BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

Độc lập – Tự do – Hạnh phúc

**KHOA CÔNG NGHỆ THÔNG TIN**

🙡🕮🙣



**BÁO CÁO TIỂU LUẬN CHUYÊN NGÀNH**

**ĐỀ TÀI**

**TÌM HIỂU KIẾN TRÚC LAMBDA CHO PHÂN TÍCH DỮ LIỆU VÀ ỨNG DỤNG**

**GVHD:** Th.S Quách Đình Hoàng

**SVTH:**

Lê Thị Thanh Phương 19133046

Nguyễn Thị Nhả Thư 19133054

***Thành phố Hồ Chí Minh, ngày 3 tháng 1 năm 2023***

# **Lời cảm ơn**

Lời đầu tiên, em xin chân thành cảm ơn Thầy Hoàng, Thầy đã tận tình hướng dẫn, chỉ bảo và luôn có sự phản hồi tỉ mỉ trong thời gian nhanh nhất nhằm giúp em trong suốt thời gian qua để có thể hoàn thành tiểu luận

Em chân thành cảm ơn quý Thầy/Cô của khoa Công nghệ thông tin, Trường Đại Học Đại học Sư phạm Kỹ thuật Tp Hồ Chí Minh đã tận tình truyền đạt kiến thức trong những năm em học tập. Với vốn kiến thức được tiếp thu trong quá trình học không chỉ là nền tảng cho quá trình nghiên cứu tiểu luận này mà nó còn là hành trang quý báu để em áp dụng vào thực tế một cách vững chắc và tự tin.

Mặc dù đã cố gắng hoàn thành tiểu luận trong phạm vi và khả năng cho phép nhưng chắc chắn sẽ không tránh khỏi những thiếu sót. Em rất mong nhận được sự thông cảm, góp ý và tận tình chỉ bảo của quý Thầy/Cô.

# **Nhận xét của Giảng viên**

# **Danh mục hình ảnh**

[Hình 1. Mô tả các lớp của kiến trúc Lambda 8](#_Toc123678590)

[Hình 2. Các giai đoạn xử lý dữ liệu của kiến trúc Lambda 9](#_Toc123678591)

[Hình 3. Mô tả lớp Batch 10](#_Toc123678592)

[Hình 4. Mô tả quá trình tính toán lại các batch views 11](#_Toc123678593)

[Hình 5. Mô tả quá trình tính toán gia tăng 11](#_Toc123678594)

[Hình 6. Mô tả quá trình xử lý của lớp Speed 15](#_Toc123678595)

[Hình 7. Minh họa việc tính toán các realtime views 17](#_Toc123678596)

[Hình 8. Mô tả việc tính toán gia tăng để cập nhật các realtime views 17](#_Toc123678597)

[Hình 9. Mô tả lớp Serving của kiến trúc Lambda 18](#_Toc123678598)

[Hình 10. Mô tả kiến trúc HDFS 22](file:///H:\TieuLuanChuyenNganh\BaoCaoTLCN_KienTrucLambda_KTDL.docx#_Toc123678599)

[Hình 11. Mô tả kiến trúc Kafka 26](#_Toc123678600)

[Hình 12. Mô tả thư viện Spark Streaming 27](#_Toc123678601)

[Hình 13. Kiến trúc Cassandra 28](#_Toc123678602)

[Hình 14. Mô tả quá trình xử lý dữ liệu Streaming 30](#_Toc123678603)

[Hình 15. Mô tả việc trình bày dữ liệu bằng Flask 31](#_Toc123678604)

[Hình 16. Tổng quát pipeline của ứng dụng 31](#_Toc123678605)

[Hình 17. Tập dữ liệu Stack Overflow 32](#_Toc123678606)

[Hình 18. Trang đăng nhập bằng User ID 33](#_Toc123678607)

[Hình 19. Trang chủ Dashboard của User tương ứng 33](#_Toc123678608)

# **Danh mục bảng**

[Bảng 1. So sánh hai thuật toán của lớp Batch 12](#_Toc123678623)

[Bảng 2. Yêu cầu và nhiệm vụ của lớp Batch 13](#_Toc123678624)

**MỤC LỤC**

[**Lời cảm ơn** 1](#_Toc123680368)

[**Nhận xét của Giảng viên** 2](#_Toc123680369)

[**Danh mục hình ảnh** 3](#_Toc123680370)

[**Danh mục bảng** 3](#_Toc123680371)

[**Phần 1: MỞ ĐẦU** 5](#_Toc123680372)

[**1.1.** **Lý do chọn đề tài** 5](#_Toc123680373)

[**1.2.** **Mục tiêu đề tài** 6](#_Toc123680374)

[**1.3.** **Đối tượng và phạm vi nghiên cứu** 6](#_Toc123680375)

[**1.3.1.** **Đối tượng nghiên cứu** 6](#_Toc123680376)

[**1.3.2.** **Phạm vi nghiên cứu** 6](#_Toc123680377)

[**1.4.** **Phương pháp nghiên cứu** 7](#_Toc123680378)

[**1.5.** **Kết quả dự kiến** 7](#_Toc123680379)

[**Phần 2: NỘI DUNG** 7](#_Toc123680380)

[**2.1. Tổng quan về Big data và kiến trúc Lambda** 7](#_Toc123680381)

[**2.1.1. Big data trong thời đại công nghệ 4.0** 7](#_Toc123680382)

[**2.1.2. Tổng quan kiến trúc Lambda** 8](#_Toc123680383)

[**2.2. Các framework sử dụng để xây dựng kiến trúc Lambda** 23](#_Toc123680384)

[**2.2.1. Lớp Batch** 23](#_Toc123680385)

[**2.2.2. Lớp Speed** 25](#_Toc123680386)

[**2.2.3. Lớp Serving** 28](#_Toc123680387)

[**2.3. Xây dựng các lớp của kiến trúc Lambda** 30](#_Toc123680388)

[**2.3.1. Xây dựng lớp Batch** 30](#_Toc123680389)

[**2.3.2. Xây dựng lớp Speed** 31](#_Toc123680390)

[**2.3.3. Xây dựng lớp Serving** 31](#_Toc123680391)

[**2.4. Xây dựng Dashboard** 32](#_Toc123680392)

[**2.4.1. Tổng quát về ứng dụng** 32](#_Toc123680393)

[**2.4.2. Tập dữ liệu** 33](#_Toc123680394)

[**2.4.3. Kết quả Dashboard** 34](#_Toc123680395)

[**Phần 3: KẾT LUẬN** 35](#_Toc123680396)

[**3.1. Kết quả đạt được** 35](#_Toc123680397)

[**3.2. Hạn chế** 35](#_Toc123680398)

[**3.3. Hướng phát triển** 35](#_Toc123680399)

[**TÀI LIỆU THAM KHẢO** 35](#_Toc123680400)

# **Phần 1: MỞ ĐẦU**

## **Lý do chọn đề tài**

Hiện nay với việc dữ liệu ở tất cả các lĩnh vực phát sinh ngày càng nhiều theo thời gian thì vấn đề xây dựng một hệ thống để xử lý các tập dữ liệu khổng lồ bao gồm dữ liệu trong quá khứ và dữ liệu theo thời gian thực là một vấn đề phổ biến trong các ngành khoa học và kỹ thuật. Nếu muốn khám phá hay thực hiện các truy vấn đến các tập dữ liệu khổng lồ này mà không có hệ thống xử lý thì sẽ rất khó khăn khi muốn quan sát đến các dữ liệu trong tập dữ liệu đó.

Nhóm em chọn đề tài “Tìm Hiểu Kiến Trúc Lambda Cho Phân Tích Dữ Liệu Và Ứng Dụng” với mong muốn tạo ra được một hệ thống xử lý dữ liệu theo từng lớp và các lớp này có tính liên kết chặt chẽ bằng hướng tiếp cận là xây dựng một đường ống (pipeline) xử lý các tập dữ liệu hàng nghìn bản ghi.

## **Mục tiêu đề tài**

Với đề tài này, nhóm chúng em tập trung tìm hiểu về các lớp của kiến trúc Lambda, sau đó dùng tập dữ liệu của Stack Overflow để minh họa lại cách xử lý của từng lớp trong kiến trúc Lambda. Sau khi xử lý tập dữ liệu xong thì người dùng có thể xem thông tin trong tập dữ liệu một cách dễ dàng thông qua Dashboard mà nhóm tạo ra.

## **Đối tượng và phạm vi nghiên cứu**

### **Đối tượng nghiên cứu**

Tập dữ liệu của Stack Overflow bao gồm các thông tin:

Xác định các Tag hàng đầu cho mỗi người dùng.

Cập nhật thông tin về UpVotes, DownVotes, Reputation trong thời gian thực.

Đề xuất câu hỏi trong thời gian thực bởi các Tag hàng đầu.

### **Phạm vi nghiên cứu**

Phạm vi: các framework được xây dựng và phát triển bởi Apache.

Phạm vi thời gian: trong khoảng 15 tuần từ 10/9/2022 đến ngày 25/12/2022

Phạm vi nội dung: trong bài nghiên cứu này, nhóm em qua tập trung vào việc cài đặt và kết nối các lớp lại với nhau để tạo thành một hệ thống xử lý tập dữ liệu lớn một cách suôn sẻ, mượt mà.

## **Phương pháp nghiên cứu**

Phương pháp thu thập dữ liệu: tập dữ liệu được lấy từ Stack Overflow và tập dữ liệu realtime nhóm em dùng API của Stack Overflow để lưu lại.

Phương pháp thực hiện: tập dữ liệu lịch sử được Spark xử lý sau đó đưa vào Cassandra và được lưu lại thành các bảng. Tập dữ liệu realtime được đọc thông qua Kafka và Spark Streaming và kết quả lưu vào Cassandra dạng bảng. Cuối cùng thực hiện truy vấn thông qua Dashboard xử lý bằng Flask.

## **Kết quả dự kiến**

Kết quả xây dựng kiến trúc Lambda: hiện thực được các lớp của kiến trúc Lamda thông qua xây dựng pipeline.

Kết quả ứng dụng: xem thông tin người dùng của Stack Overflow bằng cách đăng nhập vào hệ thống (web app) và Dashboard sẽ hiển thị các thông tin như độ nổi tiếng của người dùng, các bài đăng, lượt bình chọn,…

# **Phần 2: NỘI DUNG**

## **2.1. Tổng quan về Big data và kiến trúc Lambda**

### **2.1.1. Big data trong thời đại công nghệ 4.0**

Big data 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 hệ thống để đưa ra quyết định, khám phá và tối ưu hóa quy trình.

Với tổng khối lượng, tốc độ nhanh chóng và sự đa dạng của luồng dữ liệu, thực sự tạo ra thách thức lớn cho việc lấy thông tin từ các nguồn dữ liệu và đưa ra thông tin hữu ích. Trước tiên, ta cần phân tích vấn đề big data với các khía cạnh về loại thông tin cần được sử dụng, kiểu phân tích được sử dụng với các giải pháp, tiếp cận quá trình tiền xử lý, thông tin dữ liệu ban đầu về dữ liệu mà các giải pháp framework được yêu cầu để tải, xử lý, phân tích và lưu trữ.

Đối với loại thông tin cần thiết được sử dụng, loại phân tích được sử dụng như một giải pháp, cách tiếp cận tiền xử lý và nguồn gốc thông tin cho dữ liệu mà khung giải pháp được yêu cầu để thu nhận, tải, xử lý, phân tích và lưu trữ. Kiến trúc Lambda được biết đến như một mô hình dễ dàng hơn về thiết kế đến lưu trữ và xử lý dữ liệu một cách hợp lý. Trong kiến trúc Lambda đồng ý với ý tưởng cho rằng một công cụ hoặc một ứng dụng duy nhất không thể phục vụ như một giải pháp cho tất cả các vấn đề xử lý dữ liệu lớn. Thay vào đó, các công cụ khác nhau kết hợp để xây dựng một hệ thống phân lớp, trong đó mỗi lớp sẽ làm các nhiệm vụ cụ thể.

### **2.1.2. Tổng quan kiến trúc Lambda**

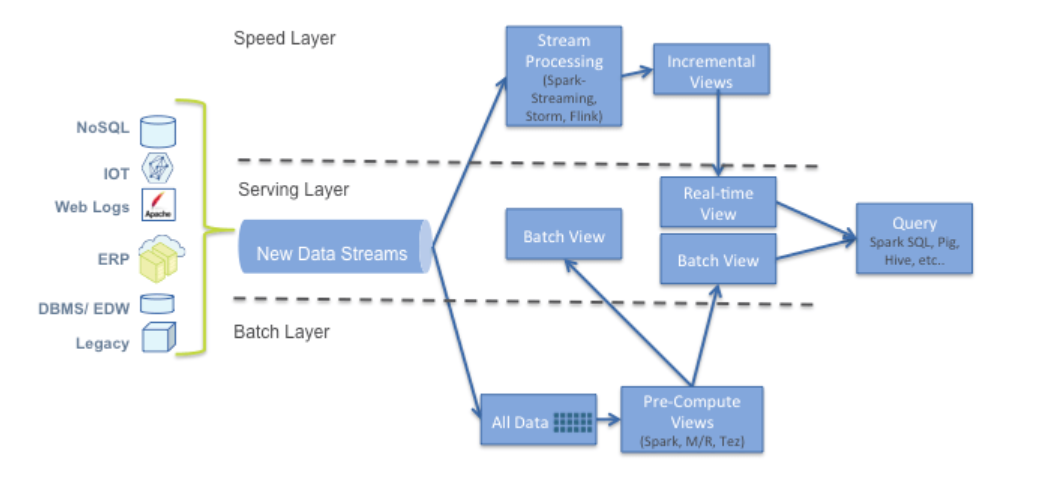
#### **2.1.2.1. Lịch sử ra đời và phát triển**

Kiến trúc Lambda ra đời bởi tác giả Nathan Marz, là một thuật ngữ chung đề cập đến một kiến trúc dữ liệu có thể mở rộng và chịu được lỗi, đảm bảo xử lý dữ liệu thời gian thực với độ trễ thấp.

Kiến trúc Lambda là kiến trúc xử lý dữ liệu được thiết kế để xử lý lượng dữ liệu khổng lồ bằng cách tận dụng cả phương pháp xử lý hàng loạt và xử lý luồng. Cách tiếp cận kiến trúc này là cố gắng cân bằng độ trễ, thông lượng và khả năng chịu lỗi bằng cách sử dụng phương pháp xử lý hàng loạt để cung cấp chế độ xem toàn diện và chính xác cho dữ liệu lô (batch data), đồng thời sử dụng xử lý luồng thời gian thực để cung cấp chế độ xem dữ liệu trực tuyến (realtime view). Hai đầu ra dạng view có thể được nối với nhau. Sự phát triển của kiến trúc Lambda tương quan với sự phát triển của dữ liệu lớn, phân tích thời gian thực và nỗ lực giảm thiểu độ trễ của việc xử lý dữ liệu.

#### **2.1.2.2. Hoạt động của kiến trúc Lambda**

Kiến trúc Lambda được chia thành 3 lớp: Batch layer, Speed layer và Serving layer.



Hình 1. Mô tả các lớp của kiến trúc Lambda [1]

*Hình 1* mô tả công việc của từng lớp của kiến trúc Lambda như lớp Batch sẽ xử lý tập dữ liệu gốc (master dataset) và tạo ra các batch view, lớp Speed sẽ xử lý dữ liệu mới được cập nhật đến (realtime) và tạo ra các real-time view, lớp Serving sẽ hợp nhất các batch views và realtime views sau đó thực hiện các truy vấn đến dữ liệu.

Kiến trúc Lambda được xây dựng như một hệ thống dữ liệu lớn với thành phần gồm các lớp. Mỗi lớp đáp ứng một vai trò, nhiệm vụ cụ thể (*Hình 1*) đảm bảo một số đặc điểm, thuộc tính nhất định. Kiến trúc Lambda kết hợp xử lý dữ liệu hàng loạt và thời gian thực trong một hệ thống.

Kiến trúc Lambda có 5 giai đoạn quan trọng:

Diagram

Description automatically generated

Hình 2. Các giai đoạn xử lý dữ liệu của kiến trúc Lambda [2]

*Hình 2* minh họa các giai đoạn xử lý dữ liệu của kiến trúc Lambda một cách chi tiết như sau:

Giai đoạn 1: liên quan đến việc nhập thêm dữ liệu vào hệ thống. Tại giai đoạn này dữ liệu mới được gửi đến hai lớp khác nhau, lớp Speed và lớp Batch nơi dữ liệu được xử lý.

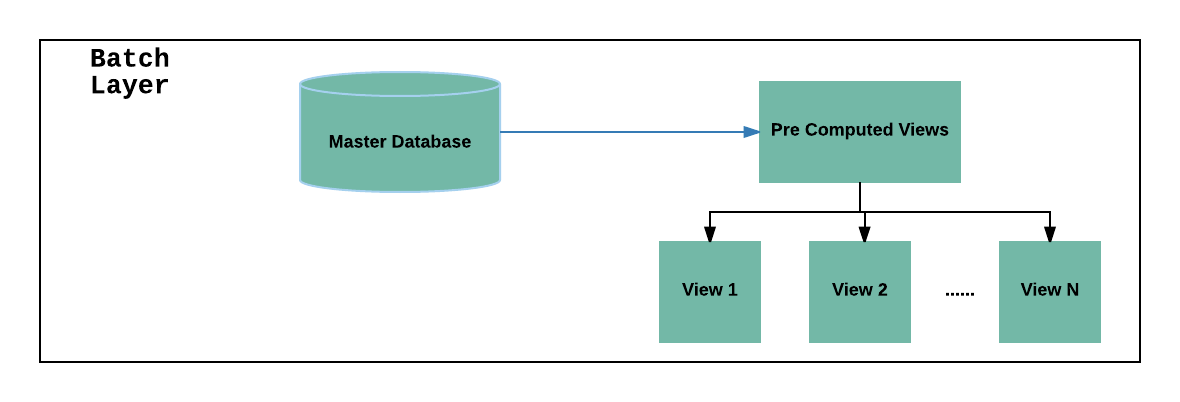
Giai đoạn 2: trong lớp Batch, dữ liệu được quản lý trong tập dữ liệu master và được tính toán trước tạo thành các batch views.

Giai đoạn 3: các batch view được chuyển đến lớp Serving - tại đây các batch view sẽ được đánh chỉ mục để cho phép dữ liệu được truy vấn với độ trễ thấp.

Giai đoạn 4: trong lớp Speed, chỉ những dữ liệu gần đây mới được xử lý. Lớp Speed thực hiện cập nhật dữ liệu mới nhất cho hệ thống, cung cấp truy vấn với độ trễ cao.

Giai đoạn 5: truy vấn kết quả từ hệ thống được trả lời khi kết quả của batch view trong lớp Serving và lớp Speed được hợp nhất.

#### **2.1.2.3. Lớp Batch**



Hình 3. Mô tả lớp Batch [2]

*Hình 3* tập dữ liệu chính sẽ được tính toán thành các batch views trong lớp Batch trước khi đưa đến lớp Serving

Lớp Batch là thành phần cốt lõi của kiến trúc Lambda, là nơi lưu trữ bản sao của tập dữ liệu chính (master dataset) và thực hiện tính toán trước các batch view trên tập dữ liệu đó (*Hình 3*). Lớp batch có độ trễ cao, điều này cho phép các phân tích sâu hơn và tính toán phức tạp.

Vì tập dữ liệu chính quá lớn để xử lý với một máy tính duy nhất, nên nó được phân phối trên nhiều máy trong cụm. Vì dữ liệu được phân chia giữa các máy nên hệ thống tệp phân tán (distributed file system) là cần thiết để đọc, tải và xử lý dữ liệu này.

Trong lớp Batch thực hiện 2 công việc:

Lưu trữ tập dữ liệu chính (master dataset) – tập dữ liệu bất biến, chỉ cho phép nạp thêm dữ liệu.

Tính toán các function trên tập dữ liệu lịch sử với độ chính xác cao, thực hiện batch view = function (all data)

Mô tả lớp Batch như sau:

function batch\_layer ():

while (True):

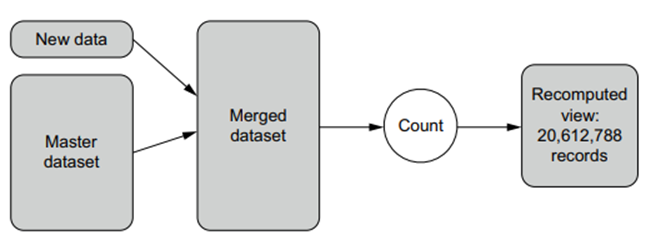
ingest\_data ()

compute\_batch\_view ()

append\_to\_master\_dataset ()

##### **2.1.2.3.1. Tính toán trong lớp Batch**

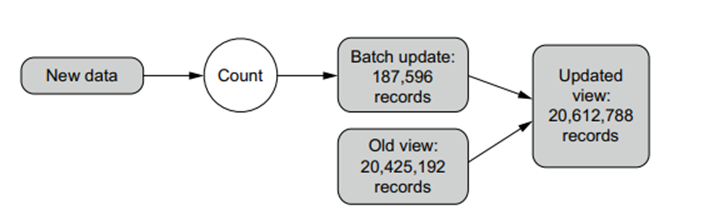
Vì tập dữ liệu liên tục phát triển, nên việc cập nhật batch view mỗi khi có dữ liệu mới cũng là một vấn đề đối với hệ thống. Trong lớp Batch, cách tiếp cận để xử lý vấn đề này qua hai thuật toán tính toán như sau:



Hình 4. Mô tả quá trình tính toán lại các batch views [2]

*Hình 4* khi có dữ liệu mới đến, bằng cách gộp tập dữ liệu cũ và dữ liệu mới lại thành một và thực hiện tính lại batch views.

Đối với thuật toán tính toán lại các batch view khi có thêm dữ liệu mới sẽ làm cho hiệu suất hoạt động của kiến trúc sẽ không cao, tốc độ tính toán chậm hơn.



Hình 5. Mô tả quá trình tính toán gia tăng [2]

*Hình 5* đối với thuật toán gia tăng chỉ tính toán trên dữ liệu mới, không tính toán lại dữ liệu cũ, sau khi tính toán dữ liệu mới thì cập nhật lại batch view và kết hợp với old view (được tính toán từ dữ liệu cũ), kết quả cuối cùng là hợp nhất batch update (dữ liệu mới) và old view (dữ liệu cũ) thành batch view hoàn chỉnh.

Bảng 1. So sánh hai thuật toán của lớp Batch

|  |  |  |
| --- | --- | --- |
| **Yếu tố** | **Thuật toán tính toán lại** | **Thuật toán gia tăng** |
| Hiệu suất | Yêu cầu sự nỗ lực xử lý trên toàn bộ tập dữ liệu master | Yêu cầu tính toán trên tài nguyên dữ liệu ít hơn, nhưng tạo ra batch view nhiều hơn |
| Chịu lỗi từ con người | Vì việc tính toán lỗi có thể dễ dàng khắc phục lỗi do con người gây ra | Không tạo điều kiện cho việc sữa chữa lỗi cho batch view |
| Tính tổng quát | Độ phức tạp của thuật toán được giải quyết trong quá trình tính toán trước, kết quả batch view đơn giản, xử lý với độ trễ thấp và nhanh chóng | Yêu cầu xử lý đặc biệt, có thể thay đổi phức tạp để xử lý truy vấn nhanh chóng |

Việc thuật toán gia tăng mang lại hiệu quả hơn so với thuật toán tính toán lại toàn bộ dữ liệu (*Bảng 1*). Nhưng tính hiệu quả không phải yếu tố duy nhất được xem xét. Sự cân bằng giữa hiệu suất và khả năng chịu lỗi của con người, tính tổng quát hóa của thuật toán.

Một trong những vấn phát sinh là việc tính toán trước các batch view, thì việc phát sinh dữ liệu mới chưa được xử lý (un-batch data), dẫn đến kết quả của việc truy vấn function chưa thực sự chính xác. Để giải quyết vấn đề dữ liệu real-time và near real-time, kiến trúc Lambda xây dựng một lớp để giải quyết riêng cho dữ liệu mới phát sinh, lớp được tạo ra để giải quyết vấn đề đó là lớp Speed.

##### **2.1.2.3.2. Yêu cầu dữ liệu trên lớp Batch**

Việc lưu dữ dữ liệu trong lớp Batch phải được tối ưu hóa để xử lý một tập dữ liệu lớn, liên tục được phát triển. Vì lớp Batch chịu trách nhiệm về việc tính toán các function trên tập dữ liệu chính (master dataset) để tạo nên batch view. Nghĩa là lớp này yêu cầu lưu trữ, xử lý tính toán rất nhiều dữ liệu cùng một lúc.

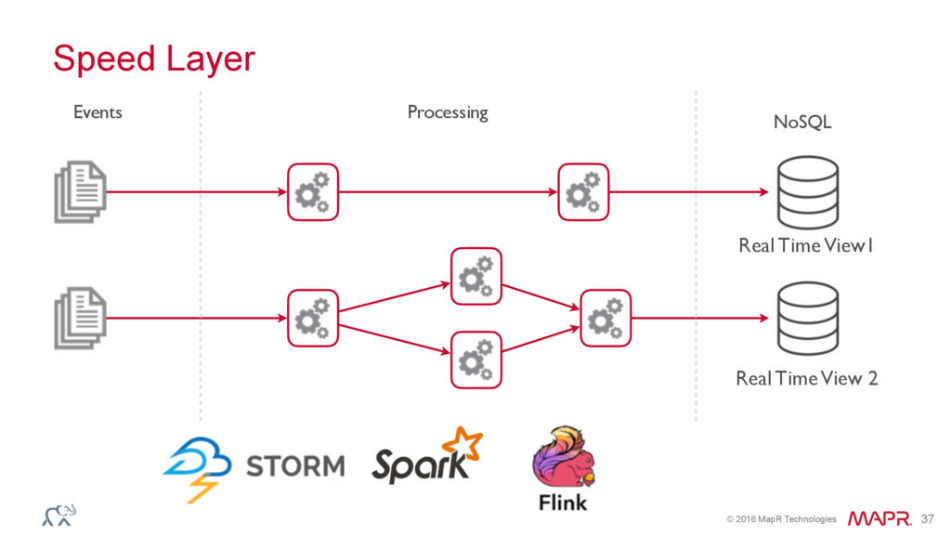
Bảng 2. Yêu cầu và nhiệm vụ của lớp Batch

|  |  |  |
| --- | --- | --- |
| **Hành động** | **Yêu cầu** | **Nhiệm vụ** |
| Ghi dữ liệu (Write) | Hiệu quả trong việc thêm dữ liệu mới vào tập dữ liệu chính. | Thao tác ghi duy nhất được thực hiện trong lớp batch là thêm các đối tượng vào tập dữ liệu chính. |
| Mở rộng lưu trữ | Batch layer lưu trữ toàn bộ tập dữ liệu chính - nó có thể lên đến Terabytes hay PetaBytes. Do đó nó phải dễ dàng được mở rộng lưu trữ khi tập dữ liệu ngày càng phát triển |
| Đọc (Read) | Hỗ trợ xử lý song song | Việc xây dựng tính toán function batch view theo yêu cầu trên toàn bộ tập dữ liệu. Nên lưu trữ trên lớp Batch phải hỗ trợ quá trình xử lý song song để giải quyết dữ liệu với số lượng lớn và có thể mở rộng |
| Ghi và Đọc (Write and Read) | Điều chỉnh chi phí lưu trữ và xử lý tính toán | Việc lưu trữ có thể tốn chi phí. Việc lựa chọn nén tập dữ liệu để giảm thiểu chi phí, nhưng thực hiện giải nén dữ liệu trong quá trình tính toán có thể ảnh hưởng đến hiệu suất. Việc xây dựng lớp Batch về cách lưu trữ và nén dữ liệu phụ thuộc vào từng yêu cầu của người dùng. |
| Tính bất biến của dự liệu được đảm bảo | Điều quan trọng trong việc lưu trữ dữ liệu chính là không làm thay đổi dữ liệu. Để tránh các tác động đến dữ liệu nên thực hiện các kiểm tra không cho phép các hoạt động không cho phép thay đổi dữ liệu xảy ra. |

##### **2.1.2.3.3. Giải pháp lưu trữ cho lớp Batch**

Hệ thống tệp phân tán là một trong những giải pháp tối ưu được sử dụng để triển khai cho lớp Batch. Khi việc lưu trữ được thực hiện trên nhiều cụm bao gồm nhiều máy tính khác nhau. Việc mở rộng được thực hiện bằng cách thêm nhiều máy tính. Khả năng chịu lỗi của hệ thống tệp phân tán được thiết kế, khi một máy bị lỗi, tất cả file và dữ liệu vẫn có thể được truy cập bởi dữ liệu được nhân bản và lưu trữ trên máy khác.

#### **2.1.2.4. Lớp Speed**



Hình 6. Mô tả quá trình xử lý của lớp Speed [3]

*Hình 6* khi có dữ liệu mới xuất hiện thì lớp Speed sẽ xử lý thông qua thư viện Spark Streaming và kết quả là tạo ra các realtime views tương ứng.

Lớp Speed chỉ hoạt động trên dữ liệu gần đây nhất. Mục đích của lớp Speed là: đảm bảo khi dữ liệu mới được hình thành, sẽ hiện diện trong truy vấn một cách nhanh chóng nhất có thể khi mà ứng dụng yêu cầu. Điểm tương đồng với lớp Batch là cung cấp view dựa trên dữ liệu mà lớp đó nhận được. Điều khác biệt cơ bản, khi lớp Speed xem xét dữ liệu cần đây khi lớp Batch xem xét dữ liệu lịch sử.

real time view = function (real-time data, new data)

Lớp Speed hỗ trợ cả truy vấn đọc và ghi ngẫu nhiên. Do đó, nó đòi hỏi một hệ thống cơ sở dữ liệu phức tạp hơn nhiều.

Mô tả lớp này như sau:

function speed\_layer ():

while (True):

ingest\_data ()

compute\_speed\_view ()

Lớp Speed hỗ trợ nhiều hoạt động phức tạp hơn lớp Batch. Điều này một phần là do lớp Speed nên có độ trễ thấp và một phần do lớp Speed không có quyền truy cập vào toàn bộ dữ liệu (master data). Vì vậy, các hoạt động yêu cầu dữ liệu trong quá khứ không thể được lớp Speed hỗ trợ. Tuy nhiên kết quả của lớp Speed thường không có độ chính xác như ở lớp Batch do hạn chế về thời gian xử lý. Đối với lớp Speed chúng ta có thể sử dụng các công cụ như Apache Storm, Apache Spark hoặc Apache Flume…

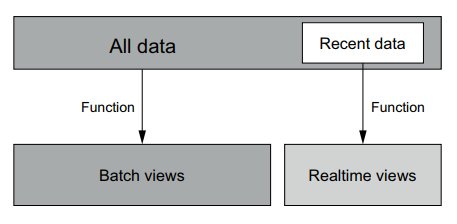
Lớp Speed sẽ cập nhật các kết quả truy vấn khi có dữ liệu mới và bù cho độ trễ cao của lớp Batch.

Lớp Speed sẽ tính toán dữ liệu gia tăng (real-time) thay vì tính toán lại dữ liệu cũ (history). Các lợi ích của lớp Speed: thứ nhất, lớp Speed chịu trách nhiệm cho dữ liệu chưa được cập nhật trong lớp Serving, dữ liệu này được cập nhật nhiều nhất trong vài giờ và nhỏ hơn tập dữ liệu chính. Xử lý dữ liệu ở qui mô nhỏ hơn và cho phép thiết kế linh hoạt hơn. Thứ hai, các realtime view của lớp Speed xảy ra nhanh. Sau khi dữ liệu được đưa vào các view của lớp Serving nó có thể được xóa khỏi lớp Speed. Có hai khía cạnh chính của lớp speed: stored (lưu trữ) và tính toán realtime view.

##### **2.1.2.4.1. Tính toán realtime view**

Mục tiêu của lớp speed cũng giống như lớp batch và hệ thống phân tán là tính toán và tạo ra các realtime view để truy vấn một cách nhanh chóng. Sự khác nhau giữ lớp Speed và lớp Batch là lớp Speed đại diện cho dữ liệu vừa mới cập nhật, lớp Batch đại diện cho dữ liệu cũ (history). Dữ liệu mới được cập nhật sau mỗi vài mili giây hoặc vài giây tùy vào mỗi ứng dụng. Cho dù ta đang xây dựng ứng dụng như thế nào thì ta cũng cần tạo ra các realtime view một cách tiết kiệm tài nguyên với độ trễ vài mili giây.

realtime view = function (recent data)

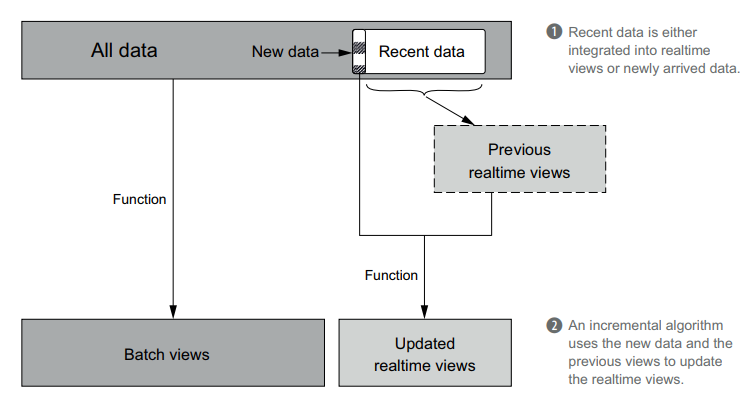


Hình 7. Minh họa việc tính toán các realtime views [2]

*Hình 7* các realtime views được tính toán thông qua dữ liệu mới được đưa vào hệ thống và không truy cập vào tập dữ liệu cũ trước đó.

Nói chung, bất kỳ giải pháp khả thi nào cũng dựa vào việc sử dụng các thuật toán gia tăng, ý tưởng là cập nhật các realtime view khi dữ liệu mới đến, do đó sử dụng lại việc trước đây đã tạo ra các view. Điều này đòi hỏi phải sử dụng cơ sở dữ liệu đọc ngẫu nhiên/ghi ngẫu nhiên để có thể thực hiện cập nhật trên cơ sở dữ liệu hiện có.

realtime view = function (new data, previous realtime view)



Hình 8. Mô tả việc tính toán gia tăng để cập nhật các realtime views [2]

*Hình 8* khi dữ liệu mới được xử lý và tạo thành các realtime views mới được cập nhật với các realtime views trước đó thành các realtime view hoàn chỉnh.

##### **2.1.2.4.2. Lưu trữ realtime view**

Kiến trúc Lambda yêu cầu đọc ngẫu nhiên có độ trễ thấp và việc sử dụng các thuật toán gia tăng đòi hỏi phải cập nhật ngẫu nhiên có độ trễ thấp. Do đó, lớp lưu trữ bên dưới phải đáp ứng các yêu cầu sau:

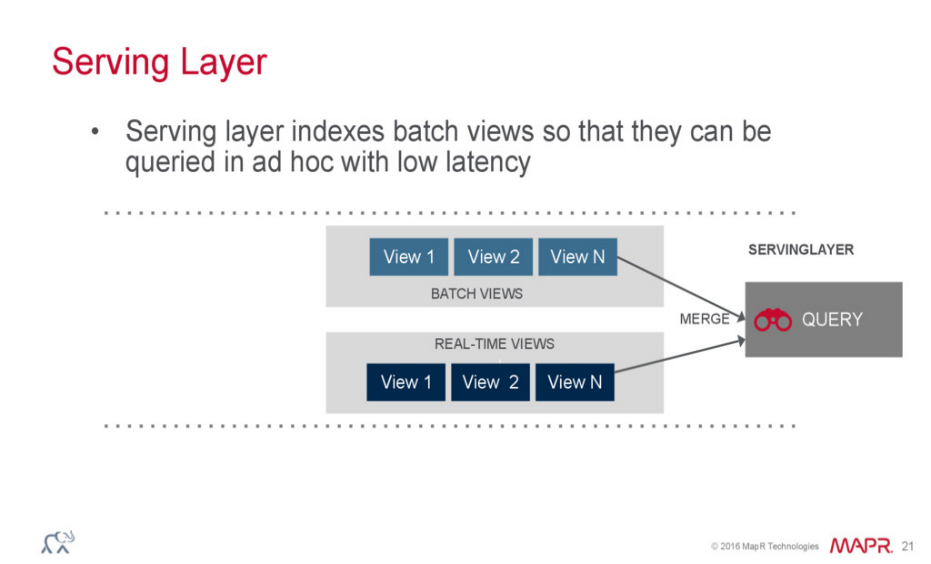
Đọc ngẫu nhiên - realtime view sẽ hỗ trợ đọc ngẫu nhiên nhanh để trả lời truy vấn nhanh chóng. Điều này có nghĩa là dữ liệu chứa trong đó phải được lập các chỉ mục (index).

Ghi ngẫu nhiên - để hỗ trợ các thuật toán gia tăng, cũng phải có khả năng sửa đổi chế độ xem thời gian thực với độ trễ thấp.

Khả năng mở rộng - cũng như các chế độ xem lớp phân phối, các chế độ xem thời gian thực sẽ mở rộng theo lượng dữ liệu họ lưu trữ và tốc độ đọc / ghi theo yêu cầu của ứng dụng. Thông thường, điều này ngụ ý rằng các chế độ xem thời gian thực có thể được phân phối trên nhiều Máy.

Khả năng chịu lỗi - nếu ổ đĩa hoặc máy gặp sự cố, chế độ xem thời gian thực sẽ tiếp tục để hoạt động bình thường. Khả năng chịu lỗi được thực hiện bằng cách sao chép dữ liệu trên máy móc để có bản sao lưu nếu một máy duy nhất bị lỗi.

#### **2.1.2.5. Lớp Serving**



Hình 9. Mô tả lớp Serving của kiến trúc Lambda [4]

Lớp Serving là một thành phần cuối cùng của kiến trúc Lambda. Lớp này liên kết chặt chẽ với lớp Batch và lớp Speed. Lớp Batch cung cấp các batch view. Các batch view này sẽ luôn được cập nhật bởi lớp Speed. Khi một truy vấn được gửi đến lớp Serving, nó sẽ lần lượt hình thành và kích hoạt hai truy vấn đến hai lớp khác nhau, cụ để là lớp Batch và lớp Speed. Các truy vấn được kích hoạt xem xét timestamp của các bản ghi và tìm nạp kết quả theo các tham số cung cấp. Khi lớp Serving nhận được bản ghi từ hai lớp, nó sẽ tiến hành hợp nhất kết quả truy vấn.

Lớp Serving là một cơ sở dữ liệu phân tán, có thể hợp nhất batch view và real time view thành một và truy vấn kết quả trong thời gian thực trên toàn bộ tập dữ liệu. Lớp cũng có thể dễ dàng mở rộng, và cung cấp khả năng truy cập độ trễ thấp trên toàn bộ dữ liệu.

query = function (batch view, real time view)

Lớp Serving cung cấp khả năng đọc ngẫu nhiên, bất cứ khi nào ứng dụng được truy vấn, lớp này cung cấp kết quả truy vấn từ hai lớp Batch và lớp Speed.

Một ứng dụng được xây dựng để xử lý hàng nghìn lượt sử dụng đồng thời. Lớp Speed được phân phối trên toàn cụm để có khả năng năng mở rộng và cân bằng tải để hỗ trợ truy cập đồng thời. Hai yêu tố cần được xem xét khi thiết kế chỉ số cho lớp Serving – thông lượng và độ trễ. Độ trễ là thời gian cần thiết để trả lời một truy vấn, trong khi thông lượng là số lượng truy vấn có thể được phục vụ trong một thời gian nhất định.

##### **2.1.2.5.1. Yêu cầu của lớp Serving đối với cơ sở dữ liệu**

Kiến trúc Lambda đặt ra một số yêu cầu nhất định trên cơ sở dữ liệu của lớp Serving:

Có thể ghi hàng loạt: các batch view được tạo ra từ đầu. Khi có phiên bản mới của batch view, phải tiến hành thay đổi bản batch view mới nhất.

Có thể mở rộng: cơ sở dữ liệu lớp Serving phải có khả năng xử lý view có kích thước lớn.

Đọc ngẫu nhiên: cơ sở dữ liệu lớp Serving phải hỗ trợ đọc ngẫu nhiên, với việc cung cấp chỉ mục có thể giúp cho việc truy cập trực tiếp vào các view. Yêu cầu này là cần thiết để có độ trễ thấp cho các truy vấn. Đồng thời, lớp Serving không cung cấp khả năng ghi ngẫu nhiên vì lớp Serving được thiết kế chỉ để trả lời câu hỏi truy vấn. Do đó, sự đơn giản của cơ sở dữ liệu giúp lớp Serving trả lời các câu hỏi truy vấn có độ trễ thấp.

Khả năng chịu lỗi: vì cơ sở dữ liệu lớp Serving được phân phối, và phải có khả năng xử lý các lỗi của máy.

#### **2.1.2.6. Ưu và nhược điểm của kiến trúc Lambda**

##### **2.1.2.6.1. Ưu điểm**

Độ bền và khả năng chịu lỗi. Bởi vì lớp batch được thiết kế để chỉ phần phụ, chứa toàn bộ tập dữ liệu kể từ thời điểm ban đầu, hệ thống có khả năng chịu lỗi do con người gây ra. Nếu có bất kỳ lỗi dữ liệu nào, thì tất cả dữ liệu từ điểm bị hỏng trở đi có thể bị xóa và thay thế bằng dữ liệu chính xác. Các batch view có thể được hoán đổi cho các lượt xem được tính toán lại hoàn toàn. Lớp Speed có thể bị loại bỏ. Trong thời gian cần thiết để tạo một tập hợp các batch view mới, toàn bộ hệ thống có thể được đặt lại và chạy lại.

Khả năng mở rộng. Kiến trúc Lambda được thiết kế với các lớp được xây dựng dưới dạng hệ thống phân tán. Chỉ cần thêm nhiều máy hơn, người dùng cuối có thể dễ dàng mở rộng các hệ thống đó theo chiều ngang. Khi các loại dữ liệu mới xâm nhập vào hệ thống dữ liệu, các khung nhìn mới sẽ trở nên cần thiết. Hệ thống dữ liệu không bị khóa vào một số loại hoặc một số batch view nhất định.

Sự khái quát. Vì Kiến trúc Lambda là một mô hình chung, những người chấp nhận không bị khóa vào một cách cụ thể để tính toán chế độ xem này hoặc chế độ xem hàng loạt đó. Chế độ xem hàng loạt và tính toán lớp tốc độ có thể được thiết kế để đáp ứng các nhu cầu cụ thể của hệ thống dữ liệu.

Truy vấn đặc biệt. Nếu cần, lớp Batch có thể hỗ trợ các truy vấn đặc biệt không có sẵn trong batch view. Giả sử độ trễ cao cho các truy vấn đặc biệt này là cho phép, thì tính hữu ích của lớp Batch không chỉ bị giới hạn ở các batch view mà nó tạo ra.

Bảo trì tối thiểu. Kiến trúc Lambda, trong hiện thân điển hình của nó, sử dụng Apache Hadoop cho lớp Batch và ElephantDB cho lớp Serving. Cả hai đều khá đơn giản để bảo trì.

Khả năng gỡ lỗi. Các đầu vào để tính toán các batch view của lớp Batch luôn giống nhau: toàn bộ tập dữ liệu. Trái ngược với các chế độ xem gỡ lỗi được tính toán trên ảnh chụp nhanh của một luồng dữ liệu, các đầu vào và đầu ra cho mỗi lớp trong Kiến trúc Lambda không phải là mục tiêu di động, giúp đơn giản hóa rất nhiều việc gỡ lỗi các tính toán và truy vấn.

Đọc và cập nhật dữ liệu có độ trễ thấp. Trong kiến trúc Lambda, thuộc tính cuối cùng của hệ thống dữ liệu lớn được thực hiện bởi lớp Speed, lớp này cung cấp các truy vấn thời gian thực về tập dữ liệu mới nhất.

##### **2.1.2.6.2. Nhược điểm**

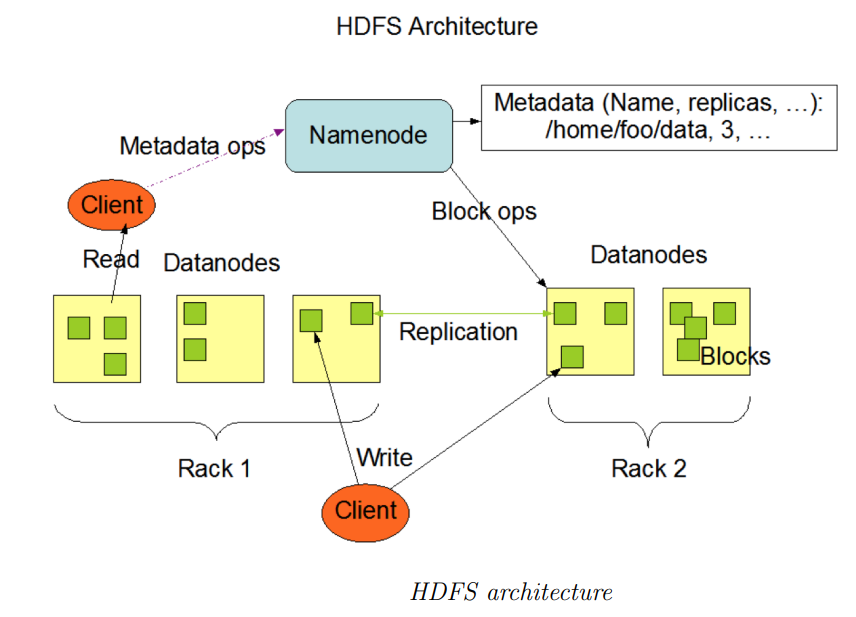
Mặc dù những ưu điểm của kiến trúc Lambda có vẻ rất nhiều và đơn giản, nhưng có một số nhược điểm cần lưu ý. Đầu tiên và quan trọng nhất, chi phí sẽ trở thành một yếu tố cần cân nhắc. Mặc dù cách mở rộng quy mô không quá phức tạp - chỉ cần thêm nhiều máy hơn - ta có thể thấy rằng lớp Batch nhất thiết sẽ cần phải mở rộng và phát triển theo thời gian. Vì tất cả dữ liệu chỉ là phần phụ và không có dữ liệu nào trong lớp Batch bị loại bỏ, chi phí mở rộng quy mô nhất thiết sẽ tăng theo thời gian.

Những người khác đã ghi nhận thách thức của việc duy trì hai bộ mã riêng biệt để tính toán các batch view và realtime view cho lớp Batch và lớp Speed. Cả hai lớp hoạt động trên cùng một tập hợp, trong trường hợp lớp Speed, tập hợp con - dữ liệu và các câu hỏi được hỏi của cả hai lớp là tương tự nhau. Tuy nhiên, vì hai lớp được xây dựng trên các hệ thống hoàn toàn khác nhau (ví dụ: Hadoop hoặc Snowflake cho lớp Batch, nhưng Storm hoặc Spark cho lớp Speed), việc bảo trì mã cho hai hệ thống riêng biệt có thể phức tạp.

## **2.2. Các framework sử dụng để xây dựng kiến trúc Lambda**

### **2.2.1. Lớp Batch**

#### **2.2.1.1. Hadoop Distributed File System (HDFS)**



Hình 10. Mô tả kiến trúc HDFS [6]

HDFS là một hệ thống tệp có khả năng chịu lỗi, phân tán và có thể mở rộng để lưu trữ và xử lý tập dữ liệu. HDFS quản lý dữ liệu được lưu trữ trên các cụm để tính toán bằng cách xử lý batch như các công cụ Hadoop hoặc Spark được triển khai trên cụm.

HDFS có kiến trúc master/ slave với hai loại nút: một NameNode duy nhất và nhiều DataNode. NameNode hoạt động như một máy chủ - master quản lý không gian trên hệ thống tệp và điều chỉnh quyền truy cập vào tệp bởi người dùng. DataNode quản lý bộ nhớ gắn liền với các nút mà chúng chạy trên đó. NameNode thực hiện các hoạt động như mở, đóng và đổi tên tệp và thư mục. Nó cũng xác định ánh xạ giữa các khối (block) đến DataNode. DataNode chịu trách nhiệm phục vụ việc đọc và ghi yêu cầu từ hệ thống tệp. Khi một tệp được tải lên HDFS, lần đầu tiên nó được chia thành các khối (block) có kích thước cố định từ 64 MB đến 256 MB. NameNode theo dõi ánh xạ từ tệp sang khối và vị trí mỗi khối được đặt. Khi bất kỳ ứng dụng nào xử lý tệp được lưu trữ trong HDFS, trước tiên nó sẽ truy vấn NameNode để lấy vị trí khối. Sau khi biết vị trí, ứng dụng sẽ liên hệ với DataNode trực tiếp để truy cập nội dung tệp (*Hình 10*) .

HDFS lưu trữ mỗi tệp dưới dạng một chuỗi các khối có cùng kích thước, ngoại trừ kích thước khối cuối cùng. Nó cung cấp khả năng chịu lỗi bằng cách sao chép các khối dữ liệu trên các DataNode khác nhau. Kích thước khối và hệ số nhân rộng có thể cấu hình được. NameNode xử lý việc sao chép các khối dữ liệu và định kỳ nhận được một tín hiệu (heartbeat) từ DataNode như một xác nhận rằng DataNode đang tồn tại và hoạt động và làm việc.

#### **2.2.1.2. Apache Spark**

Apache Spark là một framework tính toán phân cụm được xây dựng bởi ngôn ngữ Scala. Spark làm việc một cách nhanh chóng và tổng quát để xử lý dữ liệu song song với quy mô lớn. Spark mở rộng mô hình MapReduce. Có thể nói một trong những đặc điểm quan trọng của Spark là hoạt động trong bộ nhớ, và do đó cung cấp hiệu suất tốt hơn Hadoop trong hầu hết các trường hợp. Spark có thể chạy cục bộ hoặc theo cụm hoặc trên đám mây. Nó có thể chạy trên Hadoop Yarn, hoặc trên đám mây. Spark có thể truy cập dữ liệu từ các nguồn khác nhau như HDFS, Cassandra, HBase, S3...

Spark có kiến trúc master/worker, trong đó máy master có nhiệm vụ quản lý các nút worker thực thi các công việc và chức năng. Apache Spark cung cấp các thư viện khác nhau bao gồm Spark Core, Streaming, SQL, MLlib, GraphX.

Spark Core là phần cốt lõi của Spark và là công cụ cơ bản cho thư viện. Nó chịu trách nhiệm cho việc lập lịch tác vụ và cung cấp tính trừu tượng hiệu quả cho tính toán cụm trong bộ nhớ được gọi là Tập dữ liệu phân tán có khả năng phục hồi (Resilient Distributed Datasets). RDD là cốt lõi của Spark được thiết kế để hỗ trợ và lưu trữ dữ liệu trong bộ nhớ và xử lý phân tán trên một cụm trong khả năng chịu lỗi và cách làm hiệu quả. RDD đạt được khả năng chịu lỗi bằng cách sử dụng đồ thị xoay chiều có hướng (DAG) – có nhiệm vụ theo dõi dòng biến đổi được áp dụng cho dữ liệu trong quá trình xử lý. Hiệu quả đạt được trong RDD thông qua việc xử lý song song trên nhiều nút trong cụm và giảm thiểu sao chép dữ liệu giữa các nút.

Spark SQL phù hợp nhất để làm việc với dữ liệu có cấu trúc. Nó thúc đẩy công cụ truy vấn SQL, sang dữ liệu có cấu trúc lớn cho phép phân tích dữ liệu phức tạp hơn. Spark SQL hỗ trợ kết nối với JDBC, ODBC và Hive. Spark SQL có thể được tích hợp dễ dàng với các cơ sở dữ liệu hiện có, kho dữ liệu... Thực hiện Spark MLlib thường được sử dụng trong học máy và các thuật toán thống kê như phân loại, hồi quy, phân cụm, PCA… trên nền tảng phân tán. Công cụ tính toán đồ thị Spark GraphX cho phép người dùng tương tác xây dựng cấu trúc đồ thị ở quy mô lớn. Nó hỗ trợ phân tích và tính toán đồ thị dữ liệu lớn.

### **2.2.2. Lớp Speed**

#### **2.2.2.1. Apache Kafka**

Apache Kafka là một nền tảng streaming phân tán giúp xây dựng đường truyền dữ liệu streaming realtime có khả năng chịu lỗi. Kiến trúc Kafka là sự kết hợp của Message Queue và mô hình Publish-Subscribe, có khả năng mở rộng cao. Kafka được chạy dưới dạng một cụm trên một hoặc nhiều server. Các cụm Kafka lưu trữ các dữ liệu stream chúng dưới dạng các topic. Kafka chứa producers và consumers.

Những thành phần chính tạo nên kiến trúc Kafka như sau:

Message: là dữ liệu stream truyền từ producer đến consumer thông qua các topic tồn tại trong broker

Producer: nơi tạo ra các message stream và bỏ vào topic để dữ liệu được truyền đi

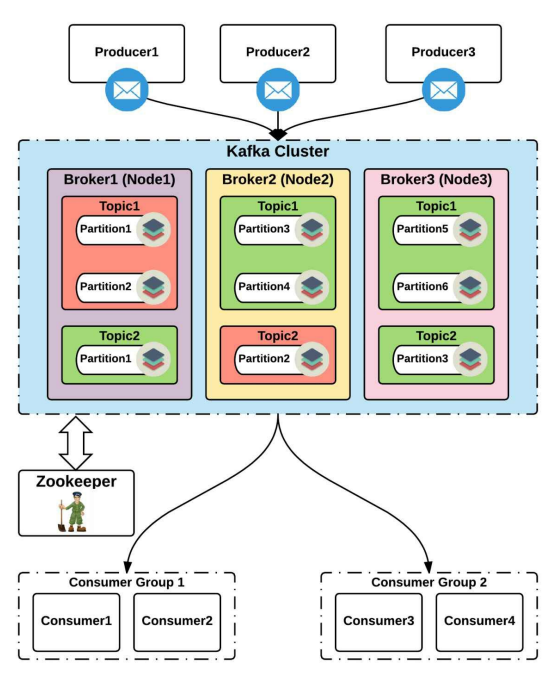
Topic: là một danh mục hoặc tên nguồn mà producer gửi message vào để truyền đi. Bất cứ khi nào người đăng kí (subcriber) muốn đọc topic này và nó sẽ nhận được tất cả messages. Đối với mỗi topic, Kafka duy trì một log phân vùng

Broker: các messages được tạo ra bởi producer được gắn với topic và topic được lưu trữ trong broker và consumer muốn truy cập những message từ topic thì phải thực hiện đăng kí đến topic trong broker và broker sẽ gửi message đến nó. Có thể nói, broker thực hiện trách nhiệm lưu trữ trong Kafka. Tìm kiếm message trong phân vùng và trong một broker là trách nhiệm của broker. Message được lưu trữ trong topic bao gồm nhiều phân vùng và các phân vùng này được sao chép đến nhiều broker trong cùng một cụm. Một cụm Kafka gồm nhiều máy chủ hay nút. Mỗi nút là một broker Kafka và chứa nhiều phân vùng để chứa nhiều topic.

Partition: Mỗi topic trong Kafka được phân vùng và mỗi phân vùng chứa các message được tạo ra bởi producer. Mỗi message được ghi vào phân vùng có tính chất tuần tự và đây là một trong những lý do khiến Kafka có hiệu suất cao. Khi ghi vào phân vùng topic, mỗi message được cung cấp một số thứ tự, được gọi là offset và đây là số định danh message duy nhất trong một phân vùng. Kafka có tính chất phân tán, nó thực hiện phân tán phân vùng topic vào nhiều broker để có khả năng chịu lỗi và duy trì message.

Consumer: như một yêu cầu gửi đến topic, để có thể đọc message từ topic. Các messages được gửi đến tuần tự từ bên trong phân vùng của topic.

Zookeeper: là một trong những thành phần rất quan trọng và bắt buộc để Kafka hoạt động. Zookeeper thực hiện những công việc sau: quản lý trạng thái của phân vùng và các bản sao, thực hiện cấu hình topic như số lượng phân vùng, quản lý vị trí của các bản sao…kiểm soát danh sách truy cập vào broker.



Hình 11. Mô tả kiến trúc Kafka [5]

*Hình 11* kiến trúc Kafka được quản lý bởi thành phần Zookeeper và các message được gửi từ producer đến consumer thông qua các topic được chứa trong blocker của hệ thống.

#### **2.2.2.2. Spark Streaming**

Spark Streaming là một thư viện Spark xử lý dữ liệu streaming. Nó cho phép khả năng mở rộng, thông lượng cao, khả năng chịu lỗi khi xử lý streaming với dữ liệu trực tiếp. Spark Streaming về bản chất không phải là framework streaming và do đó nó không xử lý stream dựa trên sự kiện. Thay vào đó, nó sử dụng Discretized stream (DStream) có thể đại diện cho nguồn dữ liệu đầu vào. DStream là một chuỗi các mini-batch, mỗi mini-batch được biểu diễn như một Spark RDD.



Hình 12. Mô tả thư viện Spark Streaming [6]

*Hình 12* thư viện Spark Streaming xử lý dữ liệu realtime từ Kafka và kết quả được đưa vào Database và cập nhật đến Dashboard.

### **2.2.3. Lớp Serving**

#### **2.2.3.1. Apache Cassandra**

Apache Cassandra là cơ sở dữ liệu phân tán mã nguồn mở có khả năng mở rộng và chịu lỗi cao. Cassandra có một kiến trúc masterless có nghĩa là nó có không có một điểm lỗi nào vì tất cả các nút đều được sao chép. Mỗi nút trong cụm là giống hệt nhau và có vai trò tương tự cho phép khả năng khả dụng cao và khả năng mở rộng vì bất kỳ nút nào cũng có thể phục vụ yêu cầu. Cassandra cung cấp phân phối dữ liệu tự động trên các nút trong cụm cơ sở dữ liệu.

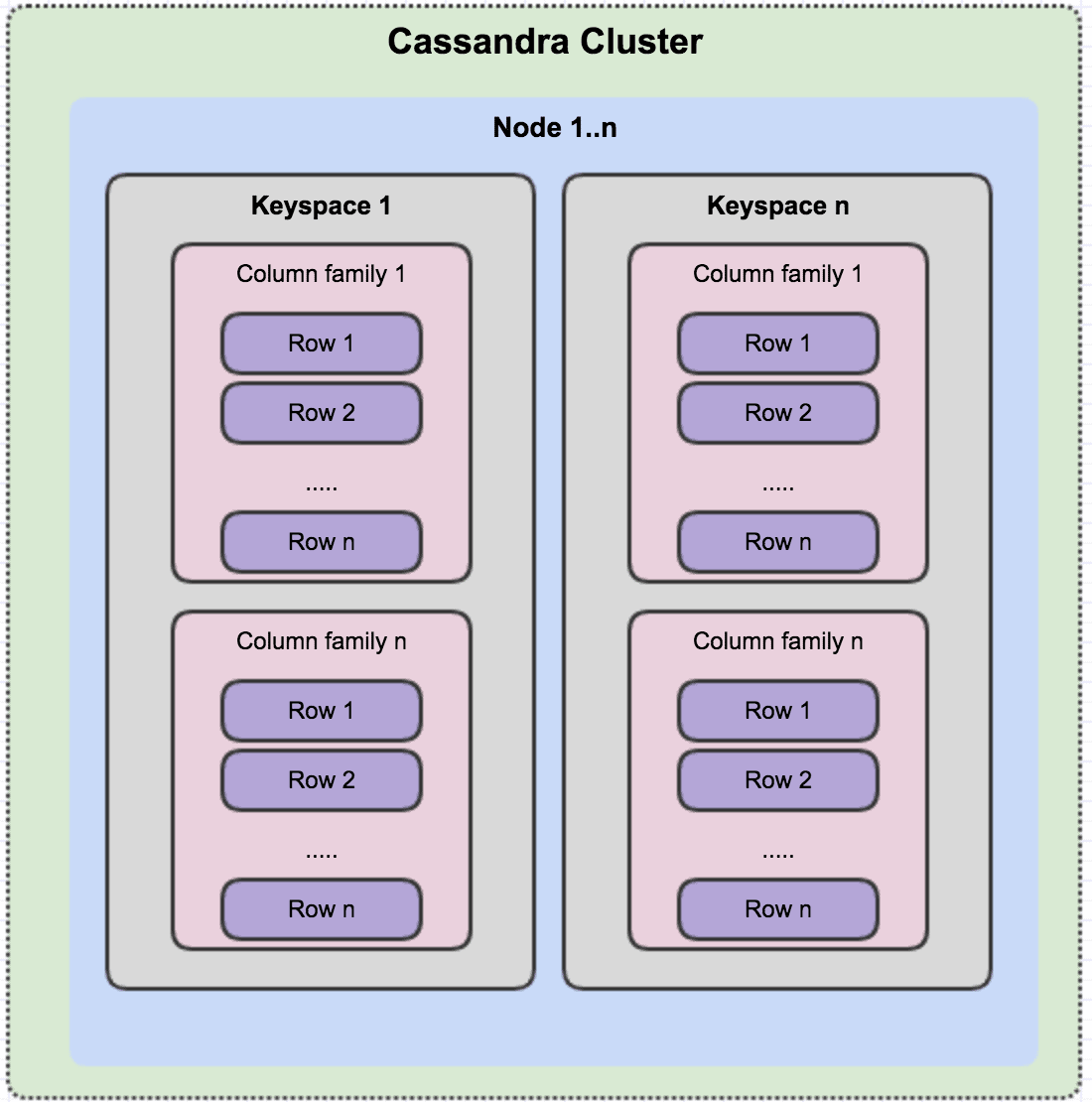
Những đặc trưng chính của Cassandra:

Phân cấp: dữ liệu được phân phối trên toàn bộ cụm. Không có nút chính. Mỗi nút đóng một vai trò quan trọng và giống hệt nhau.

Khả năng mở rộng: nó có thể chia tỷ lệ dữ liệu theo chiều ngang. Nó cung cấp các thao tác đọc và ghi ngẫu nhiên. Nó có thể tăng khi máy hoặc người dùng mới được tăng lên và luôn có thời gian hoạt động và không có chỗ cho thời gian chết hoặc xáo trộn các ứng dụng.

Khả năng chịu lỗi: trong cơ sở dữ liệu hướng cột, dữ liệu được sao chép sang nhiều nút để nó có thể được sử dụng để chịu lỗi. Nếu các nút bị lỗi do gián đoạn hoặc nhiều lý do khác không thành công, các nút có thể được thay thế

Nhân rộng: trong điều này, chúng ta có thể sao chép dữ liệu qua nhiều trung tâm. Dữ liệu được sao chép theo hướng nhiều nút để không xảy ra lỗi ngay cả ở một điểm duy nhất.

****

Hình 13. Kiến trúc Cassandra [7]

*Hình 13* kiến trúc Cassandra để lưu trữ cơ sở dữ liệu không có cấu trúc (NoSQL) có thành phần chính là keyspace, mỗi keyspace gồm có hệ số sao chép để chỉ định số lượng node trong cụm, sắp xếp bản sao và thành phần họ cột mô tả cấu trúc của dữ liệu, mỗi keyspace bao gồm một hoặc nhiều họ cột.

#### **2.2.3.2. Flask**

Flask là một micro web framework được viết bằng python. Flask cung cấp cho các lập trình viên khả năng tùy biến khi phát triển ứng dụng web, nó cung cấp cho bạn các công cụ, thư viện và cơ chế cho phép bạn xây dựng một ứng dụng web nhưng nó sẽ không thực thi bất kỳ sự phụ thuộc nào hoặc cho bạn biết dự án sẽ như thế nào.

Ứng dụng web có thể là blog, trang web thương mại hoặc một số trang web khác, nó vẫn cho phép các lập trình viên cơ hội sử dụng một số tiện ích mở rộng để thêm nhiều chức năng hơn cho ứng dụng web.

Ngoài ra, nhóm đã sử dụng HTML và Bootstrap để phát triển trang ứng dụng.

## **2.3. Xây dựng các lớp của kiến trúc Lambda**

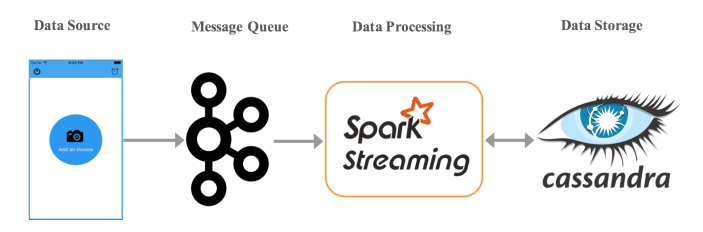
### **2.3.1. Xây dựng lớp Batch**

Dữ liệu lịch sử từ HDFS được Spark xử lý. Dữ liệu sau khi được xử lý sẽ được thêm vào Cassandra. Đối với mỗi người dùng, lưu thông tin UpVotes, DownVotes, Reputation, các chủ đề mà người dùng yêu thích. Công cụ dùng để xử lý dữ liệu trong lớp Batch là PySpark. Quá trình truyền dữ liệu từ HDFS vào Spark xử lý, dữ liệu từ file csv chuyển đổi dataframe. Tại đây thực hiện các truy vấn Spark SQL để tính toán.

Các bảng được lưu trữ trong Cassandra gồm:

1. userprofile - id, displayname, upvotes, downvotes, reputation
2. toptags - userid, tag

### **2.3.2. Xây dựng lớp Speed**



Hình 14. Mô tả quá trình xử lý dữ liệu Streaming [6]

*Hình 14* dữ liệu được tải từ Stack Overflow API được lưu về máy, tại đây Kafka producer đọc dữ liệu stream. Spark Streaming đóng vai trò như consumer và đọc dữ liệu từ Kafka topic. Việc xử lý dữ liệu stream được thực hiện tại Spark Streaming và kết quả truy vấn được lưu trữ tại Cassandra. Kết quả của quá trình này đại diện cho quá trình xử lý dữ liệu gần đây trong kiến trúc Lambda.

Xử lý dữ liệu gần đây giúp thực hiện cập nhật các số liệu thống kê về người dùng một cách chính xác, gợi ý các câu hỏi gần đây cho người dùng, xu hướng các thẻ được tìm kiếm nhiều.

Các bảng được lưu trữ trong Cassandra gồm:

1. trendingtags - updatedon, count, tagname
2. realtimequestions – tags, title

realtimereputations – id, reputation

### **2.3.3. Xây dựng lớp Serving**

Lớp Serving thực hiện các truy vấn. Tất cả các view được lưu trữ trên Cassandra và người dùng có thể truy vấn các view từ Dashboard. Dữ liệu được trình bày bằng Flask.



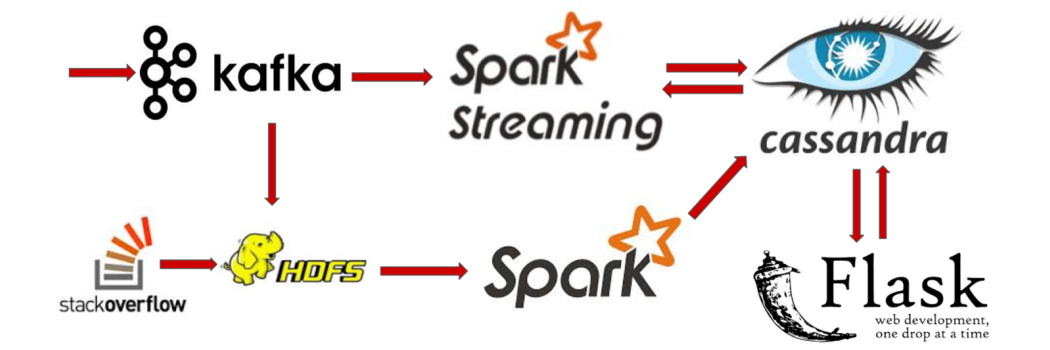
Hình 15. Mô tả việc trình bày dữ liệu bằng Flask [7]

## **2.4. Xây dựng Dashboard**

### **2.4.1. Tổng quát về ứng dụng**

Stack Overflow là diễn đàn hỏi và đáp với cộng đồng người dùng khổng lồ với hàng triệu bài viết mỗi năm. Bài nghiên cứu minh họa kiến trúc Lambda bằng cách xây dựng đường ống dữ liệu, để đề xuất câu hỏi cho mỗi hồ sơ người dùng dựa trên những chủ đề họ quan tâm nhiều, và hiển thị các số liệu thống kê như UpVotes, DownVotes, Reputation của người dùng, các chủ đề phổ biến hiện tại. Dự án thực hiện truy vấn thông xử lý batch và xử lý streaming, kết quả truy vấn được hiển thị qua Dashboard.

Mục tiêu của dự án là xây dựng một hệ thống dữ liệu hiệu quả về chi phí, có thể mở rộng và chịu lỗi dựa trên kiến trúc Lambda.



Hình 16. Tổng quát pipeline của ứng dụng [4]

*Hình 16* pipeline hiện thực các lớp trong kiến trúc Lambda. Lớp Batch (HDFS, Spark), lớp Speed (Kafka, Spark Streaming), lớp Serving (Cassandra, Flask).

### **2.4.2. Tập dữ liệu**

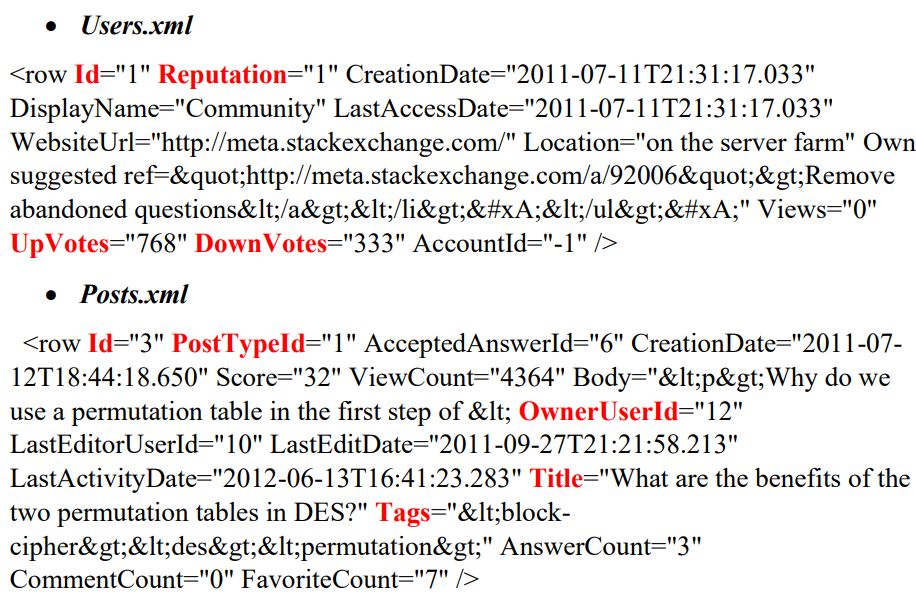
Để thực hiện kiến trúc Lambda, cần một tập dữ liệu có thể sử dụng để xử lý batch và realtime. Tập dữ liệu được sử dụng trong ứng dụng là tập dữ liệu về Stack Overflow.

Tập dữ liệu lịch sử để dùng cho lớp Batch:

<https://archive.org/details/stackexchange>

Tập dữ liệu realtime dùng cho lớp Speed:

<https://api.stackexchange.com/>



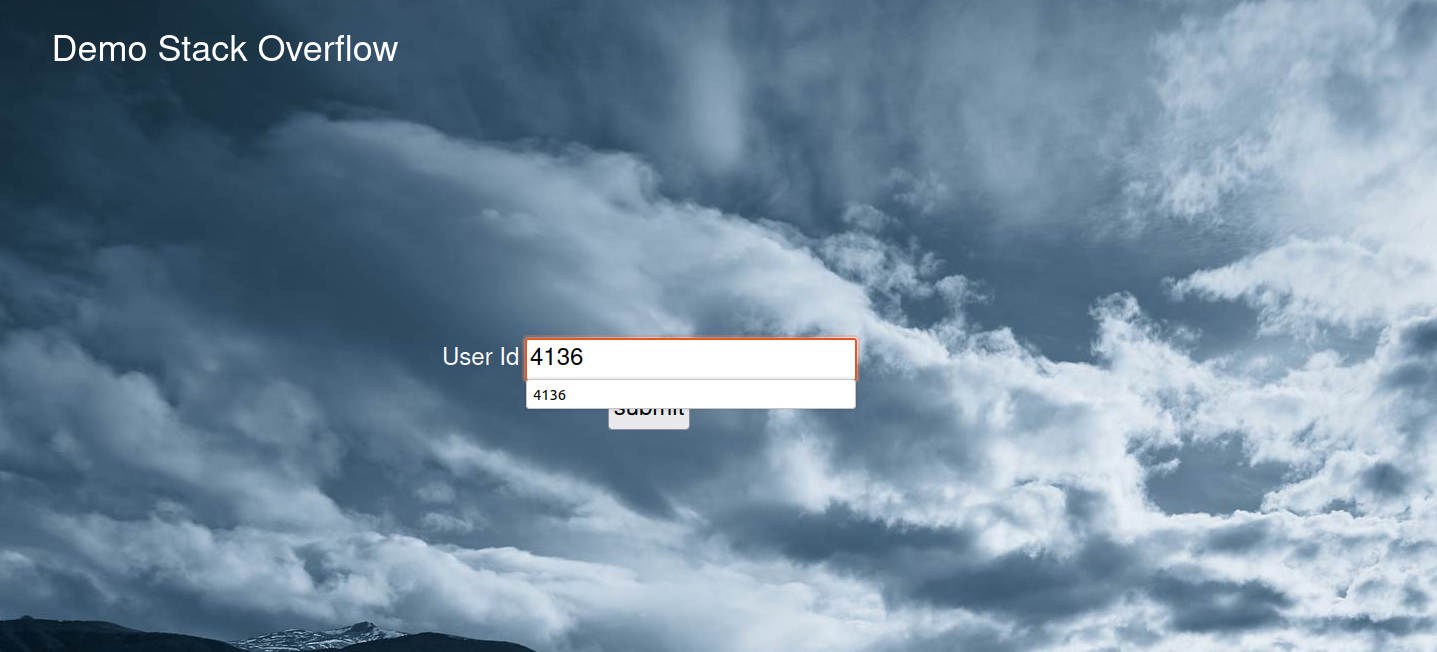
Hình 17. Tập dữ liệu Stack Overflow

*Hình 17*  tập dữ liệu dạng file XML khi được lưu trữ trong HDFS sẽ chuyển thành các dataframe và xử lý dưới dạng file csv.

Tập dữ liệu sẽ hỗ trợ về truy vấn và lưu trữ dữ liệu về thông tin người dùng như:

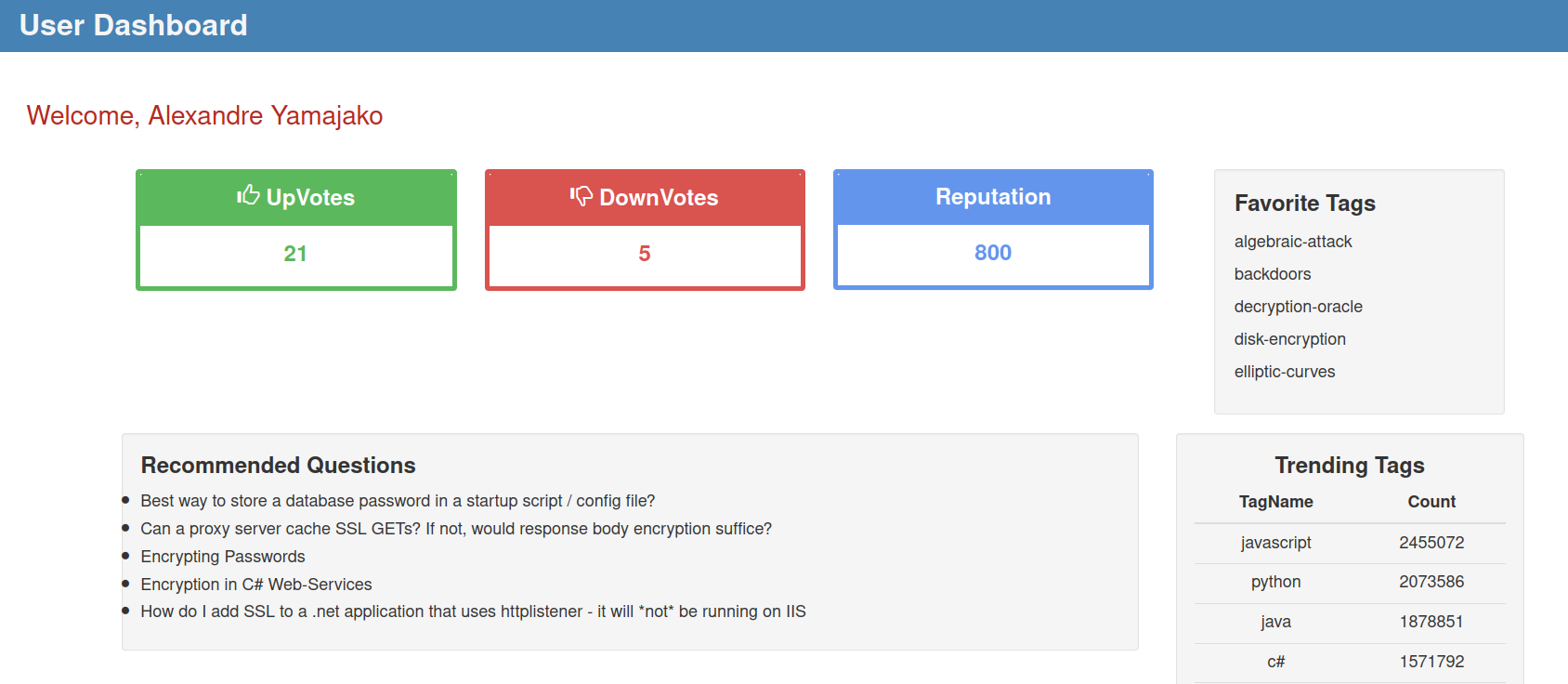
* Xác định các Tag nổi bật cho mỗi người dùng.
* Cập nhật thông tin về UpVotes, DownVotes, Reputation trong thời gian thực.
* Đề xuất câu hỏi trong thời gian thực bởi các Tag nổi bật
* Xu hướng các Tag đang được tìm kiếm nhiều và cập nhật số lượng trong thời gian thực.

### **2.4.3. Kết quả Dashboard**



Hình 18. Trang đăng nhập bằng User ID

Hình 18 cho phép người dùng đăng nhập vào trang Dashboard để xem các thông tin như số lượng UpVotes, DownVotes, Reputation, Recommended Question, số lượng Favorite Tags và Trending Tags.



Hình 19. Trang chủ Dashboard của User tương ứng

Hình 19 các thông tin như số lượng UpVotes, DownVotes, Reputation là kết quả của việc xử lý dữ liệu lịch sử của lớp Batch, các thông tin như Favorite Tags, Trending Tags là kết quả của quá trình xử lý dữ liệu realtime của lớp Speed và các thông tin này sẽ thay đổi theo thời gian khi có dữ liệu mới được cập nhật đến và kết quả của phần Recommended Questions sẽ thay đổi tùy theo các Trending Tags và Favorite Tags.

# **Phần 3: KẾT LUẬN**

## **3.1. Kết quả đạt được**

Hiểu cấu trúc, cách hoạt động và cách xây dựng kiến trúc Lambda.

Sử dụng các công cụ để xử lý dữ liệu hng loạt và dữ liệu stream.

## **3.2. Hạn chế**

Tập dữ liệu được sử dụng nhỏ và ứng dụng hướng đến với việc mô phỏng nhiều hơn, việc xây dựng một ứng dụng Big data chưa được thực tế.

## **3.3. Hướng phát triển**

Hướng đến việc xây dựng một hệ thống dữ liệu tối ưu hơn, kết hợp việc sử dụng học máy để đưa ra kết quả truy vấn tốt hơn.

# **TÀI LIỆU THAM KHẢO**

|  |  |
| --- | --- |
| [1] | "Apache kafka topics and logs," 2016. [Online]. Available: https://kafka.apache.org/intro.html. |
| [2] | "Hdfs architecture guide," 2008. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs\_design.html. |
| [3] | "Spark SQL Guide," 2016. [Online]. Available: https://spark.apache.org/docs/latest/sql-programming-guide.html. |
| [4] | "Spark Streaming vs. Kafka Streaming," [Online]. Available: https://dzone.com/articles/spark-streaming-vs-kafka-stream-1. |
| [5] | "WHY HADOOP IS IMPORTANT IN HANDLING BIG DATA?," [Online]. Available: https://blog.bigdataweek.com/2016/08/01/hadoop-important-handling-big-data/. |
| [6] | J. S. Blog, "What is the Lambda Architecture?," 5 8 2016. [Online]. Available: https://www.jamesserra.com/archive/2016/08/what-is-the-lambda-architecture/. |
| [7] | T. Grall, "Lambda Architecture: The Best Way to Build Scalable and Reliable Applications!," 9 2 2016. [Online]. Available: https://speakerdeck.com/tgrall/lambda-architecture-the-best-way-to-build-scalable-and-reliable-applications. |
| [8] | A. Huilgol, "Lambda Architecture for Real-time data processing," 30 5 2017. [Online]. Available: https://medium.com/@hsamogh/lambda-architecture-for-real-time-data-processing-62294cdf2b3. |
| [9] | P. S. R. Matli, "Apache Cassandra Data Modeling and Query Best Practices," 7 2 2019. [Online]. Available: https://www.red-gate.com/simple-talk/databases/nosql/apache-cassandra-data-modeling-and-query-best-practices/. |
| [10] | J. W. Nathan Marz, Big Data: Principles and best practices of scalable realtime data systems, Manning, 2015. |
| [11] | Pav, "Highly available Kafka cluster in Docker," 14 12 2016. [Online]. |
| [12] | P. Pedamkar, "What is Cassandra?," 2022. [Online]. Available: https://www.educba.com/what-is-cassandra/. |
| [13] | A. Ulyanov, "The “Lambda” Architecture for near real-time data processing," 16 7 2016. [Online]. Available: https://cloudgeometry.io/blog/the-lambda-architecture/. |