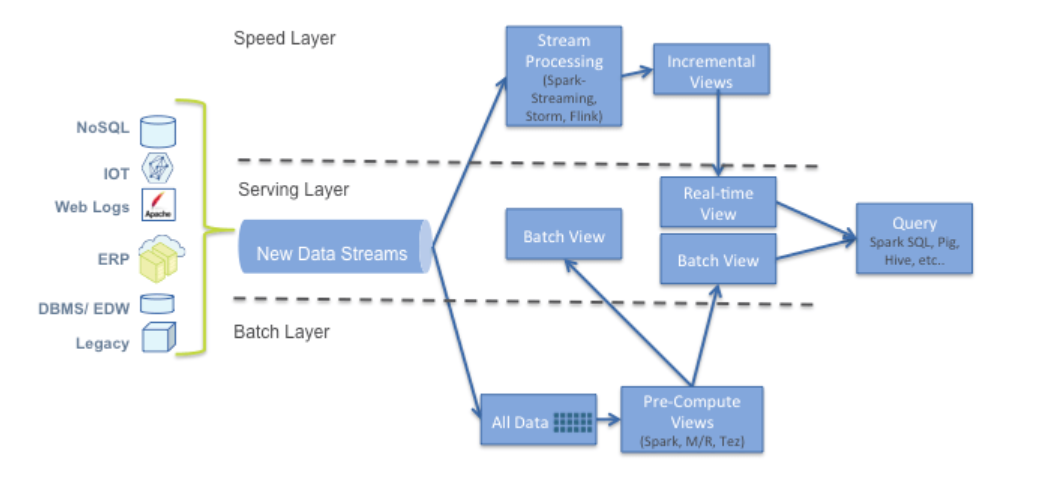
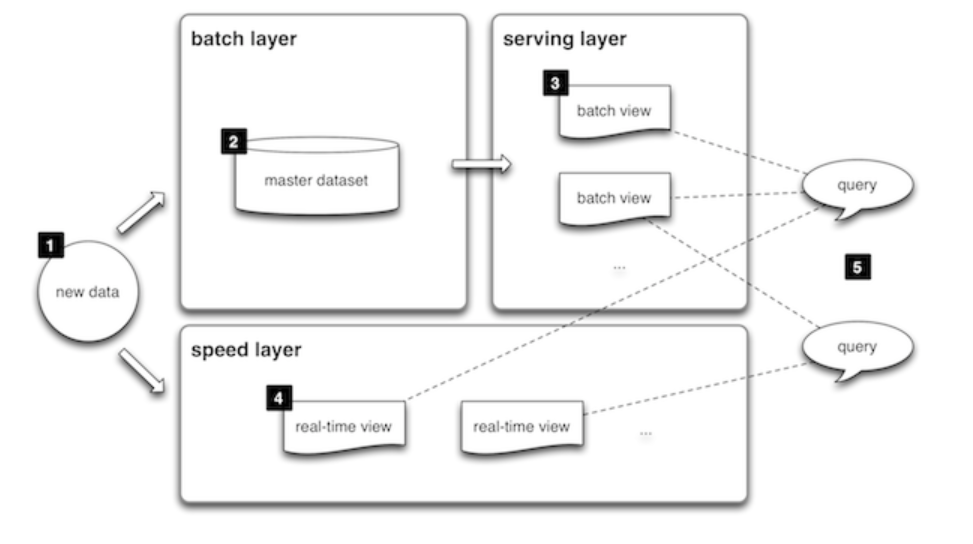
Lambda architecture là một kiến trúc xử lý dữ liệu được giới thiệu bởi Nathan Marz [1] bằng cách kết hợp các ưu điểm của cả batch processing và stream-processing. Lambda architecture được chia thành 3 layers: Batch layer, Speed layer và Serving layer.



Batch layer đóng vai trò như một ‘data lake’ để lưu trữ toàn bộ dữ liệu thu thập được và xử lý dữ liệu này dưới dạng batch processing (chúng ta cần thiết lập khoảng thời gian cho từng batch như 30min, 3 hours, 1 day,…). Việc sử dụng Batch processing đem lại ưu điểm cho Lambda Architecture vì dữ liệu thu về có thể bị trùng lặp hoặc có thể chứa nhiều thông tin rác do đó cần phải có các bước xử lý và làm sạch dữ liệu. Bên cạnh đó dữ liệu cũng có thể đến trễ do sự cố đường truyền hoặc bị thu thập muộn hơn so với thời gian đăng tải. Ngoài ra kết quả mỗi lần chạy Batch processing sẽ được cập nhật thường xuyên, từ đó giúp nâng cao độ chính xác và sửa kết quả lỗi của Real-time processing trong các lần chạy trước.

Speed layer có nhiệm vụ xử lý thông tin trong thời gian thực, đóng vai trò hỗ trợ cho Batch layer do Batch layer có độ trễ lớn và không đáp ứng được việc đưa ra kết quả xử lý cho dữ liệu mới nhận về. Tuy nhiên kết quả của Speed layer thường không có độ chính xác như ở Batch layer do hạn chế về thời gian xử lý. Ở layer này chúng ta có thể sử dụng các công cụ như Apache Storm, Apache Spark hoặc Apache Flume,…

Serving layer có vai trò lưu trữ các outputs của Batch layer và Speed layer. Ở layer này chúng ta có thể sử dụng một số công cụ như Apache Cassandra, MongoDB hoặc ElasticSearch,…



1. Tất cả các dữ liệu nhận về đều được đưa vào cả Batch layer và Speed layer

2. Batch layer thực hiện hai chức năng: lưu trữ dữ liệu và tiến hành xử lý dữ liệu với đầu ra là các batch views

3. Serving layer lập chỉ mục (indexing) cho các batch views để có thể truy cập các kết quả này một cách nhanh chóng

4. Speed layer chỉ xử lý các thông tin mới nhất trong thời gian thực để tạo ra các real-time views

5. Hệ thống trả về kết quả cho các truy vấn của người dùng bằng cách kết hợp cả batch views và real-time views.

Tại thời điểm này, bạn đã học được cách tính toán trước các chế độ xem tùy ý của bất kỳ tập dữ liệu nào bằng cách

sử dụng tính toán hàng loạt. Để các chế độ xem hữu ích, bạn phải có khả năng

truy cập nội dung của chúng với độ trễ thấp và như trong hình 10.1, đây là vai trò của

lớp phục vụ. Lớp phân phát lập chỉ mục các chế độ xem và cung cấp các giao diện sao cho

Dữ liệu được tính toán trước có thể được truy vấn nhanh chóng.

Serving layer là thành phần cuối cùng của phần batch của Lambda Architecture. Nó được gắn chặt vào lớp lô vì lớp lô chịu trách nhiệm về

liên tục cập nhật các chế độ xem lớp phân phát. Những quan điểm này sẽ luôn lỗi thời đến hạn

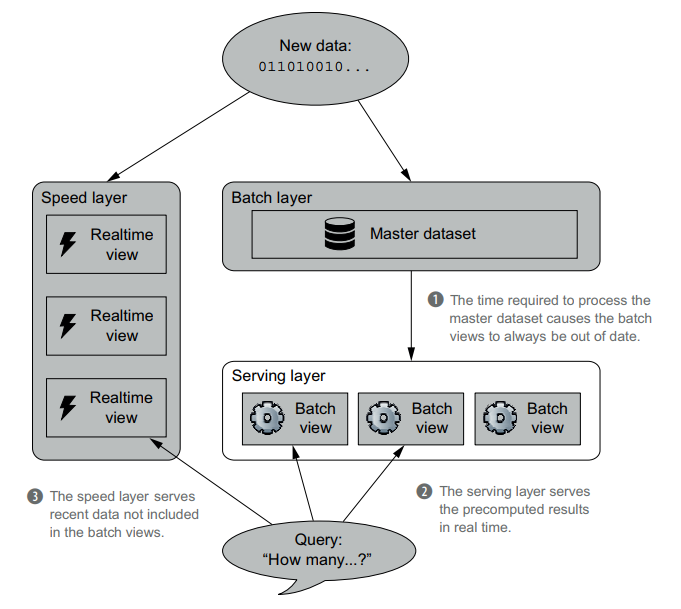
đến bản chất độ trễ cao của tính toán hàng loạt. Nhưng đây không phải là một mối quan tâm, bởi vì

Lớp tốc độ sẽ chịu trách nhiệm cho bất kỳ dữ liệu nào chưa có sẵn trong lớp phục vụ.

Trong Kiến trúc Lambda, lớp phân phối cung cấp quyền truy cập có độ trễ thấp vào

Kết quả tính toán được thực hiện trên tập dữ liệu chính. Các chế độ xem lớp phục vụ là

hơi lỗi thời do thời gian cần thiết cho tính toán hàng loạt



Thật không may, lớp phục vụ là một khu vực mà công cụ tụt hậu so với lý thuyết. Nó

Sẽ không khó để xây dựng một triển khai lớp phục vụ có mục đích chung — trên thực tế, nó

sẽ dễ dàng hơn đáng kể so với việc xây dựng bất kỳ cơ sở dữ liệu NoSQL nào hiện có. Chúng tôi sẽ trình bày lý thuyết đầy đủ đằng sau việc tạo ra một đơn giản, có thể mở rộng, chịu lỗi,

và lớp phục vụ đa năng, và sau đó chúng tôi sẽ sử dụng công cụ tốt nhất hiện có để

chứng minh các khái niệm cơ bản.

Trong khi điều tra lớp phục vụ, bạn sẽ tìm hiểu những điều sau:

■ Lập chỉ mục các chiến lược để giảm thiểu độ trễ, sử dụng tài nguyên và phương sai

■ Các yêu cầu đối với lớp phân phối trong Kiến trúc Lambda

■ Cách lớp phục vụ giải quyết vấn đề chuẩn hóa được tranh luận từ lâu so với vấn đề không chuẩn hóa

Chúng ta sẽ bắt đầu bằng cách kiểm tra các vấn đề chính mà bạn sẽ gặp phải khi cấu trúc một lớp phục vụ cảnh.

1. Chỉ số hiệu suất cho lớp phân phát

Như với lớp lô, lớp phục vụ được phân phối giữa nhiều máy để có khả năng mở rộng. Các chỉ mục của lớp phục vụ được tạo, tải và phân phối theo cách phân phối đầy đủ

Khi thiết kế các chỉ mục này, bạn phải xem xét hai chỉ số hiệu suất chính:

thông lượng và độ trễ. Trong ngữ cảnh này, độ trễ là thời gian cần thiết để trả lời một câu hỏi duy nhất

truy vấn, trong khi thông lượng là số lượng truy vấn có thể được phục vụ trong một số truy vấn nhất định

khoảng thời gian. Mối quan hệ giữa cấu trúc của các chỉ mục lớp phục vụ

và các số liệu này được giải thích tốt nhất thông qua một ví dụ

Chúng tôi sẽ quay lại ngắn gọn về ví dụ SuperWebAnalytics.com dài hạn của mình—cụ thể là truy vấn số lần xem trang theo thời gian. Mục tiêu là phục vụ số lượng

số lần xem trang cho mỗi giờ được cung cấp một URL cụ thể và một phạm vi giờ cụ thể. Để đơn giản hóa hơn nữa cuộc thảo luận, giả sử số lần xem trang chỉ được tạo bằng cách sử dụng

độ chi tiết hàng giờ. Chế độ xem kết quả sẽ trông tương tự như hình 10.2.

Một cách đơn giản để lập chỉ mục chế độ xem này là sử dụng chiến lược khóa/giá trị với

[URL, giờ] ghép nối dưới dạng khóa và số lần xem trang dưới dạng giá trị. Chỉ mục sẽ được phân vùng

sử dụng khóa, vì vậy số lần xem trang cho cùng một URL sẽ nằm trên các phân vùng khác nhau.

Các phân vùng khác nhau sẽ tồn tại trên các máy chủ riêng biệt, vì vậy hãy truy xuất một loạt giờ cho

một URL duy nhất sẽ liên quan đến việc tìm nạp giá trị từ nhiều máy chủ trong lớp phân phát của bạn

Mặc dù thiết kế này hoạt động về nguyên tắc, nhưng nó phải đối mặt với các vấn đề nghiêm trọng với cả độ trễ

và thông lượng. Để bắt đầu, độ trễ sẽ luôn cao. Bởi vì các giá trị

đối với một URL cụ thể được trải rộng khắp cụm của bạn, bạn sẽ cần truy vấn nhiều máy chủ để có được số lần xem trang trong một phạm vi lớn trong vài giờ. Quan sát chính

là thời gian phản hồi của các máy chủ khác nhau. Ví dụ: một máy chủ có thể nhiều hơn một chút

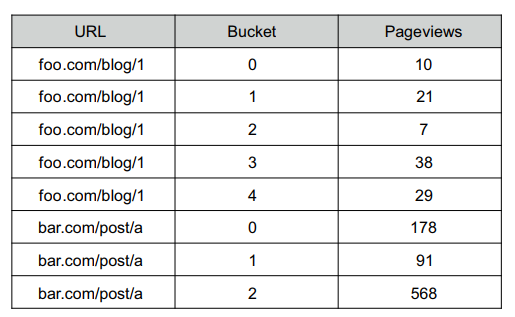
tải hơn những người khác; một người khác có thể đang thực hiện thu gom rác vào thời điểm đó.

Ngay cả khi bạn làm song song các yêu cầu tìm nạp, thời gian phản hồi truy vấn tổng thể bị giới hạn bởi

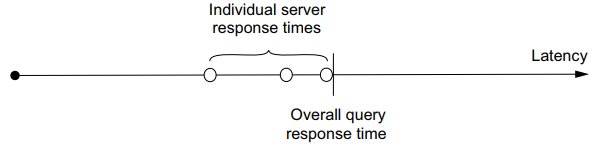
Tốc độ của máy chủ chậm nhất.

Để minh họa điểm này, giả sử một truy vấn yêu cầu tìm nạp dữ liệu từ ba máy chủ.

Một mẫu đại diện cho sự phân bố thời gian phản hồi được thể hiện trong hình 10.3



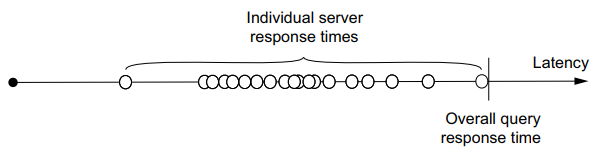
Hình 10.2 Số lần xem trang theo thời gian hàng loạt với độ chi tiết hàng giờ



Hình 10.3 Khi phân phối một tác vụ trên nhiều máy chủ, độ trễ tổng thể là được xác định bởi thời gian phản hồi của máy chủ chậm nhất.

Để so sánh, giả sử truy vấn chạm vào 20 máy chủ. Một phân phối điển hình của độ trễ

sẽ trông giống như hình 10.4.



Hình 10.4 Nếu bạn tăng số lượng máy chủ liên quan đến một tác vụ phân tán,

Bạn cũng tăng khả năng ít nhất một người sẽ phản hồi chậm.

Nói chung, truy vấn càng chạm vào nhiều máy chủ, độ trễ tổng thể của truy vấn càng cao

Truy vấn. Điều này là do thực tế đơn giản là liên quan đến nhiều máy chủ hơn làm tăng khả năng ít nhất một máy chủ sẽ phản hồi chậm. Do đó, phương sai của máy chủ

Thời gian phản hồi biến hiệu suất trường hợp xấu nhất của một máy chủ thành trường hợp phổ biến

thực hiện các truy vấn. Đây là một vấn đề nghiêm trọng để đạt được độ trễ tốt cho

truy vấn số lần xem trang theo thời gian

Một vấn đề khác với chiến lược khóa/giá trị này là thông lượng kém, đặc biệt nếu

Máy chủ của bạn sử dụng đĩa chứ không phải ổ đĩa trạng thái rắn. Truy xuất giá trị cho một khóa duy nhất

Yêu cầu tìm kiếm ổ đĩa và một truy vấn duy nhất có thể tìm nạp giá trị cho hàng chục khóa trở lên. Đĩa

tìm kiếm là những hoạt động tốn kém cho các ổ cứng truyền thống. Bởi vì có một hữu hạn

số lượng ổ đĩa trong cụm của bạn, có một giới hạn cứng đối với số lượng ổ đĩa tìm kiếm

có thể đạt được mỗi giây. Giả sử rằng trung bình một truy vấn tìm nạp 20 khóa trên mỗi

truy vấn, cụm có 100 đĩa và mỗi đĩa có thể thực hiện 500 tìm kiếm mỗi giây. Trong

Trong trường hợp này, cụm của bạn chỉ có thể phục vụ 2.500 truy vấn mỗi giây—một con số nhỏ đáng ngạc nhiên

số lượng cho số lượng đĩa.

Nhưng tất cả không bị mất — một chiến lược lập chỉ mục khác có độ trễ tốt hơn nhiều và

đặc điểm thông lượng. Ý tưởng là sắp xếp thông tin số lần xem trang cho một URL duy nhất trên cùng một phân vùng và lưu trữ tuần tự. Tìm nạp số lần xem trang sẽ

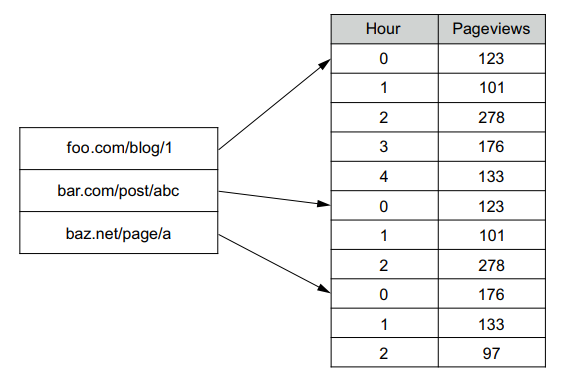
sau đó chỉ yêu cầu một tìm kiếm và quét duy nhất thay vì nhiều tìm kiếm. Quét là

cực kỳ rẻ so với tìm kiếm, vì vậy điều này tiết kiệm tài nguyên hơn nhiều. Ngoài ra

Chỉ cần một máy chủ duy nhất được liên hệ cho mỗi truy vấn, vì vậy bạn không còn phải tuân theo

Các vấn đề về phương sai của chiến lược trước đó. Bố cục của chỉ số cho chiến lược này là

thể hiện trong hình 10.5.



Hình 10.5 Chỉ mục được sắp xếp thúc đẩy quá trình quét và giới hạn đĩa tìm cách cải thiện cả độ trễ và Thông qua.

Hai ví dụ này chứng minh rằng cách bạn cấu trúc các chỉ mục lớp phân phát của mình

có ảnh hưởng đáng kể đến hiệu suất truy vấn của bạn. Một lợi thế sống còn của

Kiến trúc Lambda cho phép bạn điều chỉnh lớp phân phối cho các truy vấn

phục vụ, để tối ưu hóa hiệu quả.

1. Giải pháp lớp phục vụ cho việc chuẩn hóa / vấn đề không chuẩn hóa

Lớp phục vụ giải quyết một trong những vấn đề tồn tại lâu dài trong cơ sở dữ liệu quan hệ

thế giới: tình thế tiến thoái lưỡng nan về bình thường hóa so với không chuẩn hóa. Để nắm bắt giải pháp và

ý nghĩa của nó, trước tiên bạn cần hiểu các vấn đề cơ bản.

Quyết định bình thường hóa so với không chuẩn hóa cuối cùng là sự lựa chọn giữa

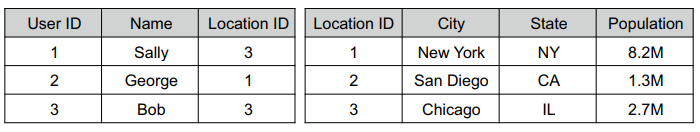
sự đánh đổi không thể chấp nhận được. Trong thế giới quan hệ, bạn muốn lưu trữ dữ liệu của mình được chuẩn hóa hoàn toàn; Điều này liên quan đến việc xác định mối quan hệ giữa các bộ dữ liệu độc lập để giảm thiểu sự dư thừa. Thật không may, việc truy vấn dữ liệu chuẩn hóa có thể chậm, vì vậy bạn có thể cần lưu trữ một số thông tin dư thừa để cải thiện thời gian phản hồi. Quá trình không chuẩn hóa này làm tăng hiệu suất, nhưng nó đi kèm với sự phức tạp rất lớn của giữ cho dữ liệu dư thừa nhất quán.

Để minh họa sự căng thẳng này, giả sử bạn đang lưu trữ thông tin vị trí người dùng trong các bảng quan hệ, như trong hình 10.6. Mỗi vị trí có một số nhận dạng và mỗi người

sử dụng một trong những số nhận dạng đó để chỉ ra vị trí của chúng. Truy vấn để truy xuất vị trí

Đối với một cá nhân cụ thể yêu cầu một nối giữa hai bảng. Đây là một ví dụ về một

lược đồ chuẩn hóa hoàn toàn, vì không có thông tin nào được lưu trữ dư thừa.



Hình 10.6 Một lược đồ chuẩn hóa sử dụng nhiều bộ dữ liệu độc lập với ít hoặc không có dữ liệu dư thừa

Bây giờ giả sử bạn quan sát thấy rằng việc truy xuất thành phố và tiểu bang cho người dùng là một

hoạt động cực kỳ phổ biến trong ứng dụng của bạn. Tham gia rất tốn kém, và bạn quyết định rằng bạn cần hiệu suất tốt hơn từ thao tác này. Các phương tiện duy nhất để tránh tham gia sẽ là lưu trữ dư thừa thông tin thành phố và tiểu bang trong bảng người dùng.

Kỹ thuật lưu trữ thông tin dư thừa này để tránh nối được gọi là không chuẩn hóa và lược đồ kết quả trong trường hợp này sẽ giống với hình 10.7.

Không chuẩn hóa không phải là một giải pháp lý tưởng — với tư cách là nhà phát triển ứng dụng, đó là của bạn

trách nhiệm đảm bảo tất cả các dữ liệu dư thừa là nhất quán. Điều này làm tăng sự khó chịu

các câu hỏi, chẳng hạn như "Điều gì sẽ xảy ra nếu các bản sao khác nhau của một trường trở nên không nhất quán? Ngữ nghĩa của dữ liệu trong trường hợp này là gì? Hãy nhớ rằng sai lầm là

không thể tránh khỏi trong các hệ thống tồn tại lâu dài, vì vậy với đủ thời gian, sự không nhất quán sẽ xảy ra

May mắn thay, sự phân chia giữa tập dữ liệu chính và lớp phục vụ trong

Lambda Architecture giải quyết vấn đề chuẩn hóa so với không chuẩn hóa.

Trong lớp hàng loạt, bạn có thể chuẩn hóa tập dữ liệu chính của mình theo nội dung trái tim của bạn.

Tính toán trên lớp hàng loạt đọc hàng loạt tập dữ liệu chính, vì vậy không có

cần thiết kế lược đồ để tối ưu hóa cho các lần đọc truy cập ngẫu nhiên. Bổ sung,

Lớp phân phát hoàn toàn phù hợp với các truy vấn mà nó phục vụ, vì vậy bạn có thể tối ưu hóa dưới dạng

cần thiết để đạt được hiệu suất tối đa. Những tối ưu hóa này trong lớp phục vụ có thể

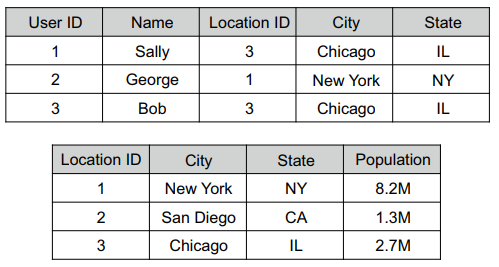
vượt xa sự không chuẩn hóa. Ngoài dữ liệu nối trước, bạn có thể thực hiện tổng hợp và chuyển đổi bổ sung để cải thiện hơn nữa hiệu quả

Đối với câu hỏi về tính nhất quán trong Kiến trúc Lambda, điều đó hoàn toàn đúng

thông tin đó sẽ được lưu trữ dư thừa giữa các lớp lô và lớp phục vụ. Các

Điểm khác biệt chính là lớp phục vụ được định nghĩa là một hàm của tập dữ liệu chính. Nếu một lỗi gây ra sự không nhất quán, bạn có thể dễ dàng sửa chúng bằng cách tính toán lại

lớp phục vụ từ đầu.



Hình 10.7 Các bảng không chuẩn hóa lưu trữ dữ liệu dư thừa để cải thiện hiệu suất truy vấn.

1. Các bảng không chuẩn hóa lưu trữ dữ liệu dư thừa để cải thiện hiệu suất truy vấn.

Kiến trúc Lambda đặt ra một bộ yêu cầu nhất định trên cơ sở dữ liệu lớp phân phát. Nhưng những gì không bắt buộc của cơ sở dữ liệu lớp phục vụ thú vị hơn nhiều so với những gì được yêu cầu. Đầu tiên, đây là những yêu cầu:

Batch writable — Chế độ xem hàng loạt cho một lớp phục vụ được tạo ra từ đầu.

Khi một phiên bản mới của một chế độ xem có sẵn, phải có thể hoán đổi hoàn toàn phiên bản cũ hơn với chế độ xem được cập nhật.

Scalable — Cơ sở dữ liệu lớp phân phát phải có khả năng xử lý các chế độ xem có kích thước tùy ý. Như với các hệ thống tệp phân tán và khung tính toán hàng loạt. Đã thảo luận trước đây, điều này đòi hỏi nó phải được phân phối trên nhiều máy.

Random reads —Cơ sở dữ liệu lớp phân phát phải hỗ trợ đọc ngẫu nhiên, với

các chỉ mục cung cấp quyền truy cập trực tiếp vào các phần nhỏ của chế độ xem. Yêu cầu này

là cần thiết để có độ trễ thấp trên các truy vấn.

Khả năng chịu lỗi — Vì cơ sở dữ liệu lớp phân phối được phân phối nên cơ sở dữ liệu đó phải chịu được các lỗi máy.

Hy vọng rằng, không có gì trong danh sách này là một bất ngờ. Nhưng một yêu cầu thông thường còn thiếu trong danh sách này — một yêu cầu tiêu chuẩn trên tất cả các cơ sở dữ liệu quen thuộc — là ghi ngẫu nhiên. Như vậy

chức năng hoàn toàn không liên quan đến lớp phục vụ vì các chế độ xem chỉ

sản xuất với số lượng lớn. Để rõ ràng, các ghi ngẫu nhiên tồn tại trong Kiến trúc Lambda,

Nhưng chúng bị cô lập trong lớp tốc độ để đạt được các bản cập nhật có độ trễ thấp. Cập nhật lên

Lớp phân phối tạo ra toàn bộ các dạng xem mới, vì vậy cơ sở dữ liệu lớp phân phát sẽ làm được

không cần khả năng sửa đổi các phần nhỏ của chế độ xem hiện tại.

Đây là một kết quả đáng kinh ngạc, bởi vì các lần ghi ngẫu nhiên chịu trách nhiệm cho đa số

về sự phức tạp trong cơ sở dữ liệu — và thậm chí còn phức tạp hơn trong cơ sở dữ liệu phân tán.

Ví dụ, hãy xem xét một trong những chi tiết khó chịu được thảo luận trong chương 1 về cách hoạt động của cơ sở dữ liệu ngẫu nhiên: sự cần thiết phải nén chặt để lấy lại không gian chưa sử dụng. Một chuyên sâu hoạt động, nén đôi khi hút đi nhiều tài nguyên của máy. Nếu đó là không được quản lý chính xác, máy móc sẽ trở nên quá tải và lỗi xếp tầng trở nên có khả năng khi tải được chuyển sang các máy khác.

Bởi vì lớp phân phát không yêu cầu ghi ngẫu nhiên, nó không yêu cầu trực tuyến

nén chặt, vì vậy sự phức tạp này, cùng với gánh nặng hoạt động liên quan của nó, hoàn toàn

biến mất trong lớp phục vụ. Tầm quan trọng của điều này được phóng đại khi bạn xem xét

Kích thước tương đối của lớp phục vụ và cụm lớp tốc độ của bạn. Lớp phục vụ đại diện cho

Các chế độ xem cho phần lớn tập dữ liệu chính của bạn, có thể trên 99%, vì vậy nó yêu cầu

phần lớn tài nguyên cơ sở dữ liệu. Điều này có nghĩa là phần lớn cơ sở dữ liệu của bạn

Máy chủ không phải chịu gánh nặng hoạt động của việc quản lý nén trực tuyến.

Nén trực tuyến chỉ là một trong nhiều phức tạp được thực hiện bởi cơ sở dữ liệu

khi nó phải hỗ trợ ghi ngẫu nhiên. Một sự phức tạp khác là sự cần thiết phải đồng bộ hóa

đọc và viết để các giá trị nửa viết không bao giờ được đọc. Khi cơ sở dữ liệu không

có ghi ngẫu nhiên, nó có thể tối ưu hóa đường dẫn đọc và có được hiệu suất tốt hơn

cơ sở dữ liệu đọc / ghi ngẫu nhiên.

Một chỉ số thô nhưng tốt về độ phức tạp có thể được nhìn thấy trong kích thước của cơ sở mã. ElephantDB, một cơ sở dữ liệu được xây dựng đặc biệt như một cơ sở dữ liệu lớp phục vụ, chỉ là một vài

ngàn dòng mã. HBase và Cassandra, hai cách đọc / ghi phân tán phổ biến

cơ sở dữ liệu, dài hàng trăm nghìn dòng. Số dòng mã không

Thông thường là một số liệu độ phức tạp tốt, nhưng trong trường hợp này, sự khác biệt đáng kinh ngạc sẽ được kể.

Một cơ sở dữ liệu đơn giản hơn có thể dự đoán được nhiều hơn vì nó thực hiện ít việc hơn. Nó cho phù hợp

ít có khả năng xảy ra lỗi hơn và như bạn đã thấy với việc nén chặt - sẽ dễ vận hành hơn đáng kể. Bởi vì các chế độ xem lớp phục vụ chứa phần lớn áp đảo

dữ liệu có thể truy vấn của bạn, sự đơn giản cơ bản của lớp phục vụ là một lợi ích rất lớn cho

sự mạnh mẽ của kiến trúc tổng thể của bạn.

1. Tương phản với một giải pháp gia tăng hoàn toàn

Trong vài chương trước, bạn đã thấy cách xây dựng lô và phân phối các lớp cho SuperWebAnalytics.com. Mọi thứ khá đơn giản bằng cách sử dụng mô hình chế độ xem máy tính này như một chức năng của tất cả dữ liệu. Để có được sự đánh giá cao hơn cho sự xuất sắc

Các thuộc tính như một hệ thống thể hiện, nó đáng để đối chiếu nó với một kiến trúc truyền thống được xây dựng bằng cách sử dụng tính toán gia tăng đầy đủ. Một kiến trúc truyền thống sử dụng một lớn

đọc / ghi cơ sở dữ liệu cho trạng thái và duy trì trạng thái đó khi dữ liệu mới xuất hiện.

Khi Lambda Architecture được giới thiệu trong chương 1, chúng ta đã đối chiếu một giải pháp truyền thống với giải pháp Lambda Architecture với vấn đề độc đáo theo thời gian. Bây giờ chúng ta đã đề cập đến các khái niệm cần thiết để xem xét sự tương phản đó một cách chi tiết. Ở đây

Chúng tôi sẽ trình bày giải pháp gia tăng đầy đủ, nổi tiếng nhất cho các tính năng độc đáo theo thời gian

vấn đề, và bạn sẽ thấy rằng giải pháp kết quả phức tạp hơn để thực hiện, là

kém chính xác hơn đáng kể, có độ trễ và đặc điểm thông lượng kém hơn, và

đòi hỏi phần cứng đặc biệt thậm chí phải khả thi.

5.1 Giải pháp gia tăng đầy đủ cho uniques theo thời gian

Chúng tôi sẽ từng bước xây dựng giải pháp gia tăng đầy đủ tốt nhất có thể. Để bắt đầu điều này

quá trình, hãy làm cho vấn đề dễ dàng hơn đáng kể bằng cách bỏ qua hoàn toàn equivs trong giải pháp ban đầu. Điều này sẽ cung cấp những hiểu biết có giá trị về khó khăn hơn nhiều

vấn đề của uniques với equivs.

Lưu ý rằng khi giải quyết vấn đề này, chúng tôi sẽ không giới hạn bản thân trong các công cụ có sẵn

trong bối cảnh hiện nay. Bất kỳ biến thể hợp lý nào của dụng cụ hiện có đều được cho phép.

Điều mình quan tâm là những gì về cơ bản có thể xảy ra, tốt hay xấu

giải pháp gia tăng đầy đủ tốt nhất có thể với công cụ tốt nhất có thể?

Bởi vì đó là một giải pháp gia tăng đầy đủ, khía cạnh quan trọng để giải quyết vấn đề là

Xác định loại cơ sở dữ liệu sẽ sử dụng và cách duy trì trạng thái trong cơ sở dữ liệu đó.

Đối với lần thử đầu tiên, hãy thử sử dụng cơ sở dữ liệu khóa-to-set.

**LỚP SPEED**

* Phần 3 tập trung vào lớp tốc độ của Lambda Architecture.Lớp tốc độ

bù đắp cho độ trễ cao của lớp lô để cho phép kết quả cập nhật

cho các truy vấn

* Chương 12 thảo luận về lượt xem thời gian thực so với lượt xem hàng loạt. Sự khác biệt chính là cơ sở dữ liệu chế độ xem thời gian thực phải hỗ trợ ghi ngẫu nhiên, điều này rất nhiều

làm tăng độ phức tạp của cơ sở dữ liệu. Bạn sẽ thấy rằng sự tồn tại của

Lớp hàng loạt làm giảm bớt sự phức tạp của việc quản lý cơ sở dữ liệu như vậy. Bạn cũng sẽ thấy

rằng một lớp tốc độ có thể được thực hiện đồng bộ hoặc không đồng bộ.

Chương 13 minh họa các chế độ xem thời gian thực bằng Apache Cassandra.

* Kiến trúc đồng bộ không yêu cầu bất kỳ lời giải thích bổ sung nào, vì vậy chương 14 bắt đầu thảo luận về kiến trúc không đồng bộ cho lớp tốc độ. Nó

thảo luận về việc sử dụng tính toán gia tăng bằng cách sử dụng hàng đợi và xử lý luồng. Có hai mô hình xử lý luồng chính, mỗi mô hình đều có mô hình riêng

Đánh đổi: từng đợt một và vi đợt. Chương 14 khám phá khái niệm

Xử lý luồng một lần và Chương 15 minh họa mô hình đó bằng cách sử dụng

Apache Kafka và Apache Storm

* Chương 16 đi sâu vào mô hình khác: xử lý luồng vi lô. Bạn sẽ thấy rằng bằng cách hy sinh một số độ trễ, bạn sẽ có được những khả năng mới mạnh mẽ. Chương 17 minh họa quá trình xử lý luồng vi lô bằng Trident.

Cho đến thời điểm này, cuộc thảo luận của chúng tôi về Kiến trúc Lambda đã xoay quanh

các lớp phân phối và phân phối — các thành phần liên quan đến các chức năng điện toán trên

mọi dữ liệu bạn có. Các lớp này đáp ứng tất cả các thuộc tính mong muốn của một

Hệ thống dữ liệu Lưu một: Cập nhật độ trễ thấp. Công việc duy nhất của lớp tốc độ là đáp ứng yêu cầu cuối cùng này.

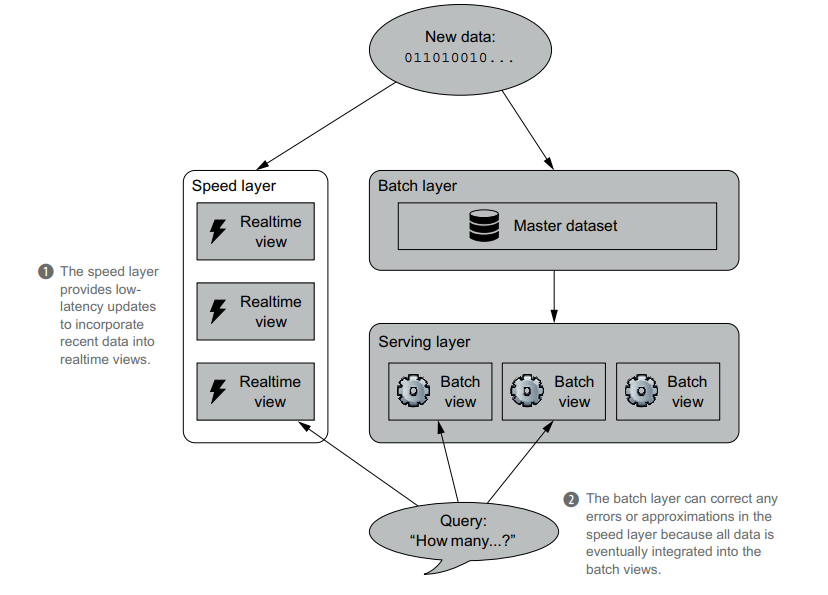
Chạy các hàm trên toàn bộ tập dữ liệu chính—có khả năng là hàng petabyte

dữ liệu—là một hoạt động sử dụng nhiều tài nguyên. Để giảm độ trễ của các bản cập nhật nhiều như

Có thể, lớp tốc độ phải có cách tiếp cận khác về cơ bản so với

hàng loạt và phục vụ các lớp. Như vậy, lớp tốc độ dựa trên tính toán gia tăng

thay vì tính toán hàng loạt.



Hình 12.1 Lớp tốc độ cho phép Kiến trúc Lambda phục vụ các truy vấn có độ trễ thấp qua dữ liệu cập nhật.

Tính toán gia tăng giới thiệu nhiều thách thức mới và nhiều hơn đáng kể

phức tạp hơn tính toán hàng loạt. May mắn thay, các yêu cầu hẹp của tốc độ

Lớp cung cấp hai lợi thế. Đầu tiên, lớp tốc độ chỉ chịu trách nhiệm cho dữ liệu chưa được bao gồm trong chế độ xem lớp phục vụ. Dữ liệu này nhiều nhất là vài giờ và rất nhiều nhỏ hơn tập dữ liệu chính. Xử lý dữ liệu ở quy mô nhỏ hơn cho phép lớn hơn

Thiết kế linh hoạt. Thứ hai, các chế độ xem lớp tốc độ là thoáng qua. Sau khi dữ liệu được hấp thụ Vào các chế độ xem lớp phục vụ, nó có thể được loại bỏ khỏi lớp tốc độ. Mặc dù

lớp tốc độ phức tạ p hơn và do đó dễ bị lỗi hơn, bất kỳ lỗi nào cũng tồn tại trong thời gian ngắn

và sẽ được tự động sửa chữa thông qua các lớp phân phối và lô đơn giản hơn.

Như chúng tôi đã nhiều lần tuyên bố, sức mạnh của Kiến trúc Lambda nằm ở sự phân tách vai trò trong các lớp khác nhau (như trong hình 12.1). Trong các kiến trúc dữ liệu truyền thống như kiến trúc dựa trên cơ sở dữ liệu quan hệ, tất cả những gì tồn tại là một lớp tốc độ. Những Các hệ thống có các tùy chọn hạn chế để chống lại sự phức tạp của tính toán gia tăng.

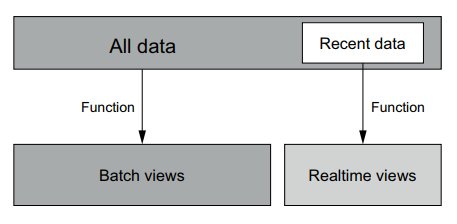
Có hai khía cạnh chính của lớp tốc độ: lưu trữ các chế độ xem thời gian thực và xử lý luồng dữ liệu đến để cập nhật các chế độ xem đó. Chương này tập trung về cấu trúc và lưu trữ các chế độ xem thời gian thực. Chúng ta sẽ bắt đầu với một cái nhìn tổng quan về Nền tảng lý thuyết của lớp tốc độ, và sau đó tiếp tục đến những thách thức khác nhau mà bạn sẽ gặp phải với tính toán gia tăng. Sau đó, chúng tôi sẽ chứng minh cách để hết hạn dữ liệu từ lớp tốc độ.

**Tính toán chế độ xem thời gian thực**

Mục tiêu cơ bản của lớp tốc độ cũng giống như đối với các lớp lô và phân phối: đến tạo ra các chế độ xem có thể được truy vấn hiệu quả. Sự khác biệt chính là các quan điểm chỉ đại diện cho dữ liệu gần đây và chúng phải được cập nhật ngay sau dữ liệu mới đến. "Rất ngắn" có nghĩa là gì khác nhau đối với mỗi ứng dụng, nhưng nó thường dao động từ một vài mili giây đến vài giây. Yêu cầu này có hậu quả sâu rộng về cách tiếp cận tính toán để tạo chế độ xem lớp tốc độ.

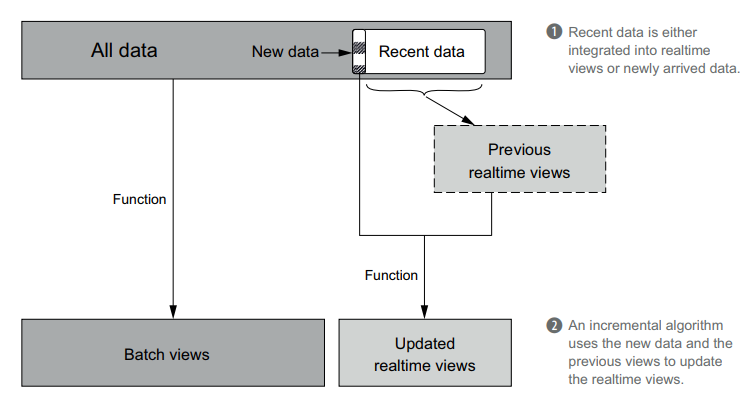
Thật không may, sơ đồ này tỏ ra không thực tế đối với nhiều ứng dụng khi bạn xem xét độ trễ và đặc điểm sử dụng tài nguyên của nó. Giả sử hệ thống dữ liệu của bạn nhận được 32 GB dữ liệu mới mỗi ngày và dữ liệu mới đó sẽ được đưa vào lớp phân phối trong vòng 6 giờ kể từ khi nhận được. Lớp tốc độ sẽ chịu trách nhiệm nhiều nhất là 6 số giờ dữ liệu—khoảng 8 GB. Mặc dù đó không phải là một số tiền lớn, nhưng 8 GB là đáng kể khi cố gắng đạt được độ trễ dưới giây. Ngoài ra, chạy một chức năng trên 8 GB dữ liệu mỗi khi bạn nhận được một phần dữ liệu mới sẽ cực kỳ tốn nhiều tài nguyên. Nếu kích thước trung bình của một đơn vị dữ liệu là 100 byte, thì 8 GB dữ liệu gần đây tương đương với khoảng 86 triệu đơn vị dữ liệu. Luôn cập nhật các chế độ xem theo thời gian thực do đó sẽ yêu cầu một số tiền bất hợp lý là 86,000,000 ✕ 8 GB trị giá xử lý cứ sau 6 giờ. Bạn có thể giảm mức sử dụng tài nguyên bằng cách gộp các cập nhật, nhưng điều này sẽ làm tăng đáng kể độ trễ cập nhật.

Nếu ứng dụng của bạn có thể chấp nhận độ trễ theo thứ tự vài phút, điều này đơn giản Chiến lược là một cách tiếp cận tốt. Nhưng nói chung, bạn sẽ cần tạo ra các chế độ xem thời gian thực trong một cách thức tiết kiệm tài nguyên với độ trễ cấp mili giây. Đối với phần còn lại của điều này chương, chúng ta sẽ giới hạn cuộc thảo luận của chúng ta trong kịch bản này.



Hình 12.2 realtime view = function(recent data)

(Một chiến lược đơn giản phản ánhLô / Lớp phục vụ và Tính toán các chế độ xem thời gian thựcsử dụng tất cả dữ liệu gần đây làm đầu vào.)



Hình 12.3 chiến lược gia tang: realtime view = function(new data, previous realtime view)

1. Dữ liệu gần đây là một trong hai Tích hợp vào thời gian thực lượt xem hoặc dữ liệu mới đến.
2. Một thuật toán gia tang sử dụng dữ liệu mới và Các dạng xem trước đó để cập nhật Các chế độ xem thời gian thực

Nói chung, bất kỳ giải pháp khả thi nào cũng dựa vào việc sử dụng các thuật toán gia tăng, như được mô tả

trong hình 12.3. Ý tưởng là cập nhật các chế độ xem thời gian thực khi dữ liệu xuất hiện, do đó sử dụng lại công việc trước đây đã tạo ra các lượt xem. 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

Views. Trong phần tiếp theo, chúng ta sẽ thảo luận thêm về các cơ sở dữ liệu này, trong khi đi sâu vào

Chi tiết lưu trữ các chế độ xem lớp tốc độ.

**Lưu trữ chế độ xem thời gian thực**

Nghĩa vụ của các chế độ xem lớp tốc độ khá khắt khe — 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 — Chế độ xem thời gian thực sẽ hỗ trợ đọc ngẫu nhiên nhanh để trả lờitruy vấn nhanh chóng. Điều này có nghĩa là dữ liệu chứa trong đó phải được lập chỉ mục.
* 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.

Các thuộc tính này là phổ biến cho một lớp cơ sở dữ liệu đã được đặt tên là NoSQL cơ sở dữ liệu. Cơ sở dữ liệu NoSQL hỗ trợ các mô hình dữ liệu và loại chỉ mục đa dạng, vì vậy bạn có thể Chọn một hoặc nhiều cơ sở dữ liệu chế độ xem theo thời gian thực để đáp ứng yêu cầu lập chỉ mục của bạn. Cho ví dụ: bạn có thể chọn Cassandra để lưu trữ các chỉ mục có định dạng khóa/giá trị và sau đó sử dụng ElasticSearch cho các chỉ mục hỗ trợ truy vấn tìm kiếm. Cuối cùng bạn đã có

Sức mạnh tuyệt vời và tính linh hoạt trong việc lựa chọn kết hợp các cơ sở dữ liệu để phù hợp với chính xác của bạn nhu cầu lớp tốc độ.

**\* Độ chính xác cuối cùng**

Lựa chọn cơ sở dữ liệu của bạn xác định cách các chế độ xem thời gian thực được lưu trữ, nhưng bạn Có sự linh hoạt tuyệt vời trong những gì bạn lưu trữ để trả lời các truy vấn. Trong nhiều trường hợp, nội dung của chế độ xem thời gian thực của bạn sẽ phản ánh chính xác nội dung của lượt xem hàng loạt của bạn. Cho Ví dụ: một truy vấn trả về số lần xem trang sẽ lưu trữ số lượng chính xác trong cả lô

lớp và lớp tốc độ. Tuy nhiên, điều này không cần phải như vậy, bởi vì nó thường xuyên

khó tính toán dần dần các hàm có thể dễ dàng tính toán theo lô.

Bạn đã gặp phải tình huống này trước đây khi xác định số lượng duy nhất. Theo lô

Điều này rất dễ tính toán vì bạn xử lý toàn bộ tập dữ liệu cùng một lúc, nhưng nó

khó hơn nhiều trong thời gian thực vì bạn cần lưu trữ toàn bộ bộ để cập nhật chính xác các số lượng.

Trong những trường hợp này, bạn có thể thực hiện một cách tiếp cận khác trong lớp tốc độ và gần đúng

câu trả lời đúng. Bởi vì tất cả dữ liệu cuối cùng được thể hiện trong lô và phân phát

Chế độ xem lớp, bất kỳ xấp xỉ nào bạn thực hiện trong lớp tốc độ đều được sửa liên tục. Điều này có nghĩa là bất kỳ xấp xỉ nào chỉ là tạm thời và các truy vấn của bạn hiển thị cuối cùng Chính xác. Đây là một kỹ thuật thực sự mạnh mẽ mang đến cho bạn những gì tốt nhất trong tất cả các thế giới: hiệu suất, độ chính xác và tính kịp thời. Cuối cùng, các phương pháp tiếp cận chính xác là phổ biến với Các truy vấn phức tạp, chẳng hạn như những truy vấn yêu cầu máy học thời gian thực. Những quan điểm như vậy có xu hướng tương quan nhiều loại dữ liệu với nhau và không thể dần dần tạo ra một chính xác trả lời theo bất kỳ cách hợp lý nào. Bạn sẽ thấy các ví dụ tận dụng độ chính xác cuối cùng khi chúng tôi loại bỏ lớp tốc độ cho SuperWebAnalytics.com

Một lần nữa, chúng ta phải nhấn mạnh rằng độ chính xác cuối cùng là một kỹ thuật tùy chọn có thể được sử dụng để giảm đáng kể việc sử dụng tài nguyên trong lớp tốc độ. Đó là một tùy chọn chỉ có sẵn cho bạn vì Lambda Architecture có hai lớp với các phương pháp tính toán riêng biệt. Trong một kiến trúc truyền thống dựa trên sự gia tăng hoàn toàn tính toán, một kỹ thuật như vậy không phải là một lựa chọn.

**\*Lượng trạng thái được lưu trữ trong lớp tốc độ**

Các lớp tốc độ lưu trữ một lượng trạng thái tương đối nhỏ vì chúng chỉ đại diện cho các chế độ xem trên dữ liệu gần đây. Đây là một lợi ích vì chế độ xem thời gian thực phức tạp hơn nhiều so với phục vụ chế độ xem lớp.

Chúng ta hãy xem xét ngắn gọn sự phức tạp của các chế độ xem thời gian thực bị bỏ qua bởi lớp phục vụ:

* Nén trực tuyến—Khi cơ sở dữ liệu đọc/ghi nhận được các bản cập nhật, các phần của chỉ mục đĩa

trở nên không sử dụng, lãng phí không gian. Định kỳ cơ sở dữ liệu phải thực hiện nén

để lấy lại không gian. Nén là một quá trình tốn nhiều tài nguyên và có khả năng

bỏ đói cỗ máy tài nguyên cần thiết để nhanh chóng phục vụ các truy vấn. Quản lý nén không đúng cách có thể gây ra lỗi xếp tầng của toàn bộ cụm.

* Tính đồng thời—Cơ sở dữ liệu đọc/ghi có khả năng nhận được nhiều lần đọc hoặc

viết cho cùng một giá trị cùng một lúc. Do đó, nó cần phải phối hợp

những điều này đọc và ghi để ngăn trả về các giá trị cũ hoặc không nhất quán. Chia sẻ Trạng thái có thể thay đổi trên các luồng là một vấn đề nổi tiếng phức tạp và kiểm soát Các chiến lược như khóa nổi tiếng là dễ bị lỗi

Điều quan trọng cần lưu ý là lớp tốc độ chịu ít áp lực hơn vì nó lưu trữ dữ liệu ít hơn đáng kể so với lớp phục vụ. Sự phân chia vai trò và trách nhiệm

trong Lambda Architecture giới hạn độ phức tạp trong lớp tốc độ. Bởi vì

lớp tốc độ thường chỉ chiếm vài giờ dữ liệu, đọc ngẫu nhiên /

cụm cơ sở dữ liệu ghi ngẫu nhiên có thể nhỏ hơn 100 lần so với nếu bạn có một cụm cơ sở dữ liệu ghi đầy đủ

kiến trúc gia tăng. Các cụm nhỏ hơn dễ quản lý hơn rất nhiều.

Bây giờ bạn đã hiểu những điều cơ bản về lớp tốc độ, hãy xem xét các thách thức Bạn sẽ phải đối mặt khi thực hiện tính toán gia tăng thay vì tính toán hàng loạt.

Ba chế độ xem, lớp hàng loạt lớp tốc độ và lớp phục vụ trả lời các câu hỏi khác nhau và ở quy mô khác. Dữ liệu được xử lý 3 lần, trong đó nó thay đổi hình dạng và âm lượng, nhưng 3 chế độ xem này trả lời các truy vấn rất khác nhau mà không thể thực hiện được từ một quan điểm duy nhất.

Lớp phân phối cập nhật khi quá trình tính toán hàng loạt đã hoàn tất. Điều đó có nghĩa là dữ liệu duy nhất không có sẵn trong hệ thống thuộc về khung thời gian khi tính toán đang chạy.

Để bù đắp cho vài phút hoặc vài giờ dữ liệu không có sẵn trong hệ thống, cần có lớp tốc độ.Sự khác biệt là lớp tốc độ chỉ xem xét một tập hợp con của dữ liệu, trong khi lớp hàng loạt xem xét toàn bộ tập dữ liệu. Ngoài ra, lớp hàng loạt có tính đến toàn bộ tập dữ liệu, trong khi lớp tốc độ xem xét dữ liệu gần đây, có nghĩa là nó hoạt động với dữ liệu gia tăng. Nó có thể được tóm tắt là

realtime view = function (realtime view, new data)

Lớp tốc độ giải quyết chủ yếu một vấn đề, cập nhật độ trễ thấp. Lớp tốc độ có dữ liệu có thể bị loại bỏ sau khi được xử lý bởi quy trình và vì tập dữ liệu thời gian thực trong một cửa sổ tương đối nhỏ (so với dữ liệu hàng loạt), nên sức mạnh xử lý cũng yêu cầu thấp. Kích thước của cửa sổ khác nhau giữa các ứng dụng. Hai điểm chính cần xem xét khi xử lý lớp tốc độ là lưu trữ các chế độ xem thời gian thực và xử lý luồng dữ liệu đến để cập nhật các chế độ xem đó. Lớp tốc độ sẽ hỗ trợ:

a) Đọc ngẫu nhiên nhanh để trả lời các truy vấn một cách nhanh chóng.

b) Ghi ngẫu nhiên nhanh chóng để cập nhật hoặc thêm dữ liệu mới

c) Khả năng mở rộng, bởi vì có một cơ hội rất thực tế kích thước dữ liệu đến sẽ tăng lên

d) Khả năng chịu lỗi, dữ liệu và logic xử lý phải được sao chép trên các máy trong trường hợp thất bại.

Định lý CAP

Một thách thức bổ sung phát sinh khi làm việc với lớp tốc độ, và đó được gọi là định lý CAP. Nó là như sau.

"Khi một hệ thống dữ liệu phân tán được phân vùng, nó có thể nhất quán hoặc có sẵn, nhưng không phải cả hai".

Hãy để chúng tôi xem nó có nghĩa là gì. Bạn có một tập dữ liệu được lưu trữ trong HDFS ví dụ. Nó sẽ được lưu trữ trên một trong các phân vùng. Nó sẽ có sẵn, nhưng không nhất quán vì không phải mọi phân vùng đều biết dữ liệu ở đâu. Chỉ có namenode mới biết nó ở đâu.

Nếu bạn lấy cơ sở dữ liệu NoSQL được phân vùng làm ví dụ, thì dữ liệu có thể được lưu trữ trên các phân vùng và cuối cùng nó có thể trở nên nhất quán và có sẵn, nhưng không phải trong thời gian đó (tại thời điểm đó, có thể được yêu cầu đưa ra câu trả lời cho các câu hỏi kinh doanh, như chúng ta sẽ thấy trong bài đăng này).

Tóm lại, lớp tốc độ liên quan đến dữ liệu theo thời gian thực, thường được thu thập trong vòng vài giây kể từ khi sự kiện xảy ra, phải được xử lý trong thời gian thực (có nghĩa là yêu cầu thời gian xử lý nhanh hơn) và phải được lưu trữ tương đối nhanh và có thể bị loại bỏ sau khi nó đã trả lời các câu hỏi mà nó có nghĩa là phải trả lời, bởi vì nó sẽ có sẵn trong lớp phục vụ hoặc hàng loạt.