storage của hệ thống nhằm lưu trữ lại file raw gốc ban đầu để sao lưu. Phục vụ cho tra cứu hoặc xử lý dữ liệu khi gặp sự cố mà không cần lấy lại file từ máy chủ EMS.

Các luồng thu thập đẩy dữ liệu sang luồng xử lý đọc và chuẩn hóa. Tạo thành luồng hoạt động xuyên suốt.

Chuẩn hóa dữ liệu

Trong Nifi có hỗ trợ sẵn việc đọc các file định dạng XML. Tuy nhiên do đặc thù dữ liệu counter do hãng Nokia xuất ra có định dạng cấu trúc phức tạm. Trong cùng một file dữ liệu, định đạng dữ liệu có thể khác nhau cho từng đối tượng, nhóm counter. Do vậy để có thể linh động trong việc đọc dữ liệu thì sẽ triển khai code các bước xử lý.

Xây dựng cơ sở dữ liệu cấu hình metadata, phục vụ cho quá trình chuẩn hóa:

- Định nghĩa cấu trúc dữ liệu cần chuẩn hóa phân loại từng nhóm counter tương ứng.
- Lưu trữ thông tin danh sách thường xuất hiện trong file raw counter.
- Lưu trữ thông tin về phân loại hàm tổng hợp dữ liệu của counter tương ứng.

Các xử lý tùy biến thực hiên chuẩn hóa file raw counter sẽ được sử dụng các thông tin cấu hình metadata trong cơ sở dữ liệu đã lưu để thực hiện chuẩn hóa dữ liệu trước khi đưa vào lưu trữ trên hệ thống. Đáp ứng việc khai thác và tổng hợp tính toán.

5.1.2 Tổng hợp dữ liệu và tính toán KPI

Phân tích yêu cầu

1. **Tổng hợp dữ liệu counter**:

   Đầu vào của bước tổng hợp, tính toán KPI chính là dữ liệu counter sau khi đã được thu thập và chuẩn hóa. Để dễ dàng cho việc tổng hợp thì hệ thống quy chuẩn nhất quán các dữ liệu về chu kỳ chung theo mức giờ.

   Người dùng có nhu cầu tổng hợp, thống kê các counter từ đó tính toán các chỉ số KPI chất lượng mạng của trạm.

2. **Tính toán KPI**:

   Từ dữ liệu counter đã được tổng hợp cần tính toán ra các chỉ số đánh giá chất lượng mạng theo công thức quy chuẩn được định nghĩa trước. Tương ứng với các công nghệ của Nokia, các đối tượng khác nhau có bộ công thức đánh giá chỉ số KPI khác nhau dựa trên các counter tương ứng.

Phương án kỹ thuật

Lưu trữ bảng định nghĩa quy tắc tổng hợp của counter mức giờ và bảng cấu hình các các công thức tính KPI trạm mức giờ trong metadata.

Hệ thống sử dụng Spark triển khai các job tổng hợp dữ liệu, tính toán KPI chạy trên Spark engine. Sử dụng các công cụ Oozie lập workflow, subworkflow để tạo các Spark ob thành luồng tổng hợp. Các công cụ này hỗ trợ:

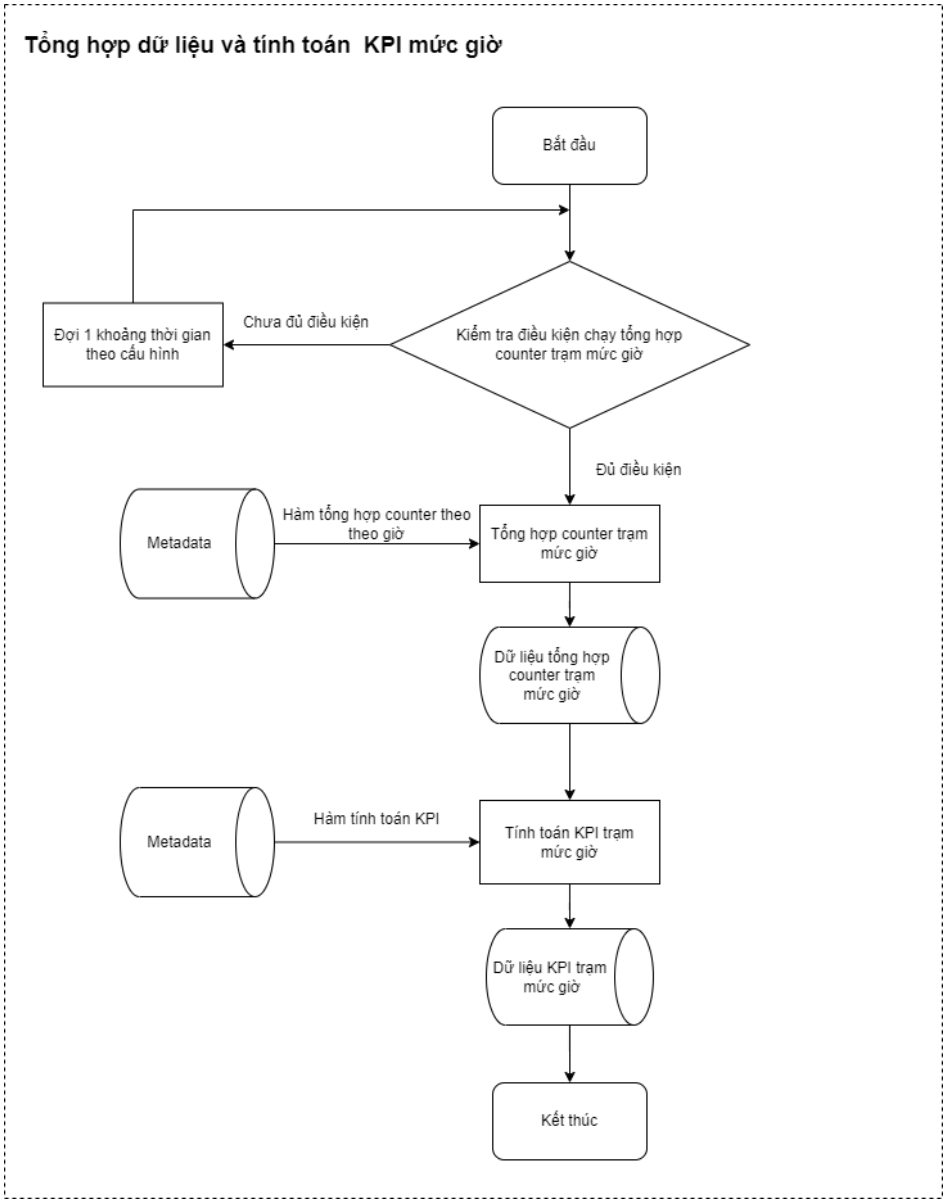
- Đặt lịch chạy cho các luồng xử lý định kỳ.

- Thiết kế các luồng xử lý tổng hợp dữ liệu counter, tính toán KPI.
- Thiết kế các luồng xử lý hoàn chỉnh. Tạo nên các luồng hoạt động xuyên suốt theo tuần tự.
- Hỗ trợ giám sát việc xử lý của các luồng đã cấu hình.
- Hỗ trợ cơ chế tự động chạy lại khi có sự cố hoặc lỗi trong quá trình xử lý.
- Hỗ trợ cấu hình các tham số hoạt động của luồng xử lý.

Các job xử lý tổng hợp dữ liệu counter của hãng Nokia. Từ dữ liệu counter đã thu thập và chuẩn hóa tổng hợp dữ liệu counter các mức giờ.

Các job xử lý tính toán KPI bao gồm: tính toán KPI các công nghệ, các loại đối tượng tương ứng từ dữ liệu counter đã tổng hợp theo công thức được khai báo theo cấu hình và tính toán KPI trạm.

Luồng tổng hợp tính toán KPI trạm mức giờ:



Hình 5.2: Luồng hoạt động tổng hợp, tính toán KPI trạm theo mức giờ.

Khi luồng bắt đầu, sẽ kiểm tra điều kiện chạy tổng hợp counter trạm mức giờ, nếu đã đủ điều kiện thì bắt đầu xử lý, nếu chưa đủ đợi một khoảng thời gian rồi quay lại bước kiểm tra:

- Kiểm tra đã vượt ngưỡng thời gian chờ đầy đủ dữ liệu hay chưa?

Tổng hợp counter trạm mức giờ tương ứng với các công việc:

- Loại bỏ các dữ liệu counter dư thừa, trùng lặp.

- Tổng hợp dữ liệu counter trạm tương ứng lên mức giờ theo hàm tổng hợp theo thời gian được cấu hình trong meta data cho từng counter.
- Lưu trữ dữ liệu counter trạm mức giờ vào cơ sở dữ liệu.

Tính toán KPI trạm theo mức giờ dựa theo công thức KPI riêng của từng trạm đã cấu hình, thực hiện tính toán KPI từ dữ liệu counter tổng hợp mức giờ và lưu trữ dữ liệu KPI đã tính toán được vào cơ sở dữ liệu.

5.2 Xây dựng hệ thống

5.2.1 Xây dựng metadata

Xây dựng metadata quản lý counter

Dữ liệu của mỗi trạm tại một thời điểm xuất file sẽ được lưu thành các file khác nhau ứng với mỗi nhóm counter, cần phải tổng hợp chúng lại thành 1 file duy nhất với dữ liệu của tất cả các nhóm counter. Vì thế cần phải xác định vị trí của counter trong file CSV chuẩn hóa. Ngoài ra đối với mỗi counter lại có một quy tắc tổng hợp riêng. Nên sẽ tiến hành xây dựng metadata conf_raw_counter_header để giải quyết những yêu cầu trên. Thông tin về bảng như sau:

STT	Trường	Kiểu dữ liệu	Chú thích
1	counter_index	string	Vị trí counter trong file CSV chuẩn hóa
2	name	string	Tên counter
3	agg_object_func	string	Quy tắc tổng hợp của counter

Bảng 5.1: Mô tả bảng conf_raw_counter_header.

Trích một số mẫu của bảng 5.1 như hình dưới đây:

COUNTER_INDEX	NAME	AGG_OBJECT_FUNC
126	VS.DataCQI.MEAN	SUM
127	DL Traffic Volume of Signaling on DCCH for Cell	SUM
128	VS.AckRetrans.4	SUM
129	VS.CQI.1	SUM
130	VS.CQI.25	SUM
131	VS.CQI.34	SUM
132	VS.DataOutput.Max	SUM
133	VS.DataOutput.Mean	SUM
134	VS.DataRabNum.Max	SUM

Hình 5.3: Mẫu bảng conf_raw_counter_header.

Xây dựng metadata công thức tính KPI

Do có nhu cầu có thể khai thác thêm các KPI đánh giá được hiệu năng mạng trong tương lai và lượng số KPI hiện đang khai thác rất lớn. Vì vậy, để giải quyết thuận tiện cho nhu cầu trên thì xây dựng metadata conf_kpi_formula động, lưu trữ các KPI và công thức tính của nó và cho phép thêm KPI mới vào. Thông tin bảng conf_kpi_formula như sau:

STT	Trường	Kiểu dữ liệu	Chú thích
1	formula_id	int	Id của công thức KPI
2	code	string	Mã của KPI
3	name	string	Tên của KPI
4	kpi_formula_code	string	Công thức KPI

Bảng 5.2: Mô tả bảng conf_kpi_formula.

Trích xuất một số mẫu của bảng 5.2 như hình dưới đây

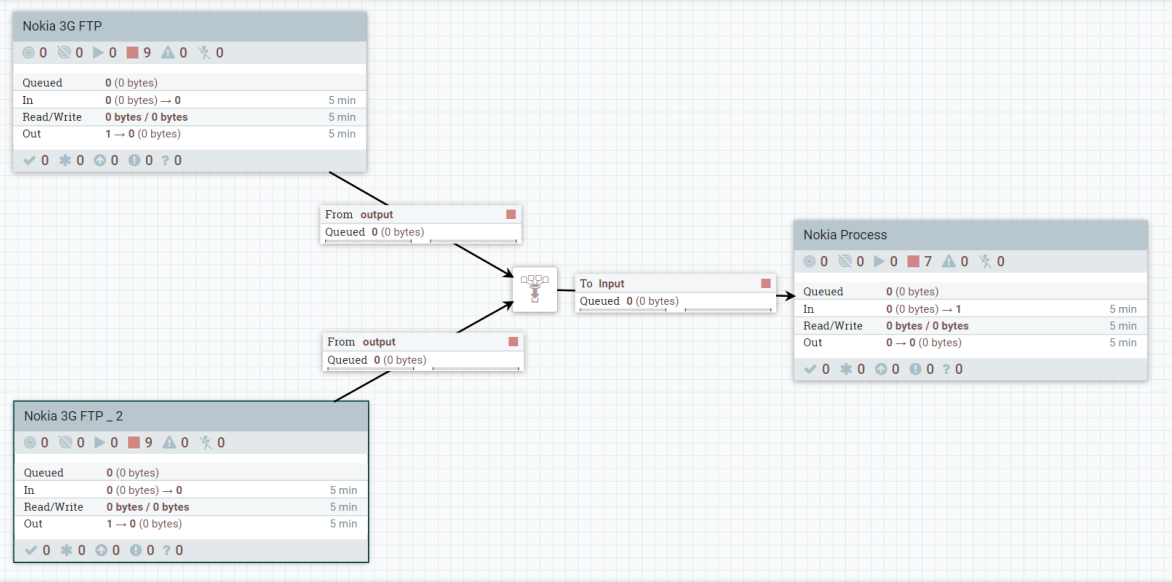
FORMULA_ID	CODE	NAME	KPI_FORMULA_CODE
6158	PSR99CALLDROPRATE	PS_R99 Call Drop Rate	100 * CASE WHEN (c_0347 + c_0361 + c_0362 + c_0348 + c_0363 + c_0364 + c_03
6171	TRAFFIC	CS_Total Traffic	(CASE WHEN c_0121 IS NULL THEN 0 ELSE c_0121 END + CASE WHEN c_0135 IS
6025	AvgUserNumber	Average User Number	c_0207
6125	CALLVOLUME	CS_Voice Call Volume	c_0086
6131	CSSR	CS_Call Setup Success Rate	CASE WHEN (c_0185 * (c_0122 + c_0130 + c_0145 - c_0166 - c_0170 - c_0174 - c_
6133	CSVIDEODROPCALLRATE	CS_Video Drop Call Rate	100 * CASE WHEN (c_0092 + c_0102 + c_0181 + c_0183 + c_0103 + c_0104 + c_01
6134	CSVIDEOTRAFFIC	CS_Video Traffic	(c_0135 * 64 / 12.2) / (60 * 100 * 60)
6140	DCR	CS_Drop Call Rate	100 * CASE WHEN (c_0091 + c_0095 + c_0096 + c_0180 + c_0182 + c_0184 + c_00

Hình 5.4: Mẫu bảng conf_kpi_formula.

Trong đó c_{counter_index} là giá trị của counter, counter_index là vị trí của counter trong file CSV đã chuẩn hóa.

5.2.2 Luồng thu thập và chuẩn hóa dữ liệu

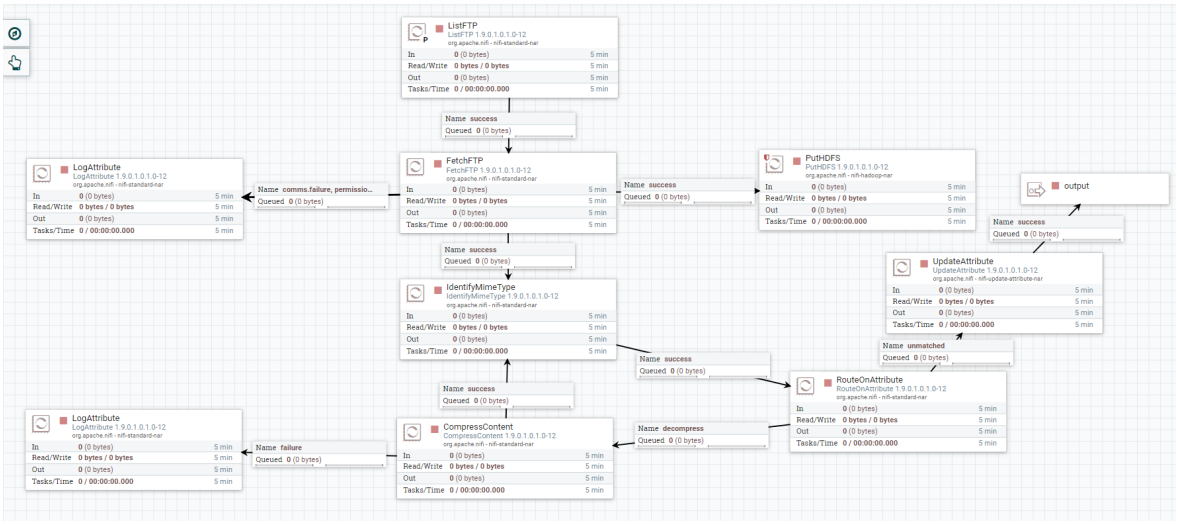
Luồng thu thập và chuẩn hóa dữ liệu được xây dựng như sau:



Hình 5.5: Luồng Nifi triển khai thu thập và chuẩn hóa trong hệ thống.

Luồng Nifi triển khai trên bao gồm hai luồng chính là luồng thu thập (Nokia 3G FTP và Nokia 3G FTP _2) và luồng chuẩn hóa dữ liệu (Nokia Process). Các file raw counter sau khi thu thập từ 2 máy chủ EMS sẽ được đưa vào vào trong Funnel để gộp lại thành nhiều luồng dữ liệu thành một luồng duy nhất và được đưa vào luồng chuẩn hóa.

Luồng thu thập dữ liệu raw counter từ máy chủ EMS được xây dựng như sau:



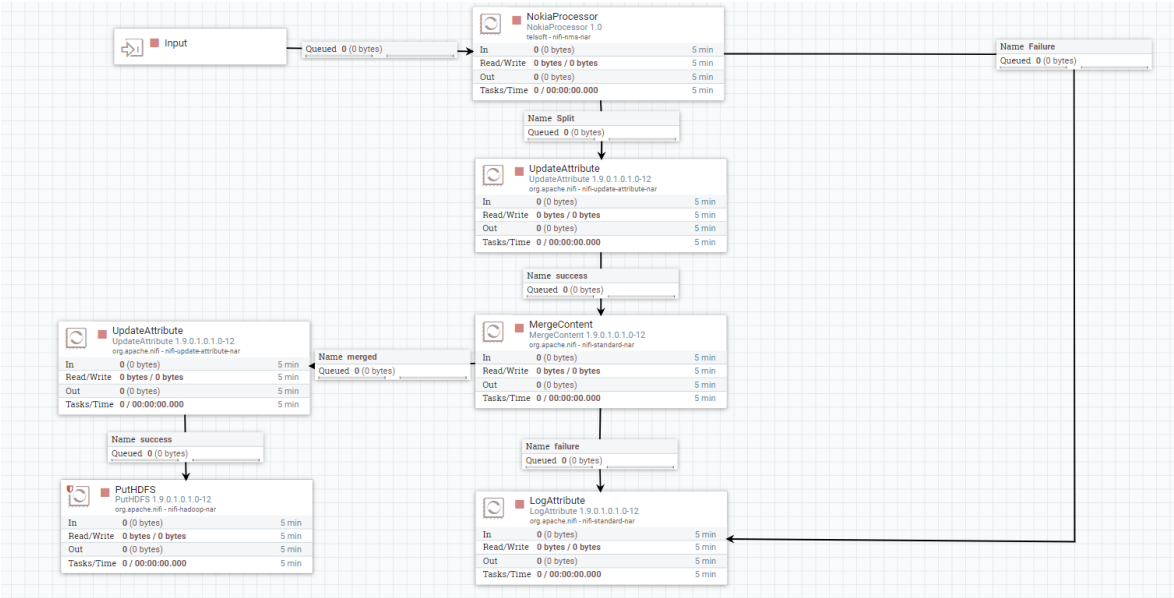
Hình 5.6: Luồng Nifi thu thập dữ liệu raw counter.

Luồng hoạt động để thu thập file raw counter từ máy chủ EMS Nokia 3G bắt đầu từ processor ListFTP, được sử dụng để liệt kê các file raw counter trên máy chủ. Tiếp theo, dữ liệu đi qua processor FetchFTP để lấy các file raw counter từ máy chủ và chuyển chúng thành các flowfile.

Các flowfile là file raw counter được đẩy vào HDFS bằng cách sử dụng processor PutHDFS để lưu trữ phiên bản gốc của file raw counter và tạo sao lưu. Các flowfile sau đó được chuyển qua processor IdentifyMimeType để xác định tự động loại MIME của chúng và điều hướng sang các processor xử lý tiếp theo.

Processor RouteOnAttribute được sử dụng để chia tách các flowfile vào các flowfile khác nhau dựa trên thuộc tính của chúng. Khi các flowfile đi qua processor RouteOnAttribute, nó sẽ kiểm tra thuộc tính của flowfile. Nếu loại MIME của flowfile là "application/gzip", nó sẽ được chuyển đến processor CompressContent để giải nén. Nếu flowfile không có loại MIME là "application/gzip", nó sẽ được đổ vào funnel để chờ xử lý tiếp theo.

Luồng Nifi chuẩn hóa dữ liệu counter được xây dựng như sau:



Hình 5.7: Luồng Nifi chuẩn hóa dữ liệu.

Sau khi được đổ vào funnel, các flowfile có định dạng XML sẽ được chuyển đến processor NokiaTransform để chuẩn hóa dữ liệu. Processor NokiaTransform đã được tạo bằng mã Java để đọc và trích xuất giá trị counter cũng như các đối tượng trạm tương ứng và sắp xếp chúng theo một quy chuẩn cụ thể. Các flowfile sau khi qua processor này sẽ chứa dữ liệu raw counter dưới dạng CSV, với các cột tương ứng là thời gian, đối tượng trạm và giá trị counter.

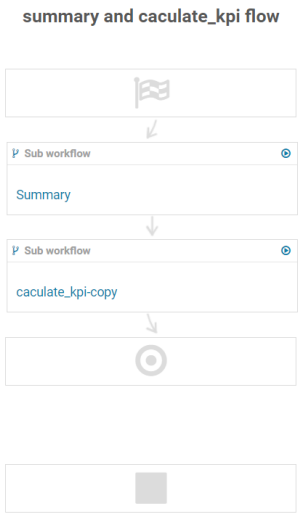
Sau khi flowfile đi qua processor UpdateAttribute, thuộc tính "correlation_file" sẽ được cập nhật để chứa thông tin về nhóm counter và ngày ghi nhận của flowfile. Điều này hỗ trợ việc ghép nội dung của các flowFile dựa trên thuộc tính "correlation_file" trong processor MergeContent. Tiếp theo, sau khi đi qua processor MergeContent, flowfile sẽ được chuyển qua processor UpdateAttribute một lần nữa để cập nhật các thuộc tính bổ sung như "data_day" (ngày ghi dữ liệu xuất), "data_hour" (giờ ghi nhận dữ liệu) và "group_counter_id" (nhóm counter) để hỗ trợ tổ chức thư mục lưu trữ trên HDFS.

Cuối cùng, dữ liệu sẽ được đẩy vào HDFS và tổ chức theo cấu trúc thư mục: "**{data_day}/ {data_hour}/{group_counter_id}**".

5.2.3 Luồng tổng hợp dữ liệu counter và tính toán KPI

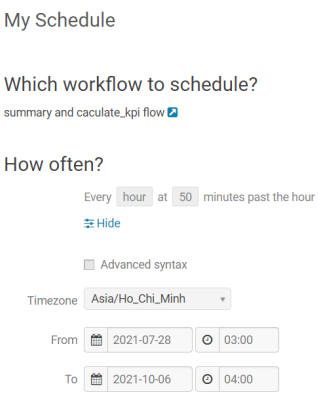
Dữ liệu sau khi được thu thập và chuẩn hóa thì sẽ được ánh xạ vào bảng raw_counter_nokia_cell3G_hourly trong Hive chứa dữ liệu counter của hãng Nokia xuất ra với tần suất 15 phút được ánh xạ từ HDFS bao gồm các trường thông tin: ID trạm (object_id), ngày thu thập dữ liệu (day_key), giờ thu thập dữ liệu (hour_key) và giá trị counter mà VNPT hiện đang khai thác đối với hãng Nokia.

Dữ liệu trong bảng raw_counter_nokia_cell3G_hourly là đầu vào của của WorkFlow tổng hợp và tính toán KPI:



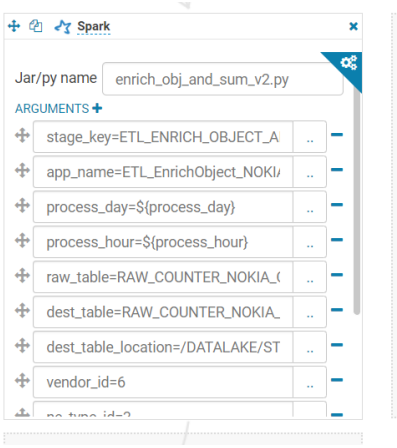
Hình 5.8: Luồng tổng hợp và tính toán KPI.

Workllow tổng hợp dữ liệu và tính toán KPI trạm theo mức giờ bao gồm 2 Subflow: "summary" và "caculate_kpi". Work Flow cứ chạy theo lịch trình, cứ mỗi giờ vào phút thứ 50 thì Workflow sẽ chạy tổng hợp và tính toán KPI giờ trước. Cấu hình lập lịch Workfllow như sau:



Hình 5.9: Cấu hình lập lịch Workflow

Đối với subflow "summary" được thực hiện bởi Spark job, nhiệm vụ của nó là tổng hợp dữ liệu counter lên mức giờ theo quy tắc tổng hợp đã được khai trong metadata conf_raw_counter_header. Cấu hình của subflow như sau:

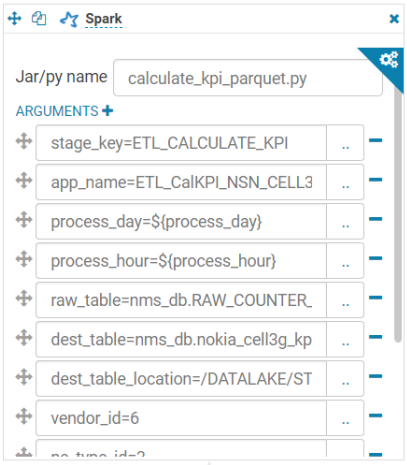


Hình 5.10: Cấu hình subflow summary

Dữ liệu counter của trạm sau khi được tổng hợp mức giờ được lưu trữ và ánh xạ lên bảng raw_counter_nokia_cell3G_hourly_all của Hive bao gồm các trường thông tin: ID trạm (object_id), ngày thu thập dữ liệu (day_key), giờ thu thập dữ liệu (hour_key) và giá trị của counter.

Dữ liệu của bảng raw_counter_nokia_cell3G_hourly_all được sử dụng để tính toán KPI mức giờ. Đối với subflow caculate_kpi thực hiện bởi Spark job

làm nhiệm vụ tính toán KPI sau khi subflow summary chạy xong. Cấu hình của subflow caculate_kpi như sau:



Hình 5.11: Cấu hình subflow caculate_kpi.

Các subflow yêu cầu truyền vào các tham số đầu vào

- app_name: tên Spark job chạy, thay đổi theo thời gian.
- process_day: ngày chạy subflow.
- process_hour: giờ chạy subflow.
- raw_table: tên bảng trên Hive dùng để tổng hợp và tính toán.
- dest_table: tên bảng trên Hive sau khi tổng hợp dữ liệu counter và tính toán KPI mức giờ.
- dest_table_location: vị trí lưu dữ liệu sau khi tổng hợp dữ counter và tính toán KPI mức giờ trên HDFS.
- vendor_id: mã của hãng Nokia.
- ne_type_id: công nghê 2G/3G/4G

Các tham số raw_table, dest_table, dest_table_location, vendor_id, ne_type_id sẽ được do quản trị hệ thống cấu hình tùy theo cách tổ chức và lưu trữ. Còn tham

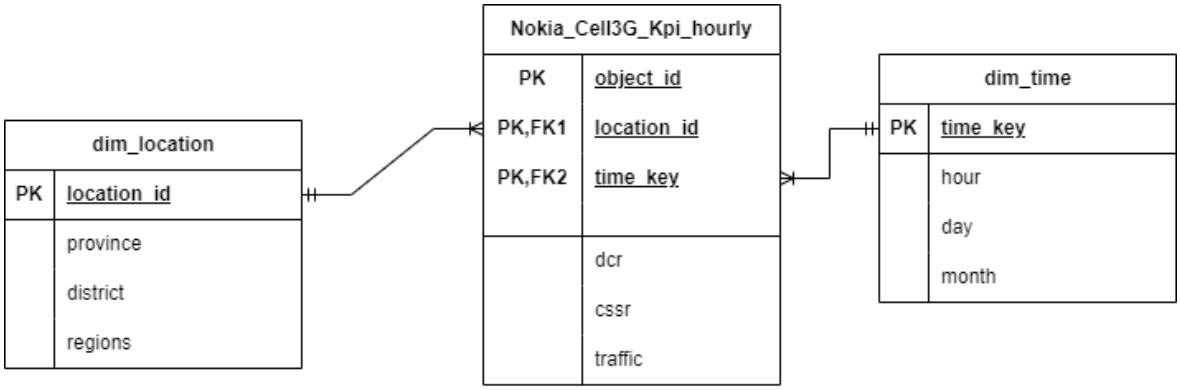
số app_name, process_day, process_hour sẽ được tự động lấy theo thời gian lập lịch.

Sau khi luồng chạy xong, dữ liệu tính KPI trạm theo mức giờ được lưu trữ trong trong HDFS với thông tin các trường là ID đối tượng trạm (object_id), ngày thu thập dữ liệu trạm (day_key), giờ thu thập dữ liệu (hour_key) và thông tin 85 KPI mà VNPT đang khai thác của hãng Nokia.

5.2.4 Kho dữ liệu và báo cáo phân tích

Dữ liệu KPI trạm được tính theo mức giờ được lưu trữ trong HDFS và được ánh xạ vào bảng dim và fact trong Impala để phục vụ cho việc trực quan hóa dữ liệu.

Thiết kế dữ liệu theo lược đồ hình sao với bảng fact: Nokia_Cell3G_Kpi_hourly lưu trữ dữ liệu hiệu suất mạng cho từng trạm, giờ và tỉnh và dim_time lưu trữ dữ liệu thời gian ghi nhận dữ liệu , bảng dim_location lưu trữ dữ vị trí (thông tin trạm thuộc vị trí địa lý được lưu trữ trong metadata). Trong phạm vi đồ án này, em xin trích xuất các KPI: dcr là tỷ lệ cuộc gọi bị gián đoạn, cssr là tỷ lệ cuộc gọi thành công, traffic là lưu lượng mạng để xây dựng bảng fact Nokia_Cell3G_Kpi_hourly. Thông tin liên kết các bảng như sau:



Hình 5.12: Mô hình dữ liệu OLAP.

Mô tả bảng dim_location

STT	Tên trường	Ý nghĩa
1	location_key	Khóa chính
2	province	Tên tỉnh, thành phố
3	district	Tên quận huyện
4	regions	Tên vùng, miền

Bảng 5.3: Mô tả bảng dim_location.

Mô tả bảng dim_time

STT	Tên trường	Ý nghĩa
1	time_key	Khóa chính
2	hour	Giờ thu thập dữ liệu
3	day	Ngày thu thập dữ liệu
4	month	Tháng thu thập dữ liệu

Bảng 5.4: Mô tả bảng dim_time

Mô tả bảng Nokia_Cell3G_Kpi_hourly.

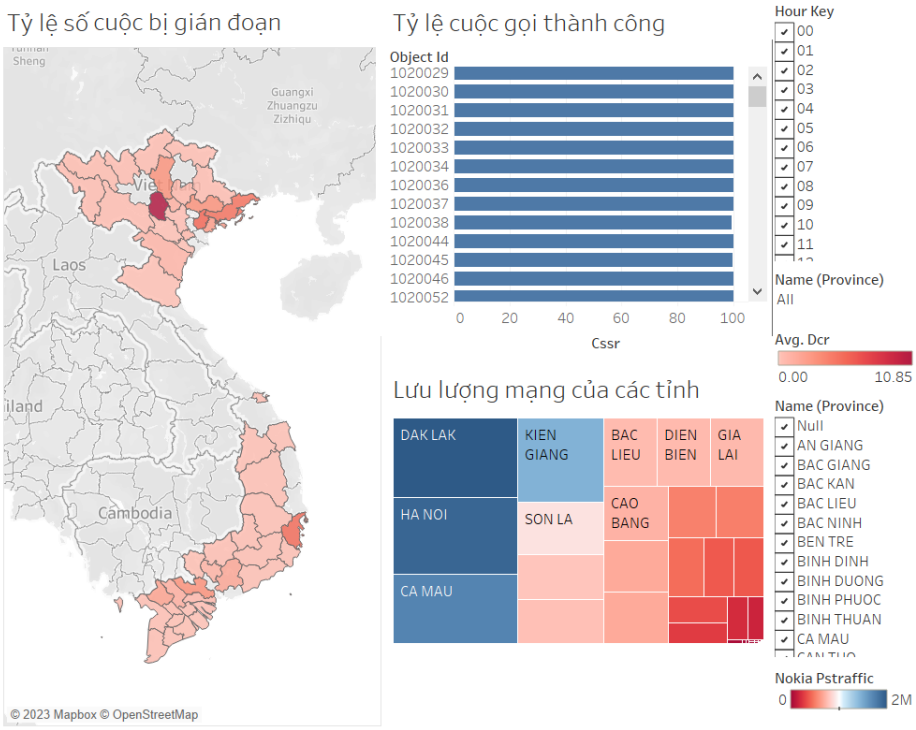
STT	Tên trường	Ý nghĩa
1	object_id	Mã định danh đối tượng trạm
2	time_key	Khóa ngoại tới bảng dim_time
3	location_key	Khóa ngoại tới bảng dim_location
4	cssr	Tỷ lệ cuộc gọi thành công

5	traffic	Lưu lượng mạng
6	dcr	Tỷ lệ cuộc gọi bị gián đoạn

Bảng 5.5: Mô tả bảng Nokia_Cell3G_Kpi_hourly.

Trên đây là thiết kế chuẩn cho mô hình dữ liệu của hệ thống. Tuy nhiên do dữ liệu sử dụng được cho việc thực hiện đồ án vẫn còn hạn chế (chưa có dữ liệu về thông tin quận/huyện hay khu vực của các trạm mà chỉ có thông tin tới mức tỉnh/thành phố) nên dữ liệu trong bảng dim_location chỉ dừng lại ở mức tỉnh/thành phố (province).

Xây dựng báo cáo cho thấy thông tin tổng quan về chất lượng mạng thông qua tỷ lệ cuộc gọi thành công, lưu lượng mạng, tỷ lệ cuộc gọi bị gián đoạn. Chi tiết báo cáo thể hiện như hình dưới đây.



Hình 5.13: Báo cáo phân tích

5.3 Kết quả thử nghiệm

5.3.1 Môi trường thử nghiệm

Hệ thống xây dựng đã được thử nghiệm trên môi trường của Cloudera phiên bản CDH 6, bao gồm 6 máy chủ. Thông tin cấu hình phần cứng của 6 máy này giống nhau nên em sẽ đưa ra cấu hình của một máy chủ.

STT	Thông số	Giá trị	Chú thích
1	Name	node1.nms.net	Tên máy chủ
2	IP	10.11.10.101	Địa chỉ IP máy chủ
3	Distribution	centos.7.9.2009	Phiên bản hệ điều hành của máy chủ
3	Cores	16	Số core CPU tối đa mà node được sử dụng
4	Disk	1TB	Dung lượng ổ đĩa
5	Memory	32GB	Dung lượng bộ nhớ

Bảng 5.6: Cấu hình phần cứng máy chủ

Thông tin cấu hình Nifi thử nghiệm thực hiện thu thập và chuẩn hóa như sau:

STT	Thông số	Giá trị	Chú thích
1	server	node3.nms.net	Tên máy chủ cài Nifi
2	IP address	10.11.10.103	Địa chỉ IP của máy chủ
3	ports	8080	Cổng kết nối giao diện web
4	queue size	8.85 GB	Kích thước hàng đợi

Bảng 5.7: Thiết lập cấu hình Nifi thu thập và chuẩn hóa dữ liệu

Thử nghiệm tổng hợp dữ liệu và tính toán KPI trên các cụm Spark gồm 1

node master chịu trách nhiệm phân phối công việc cho các node worker thực hiện tính toán song song dựa trên công việc master giao cho chúng.

Thông tin cấu hình các tham số quan trong của node master và node worker:

STT	Thông số	Giá trị	Chú thích
1	hostname	node1.nms.net	Tên máy chủ
2	Spark_MODE	master	Vai trò của node trong cụm
3	IP address	10.11.10.101	Địa chỉ IP của máy chủ
4	ports	8090	Cổng kết nối giao diện web
		7077	Cổng kết nối Spark master và Spark worker

Bảng 5.8: Thiết lập cấu hình node master cụm Spark

STT	Thông số	Giá trị	Chú thích
1	Spark_MODE	worker	Vai trò của node trong cụm
2	Spark_WORKER_MEMORY	16 GB	Dung lượng bộ nhớ tối đa mà node được sử dụng (16GB)
3	Spark_WORKER_CORE	8	Số lượng core mà node được sử dụng

Bảng 5.9: Thiết lập cấu hình node worker cụm Spark.

5.3.2 Đánh giá kết quả

Trong hệ thống xây dựng, có hai thành phần chính là thành phần thu thập chuẩn hóa dữ liệu và thành phần tổng hợp tính toán KPI mức giờ. Em sẽ tiến hành đánh giá tốc độ chạy của cả hai thành phần này.

Mục tiêu thời gian chay: Vì tần suất file raw counter hãng Nokia sinh ra là 15 phút, nên việc thu thập dữ chuẩn hóa cần thực hiện nhỏ hơn 15 phút trước khi chu kỳ dữ liệu mới được sinh ra. Và luồng tổng hợp dữ liệu counter và tính toán KPI sẽ phải hoàn thành xong trước khi file raw counter được suất ra sinh dữ liệu của giờ tiếp theo.

Đánh giá tốc độ thu thập và chuẩn hóa dữ liệu

Thực hiện thu thập và chuẩn hóa dữ liệu bằng Nifi trên một node cho kết quả như sau:

Batch	Dung lượng dữ liệu thu thập	Tốc độ thu thập chuẩn hóa
1	8.12 GB	3 phút 30 giây
2	8.22 GB	3 phút 20 giây
3	8.02 GB	2 phút 45 giây
4	8.11 GB	3 phút 03 giây
5	8.31 GB	2 phút 45 giây
6	8.13 GB	3 phút 09 giây
7	8.21 GB	3 phút 12 giây
8	8.21 GB	3 phút 08 giây
9	8.21 GB	3 phút 02 giây
10	8.21 GB	2 phút 55 giây

Bảng 5.10: Thử nghiệm tốc độ thu thập chuẩn hóa dữ liệu.

Trong thử nghiệm trên, lượng dữ liệu mà hãng Nokia xuất ra trong 15 phút trung bình là 8,16G. Tốc độ xử lý của Nifi rơi vào khoảng từ 2 phút 45 giây đến 3 phút 30 giây, kết quả này đảm bảo cho việc thu thập chuẩn hóa trước khi dữ

liệu chu kỳ mới được sinh ra.

Đánh giá thời gian chạy tổng hợp dữ liệu và tính toán KPI mức giờ

Thực hiện thử nghiệm cụm Spark với số node worker là 3, 4 và 5 để tổng hợp tính toán trong 1 giờ, đo thời gian chạy và tính trung bình từ 10 lần chạy thu được kết quả tốc độ chạy như sau:

Số lượng node worker	Thời gian chạy trung bình
3	8 phút 30 giây
4	6 phút 10 giây
5	4 phút 18 giây

Bảng 5.11: Thử nghiệm tốc độ tổng hợp dữ liệu và tính toán KPI.

Từ kết quả thử nghiệm bảng 5.11, sử dụng 5 nodeworker trong cụm đã nâng cao hiệu quả và hiệu suất xử lý dữ liệu. Sự mở rộng của cụm theo chiều ngang cung cấp tiềm năng linh hoạt và khả năng tăng cường khả năng xử lý của hệ thống, giúp đáp ứng nhu cầu ngày càng tăng của việc xử lý dữ liệu và tính toán các chỉ số KPI.

Đánh giá kết quả: Từ kết quả trên, cho thấy thời gian thu thập chuẩn hóa dữ liệu và tổng hợp, tính toán KPI có thể xử lý trong vòng 1 giờ. Việc lập lịch cho luồng tổng hợp dữ liệu và tính toán KPI chạy vào phút thứ 50 của mỗi giờ hoàn toàn đáp ứng được mục tiêu thời gian chạy. Với kết quả thử nghiệm trên hệ thống kiểm thử giải pháp cho thấy tính khả thi cho việc ứng dụng trong thực tế của VNPT.

Kết luận

Trong đồ án, em đã đạt được những kết quả sau:

- Trình bày tổng quan về dữ liệu lớn, kho dữ liệu - hồ dữ liệu, kiến trúc dữ liệu hai tầng kết hợp hồ dữ liệu và kho dữ liệu.
- Hiểu được nghiệp vụ thu thập và xử lý dữ liệu viễn thông, quản lý metadata cho dữ liệu counter, và thực hiện quy trình báo cáo thống kê về chất lượng mạng.
- Ứng dụng công nghệ lớn để xây dựng hệ thống có đầy đủ các thành phần cơ bản cho nhiệm vụ xử lý dữ liệu lớn.

Ngoài các kết quả đã đạt được ở trên, đồ án vẫn không tránh khỏi một số các hạn chế như sau:

- Hệ thống mới chỉ triển khai trên dữ liệu raw counter của hãng Nokia chưa triển khai trên toàn bộ dữ liệu các hãng Ericsson, Huawei, Alcatel và Motorola.
- Chưa trình bày được chi tiết về việc xây dựng kho dữ liệu và các báo cáo phân tích.

Để khắc phục các hạn chế và cải thiện hệ thống, em có phương hướng phát triển:

- Mở rộng hệ thống thu thập, xử lý tính toán KPI của các hãng Ericsson, Huawei, Alcatel và Motorola.
- Kết hợp mô hình máy học để dự báo trạm hư hỏng trong tương lai giúp tăng đáng kể độ chính xác của việc phát hiện sự cố và đưa ra biện pháp sửa chữa kịp thời. Việc này giúp giảm thiểu thời gian gián đoạn dịch vụ và tối ưu hóa hiệu suất mạng viễn thông.

Trong quá trình thực hiện đồ án, mặc dù đã cố gắng hết sức. Tuy nhiên với lượng kiến thức còn hạn hẹp, đồ án của em chắc chắn không tránh khỏi những thiếu sót. Kính mong nhận được sự thông cảm và góp ý của thầy cô, anh chị, bạn bè.

Em xin chân thành cảm ơn!

Tài liệu tham khảo

[1] Altexsoft, "Data Lakehouse: Concept, Key Features, and Architecture Layers", https://www.altexsoft.com/blog/data-lakehouse/.

[2] Cloudera, "Apache NiFi Overview", https://docs.cloudera.com/HDPDocuments/HDF3/HDF-3.3.0/apache-nifi-overview/content/nifi-architecture.html.

[3] Gartner, "Hype Cycle for Big Data, 2012", https://www.gartner.com/en/documents/2100215.

[4] Inmon, William H, *Building the data warehouse*, John wiley & sons, 2005.

[5] InterviewBit, "HDFS Architecture – Detailed Explanation", https://www.interviewbit.com/blog/hdfs-architecture/.

[6] Janusz Szwabiński, "Introduction into Big Data analytics", http://prac.im.pwr.wroc.pl/~szwabin/assets/bdata/1.pdf.

[7] Nguyễn Chí Thanh, "Apache Spark Fundamentals - Phần 4: Spark Catalyst Optimizer", https://www.facebook.com/legacy/notes/806633973073541/.

[8] Sindhuja Hari, "Hadoop vs Spark: Which is Better in 2023", https://hackr.io/blog/hadoop-vs-spark.

[9] SparkByExamples, "Apache Spark Tutorial", https://sparkbyexamples.com/#google_vignette.

[10] Tutorialspoint, “Hive - Introduction”, https://www.tutorialspoint.com/hive/hive_introduction.htm.

[11] Tutorialspoint, “Impala - Introduction”, https://www.tutorialspoint.com/impala/impala_quick_guide.htm.

[12] Wikipedia, “Big data”, https://en.wikipedia.org/wiki/Big_data.