**Hệ thống cơ sở dữ liệu NoSQL: Khảo sát và hướng dẫn quyết định**

Ngày nay, dữ liệu được tạo ra và tiêu thụ ở quy mô chưa từng có. Điều này đã dẫn đến các phương pháp tiếp cận mới để quản lý dữ liệu có thể mở rộng được tính theo thuật ngữ dữ liệu "NoSQL" hệ thống cơ sở để xử lý khối lượng dữ liệu ngày càng tăng và yêu cầu tải ngày càng nhiều.

Tuy nhiên, sự không đồng nhất và đa dạng của nhiều hệ thống hiện có cản trở việc lựa chọn đầy đủ thông tin về kho dữ liệu phù hợp với bối cảnh ứng dụng nhất định. Do đó, bài viết này cung cấp một cái nhìn tổng quan từ trên xuống về lĩnh vực. Thay vì đối chiếu các chi tiết cụ thể thực hiện của các đại diện riêng lẻ, chúng tôi đề xuất một mô hình phân loại so sánh liên quan đến chức năng và phi chức năng yêu cầu đối với các kỹ thuật và thuật toán được sử dụng trong cơ sở dữ liệu NoSQL. Hộp công cụ NoSQL này cho phép chúng tôi rút ra một cây quyết định đơn giản để giúp các học viên và nhà nghiên cứu lọc các ứng cử viên hệ thống tiềm năng dựa trên các yêu cầu ứng dụng trung tâm.

1. **Giới thiệu**

Hệ thống quản lý cơ sở dữ liệu quan hệ truyền thống (RDBMS) cung cấp các cơ chế mạnh mẽ để lưu trữ và truy vấn dữ liệu có cấu trúc dưới sự nhất quán mạnh mẽ, đảm bảo giao dịch và đã đạt đến mức độ tin cậy, ổn định và hỗ trợ chưa từng có trong nhiều thập kỷ phát triển. Tuy nhiên, trong những năm gần đây, lượng dữ liệu hữu ích trong một số lĩnh vực ứng dụng đã trở nên quá lớn đến mức không thể được lưu trữ hoặc xử lý bởi các giải pháp cơ sở dữ liệu truyền thống. Nội dung do người dùng tạo trong các mạng xã hội hoặc dữ liệu được truy xuất từ các mạng cảm biến lớn chỉ là hai ví dụ về hiện tượng này thường được gọi là Dữ Liệu Lớn (Big Data). Một lớp các hệ thống lưu trữ dữ liệu mới có thể đối phó với Big Data được gộp lại dưới thuật ngữ cơ sở dữ liệu NoSQL, nhiều trong số đó cung cấp khả năng mở rộng theo chiều ngang và tính khả dụng cao hơn so với cơ sở dữ liệu quan hệ bằng cách hy sinh khả năng truy vấn và đảm bảo tính nhất quán. Sự đánh đổi này rất quan trọng đối với điện toán theo định hướng dịch vụ và như một mô hình dịch vụ, vì bất kỳ dịch vụ trạng thái nào cũng chỉ có thể mở rộng và chịu lỗi như kho dữ liệu cơ bản của nó.

Có hàng tá hệ thống cơ sở dữ liệu NoSQL và thật khó để theo dõi nơi chúng vượt trội, nơi chúng thất bại hoặc thậm chí nơi chúng khác nhau, vì chi tiết triển khai thay đổi nhanh chóng và các bộ tính năng phát triển theo thời gian. Do đó trong bài viết này, chúng tôi mong muốn cung cấp một cái nhìn tổng quan về bối cảnh NoSQL bằng cách thảo luận về các khái niệm được sử dụng thay vì các đặc thù của hệ thống và khám phá các yêu cầu thường đặt ra cho các hệ thống cơ sở dữ liệu NoSQL, các kỹ thuật được sử dụng để đáp ứng các yêu cầu này và sự đánh đổi phải được thực hiện trong quá trình này. Trọng tâm của chúng tôi nằm ở các kho lưu trữ giá trị chính, tài liệu và cột rộng, vì các danh mục NoSQL này bao gồm các kỹ thuật và quyết định thiết kế phù hợp nhất trong không gian quản lý dữ liệu có thể mở rộng

Chúng tôi mô tả mức cao phổ biến nhất các phương pháp tiếp cận phân loại các hệ thống cơ sở dữ liệu NoSQL theo mô hình dữ liệu của chúng thành các kho lưu trữ khóa-giá trị, kho lưu trữ tài liệu và cột rộng hoặc theo sự đánh đổi sự sống động an toàn trong thiết kế của chúng (CAP và PACELC). Sau đó, chúng tôi khảo sát các kỹ thuật thường được sử dụng chi tiết hơn và thảo luận về mô hình về các yêu cầu và kỹ thuật có liên quan như thế nào, trước khi chúng tôi đưa ra một cái nhìn tổng quan rộng rãi về các hệ thống cơ sở dữ liệu nổi bật bằng cách áp dụng mô hình của chúng tôi cho nó. Một mô hình quyết định đơn giản và trừu tượng để hạn chế việc lựa chọn các hệ thống NoSQL phù hợp dựa trên các yêu cầu ứng dụng kết luận bài báo.

**2. Phân loại hệ thống cấp cao**

Để trừu tượng hóa các chi tiết triển khai của các hệ thống NoSQL riêng lẻ, các tiêu chí phân loại cấp cao có thể được sử dụng để nhóm các kho dữ liệu tương tự thành các danh mục. Trong phần này, chúng tôi giới thiệu hai cách tiếp cận nổi bật nhất: mô hình dữ liệu và các lớp định lý CAP.

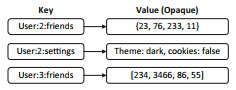
**2.1 Các mô hình dữ liệu khác nhau**

Sự khác biệt được sử dụng phổ biến nhất giữa các cơ sở dữ liệu NoSQL là cách chúng lưu trữ và cho phép truy cập vào dữ liệu. Mỗi hệ thống được đề cập trong bài báo này có thể được phân loại là kho lưu trữ giá trị chính, kho lưu trữ tài liệu hoặc kho lưu trữ cột rộng.

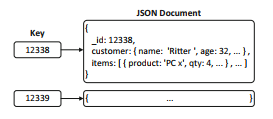
**2.1.1 Kho lưu trữ giá trị chính**

Kho giá trị khóa bao gồm một tập hợp các cặp giá trị khóa với các khóa duy nhất. Do cấu trúc đơn giản này, nó chỉ hỗ trợ các hoạt động get và put. Vì bản chất của giá trị được lưu trữ là minh bạch đối với cơ sở dữ liệu, các kho giá trị khóa thuần túy không hỗ trợ các hoạt động ngoài CRUD đơn giản (Tạo, Đọc, Cập nhật, Xóa). Do đó, các kho giá trị chính thường được gọi là không có sơ đồ [44]: mọi giả định về cấu trúc của dữ liệu được lưu trữ đều được mã hóa ngầm trong logic ứng dụng (lược đồ khi đọc [31]) và không được xác định rõ ràng thông qua ngôn ngữ định nghĩa dữ liệu (lược đồ khi ghi).

Ưu điểm rõ ràng của mô hình dữ liệu này nằm ở sự đơn giản của nó. Tính trừu tượng rất đơn giản giúp bạn dễ dàng phân vùng và truy vấn dữ liệu, để hệ thống cơ sở dữ liệu có thể đạt được độ trễ thấp cũng như thông lượng cao. Tuy nhiên, nếu một ứng dụng yêu cầu các hoạt động phức tạp hơn, ví dụ: truy vấn phạm vi, mô hình dữ liệu này không đủ mạnh. Hình 1 minh họa cách dữ liệu và cài đặt tài khoản người dùng có thể được lưu trữ trong kho giá trị khóa. Vì các truy vấn phức tạp hơn các tra cứu đơn giản không được hỗ trợ, dữ liệu phải được phân tích không hiệu quả trong mã ứng dụng để trích xuất thông tin như cookie có được hỗ trợ hay không (cookie: false).



**Hình 1** Kho giá trị chính cung cấp khả năng lưu trữ và truy xuất hiệu quả các giá trị tùy ý



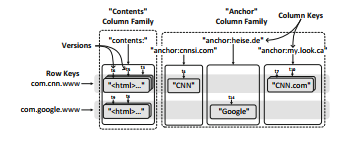
**Hình 2**: Kho tài liệu nhận thức được cấu trúc bên trong của thực thể được lưu trữ và do đó có thể hỗ trợ các truy vấn

**2.1.2 Kho tài liệu**

Kho lưu trữ tài liệu là một kho giá trị chính giới hạn các giá trị ở các định dạng bán cấu trúc như tài liệu JSON1. Hạn chế này so với các kho giá trị chính mang lại sự linh hoạt tuyệt vời trong việc truy cập dữ liệu. Không chỉ có thể tìm nạp toàn bộ tài liệu bằng ID của nó mà còn chỉ có thể truy xuất các phần của tài liệu, ví dụ: tuổi của khách hàng và thực hiện các truy vấn như tổng hợp, truy vấn theo ví dụ hoặc thậm chí tìm kiếm toàn văn bản (Hình 2).

**2.1.3.Kho lưu trữ cột rộng(Wide-column stores)**

Các cửa hàng cột rộng kế thừa tên của chúng từ hình ảnh thường được sử dụng để giải thích mô hình dữ liệu cơ bản: một bảng quan hệ với nhiều cột thưa thớt. Tuy nhiên, về mặt kỹ thuật, một cửa hàng cột rộng gần với bản đồ được sắp xếp đa cấp 2 phân tán: các khóa cấp đầu tiên xác định các hàng mà bản thân chúng bao gồm các cặp khóa-giá trị. Các phím cấp một được gọi là khóa hàng, các phím cấp hai được gọi là phím cột. Sơ đồ lưu trữ này làm cho các bảng có nhiều cột tùy ý khả thi, vì không có khóa cột nào mà không có giá trị tương ứng. Do đó, các giá trị null có thể được lưu trữ mà không có bất kỳ chi phí không gian nào. Tập hợp tất cả các cột được phân vùng thành các họ cột được gọi là để định vị các cột trên đĩa thường được truy cập cùng nhau. Trên đĩa, các kho lưu trữ cột rộng không đặt tất cả dữ liệu từ mỗi hàng, mà thay vào đó là các giá trị của cùng một họ cột và từ cùng một hàng. Do đó, một thực thể (một hàng) không thể được truy xuất bằng một lần tra cứu duy nhất như trong kho tài liệu, nhưng phải được nối với nhau từ các cột của tất cả các họ cột. Tuy nhiên, bố cục lưu trữ này thường cho phép nén dữ liệu hiệu quả cao và chỉ truy xuất một phần của thực thể rất hiệu quả. Dữ liệu được lưu trữ theo thứ tự từ điển học của các khóa của chúng, để dữ liệu được truy cập cùng nhau được đặt cùng vị trí vật lý, được thiết kế khóa cẩn thận. Vì tất cả các hàng được phân phối thành các phạm vi liền kề (được gọi là máy tính bảng) giữa các máy chủ máy tính bảng khác nhau, quét hàng chỉ liên quan đến một số máy chủ và do đó rất hiệu quả



**Hình 3** Dữ liệu trong một wide-column store

Bigtable [9], đi tiên phong trong mô hình cột rộng, được phát triển đặc biệt để lưu trữ một bộ sưu tập lớn các trang web như minh họa trong Hình 3. Mỗi hàng trong bảng trang web tương ứng với một trang web duy nhất. Khóa hàng là sự kết hợp của các thành phần URL theo thứ tự đảo ngược và mọi khóa cột bao gồm tên họ cột và vòng loại cột, được phân tách bằng dấu hai chấm. Có hai họ cột: họ cột "nội dung" chỉ có một cột giữ trang web thực tế và họ cột "neo" giữ các liên kết đến mỗi trang web, mỗi họ trong một cột riêng biệt. Mọi ô trong bảng (tức là mọi giá trị có thể truy cập bằng cách kết hợp phím hàng và cột) có thể được phiên bản theo dấu thời gian hoặc số phiên bản. Điều quan trọng cần lưu ý là phần lớn thông tin của một thực thể nằm trong các khóa và không chỉ trong các giá trị [9].

**2.2 Đánh đổi tính khả dụng nhất quán: CAP và PACELC**

Một thuộc tính xác định khác của cơ sở dữ liệu ngoài cách dữ liệu được lưu trữ và cách chúng có thể được truy cập là mức độ nhất quán được cung cấp. Một số cơ sở dữ liệu được xây dựng để đảm bảo tính nhất quán và khả năng tuần tự hóa mạnh mẽ (ACID), trong khi những cơ sở dữ liệu khác ủng hộ tính khả dụng (BASE). Sự đánh đổi này vốn có đối với mọi hệ thống cơ sở dữ liệu phân tán và số lượng lớn các hệ thống NoSQL khác nhau cho thấy rằng có một phổ rộng giữa hai mô hình. Trong phần sau đây, chúng tôi giải thích hai định lý CAP và PACELC theo đó các hệ thống cơ sở dữ liệu có thể được phân loại theo vị trí tương ứng của chúng trong phổ này

CAP Giống như định lý FLP nổi tiếng [19], Định lý CAP, được trình bày bởi Eric Brewer tại PODC 2000 [7] và sau đó được chứng minh bởi Gilbert và Lynch [21], là một trong những kết quả bất khả thi thực sự có ảnh hưởng trong lĩnh vực điện toán phân tán, bởi vì nó đặt một giới hạn trên cuối cùng vào những gì có thể được thực hiện bởi một hệ thống phân tán. Nó nói rằng một thanh ghi đọc / ghi nhất quán tuần tự mà cuối cùng phản hồi mọi yêu cầu không thể được thực hiện trong một hệ thống không đồng bộ dễ bị phân vùng mạng. Nói cách khác, nó có thể đảm bảo nhiều nhất hai trong số ba thuộc tính sau cùng một lúc:

* Tính nhất quán (C) Việc đọc và ghi luôn được thực thi nguyên tử và nhất quán nghiêm ngặt (có thể tuyến tính [26]). Nói cách khác, tất cả các khách hàng luôn có cùng quan điểm về dữ liệu.
* Tính khả dụng (A) Mọi nút không thất bại trong hệ thống luôn có thể chấp nhận các yêu cầu đọc và ghi của khách hàng và cuối cùng sẽ trở lại với một phản hồi có ý nghĩa, tức là không phải với thông báo lỗi.
* Dung sai phân vùng (P) Hệ thống duy trì các đảm bảo tính nhất quán được hiển thị trước đó và tính khả dụng khi có sự mất thông báo giữa các nút hoặc lỗi hệ thống một phần.

Brewer lập luận rằng một hệ thống có thể có sẵn và nhất quán trong hoạt động bình thường, nhưng với sự có mặt của phân vùng hệ thống, điều này là không thể: Nếu hệ thống tiếp tục hoạt động bất chấp phân vùng, có một số nút không thất bại đã mất liên lạc với các nút khác và do đó phải quyết định tiếp tục xử lý các yêu cầu của máy khách để duy trì tính khả dụng (AP, hệ thống nhất quán cuối cùng) hoặc từ chối các yêu cầu của khách hàng để duy trì đảm bảo tính nhất quán (CP). Tùy chọn đầu tiên vi phạm tính nhất quán, bởi vì nó có thể dẫn đến việc đọc cũ và ghi mâu thuẫn, trong khi tùy chọn thứ hai rõ ràng hy sinh tính khả dụng. Ngoài ra còn có các hệ thống thường có sẵn và nhất quán, nhưng thất bại hoàn toàn khi có phân vùng (CA), ví dụ như hệ thống nút đơn. Người ta đã chứng minh rằng định lý CAP giữ cho bất kỳ thuộc tính nhất quán nào ít nhất là mạnh bằng tính nhất quán nhân quả, cũng bao gồm bất kỳ giới hạn gần đây nào về độ cũ cho phép của dữ liệu (-tính nguyên tử) [37]. Khả năng tuần tự hóa là tiêu chí chính xác của sự cô lập giao dịch không đòi hỏi tính nhất quán mạnh mẽ. Tuy nhiên, tương tự như tính nhất quán, khả năng nối tiếp cũng không thể đạt được trong các phân vùng mạng [15].

Việc phân loại các hệ thống NoSQL là AP, CP hoặc CA phản ánh một cách mơ hồ khả năng của các hệ thống riêng lẻ và do đó được chấp nhận rộng rãi như một phương tiện để so sánh cấp cao. Tuy nhiên, điều quan trọng cần lưu ý là Định lý CAP thực sự không nêu bất cứ điều gì về hoạt động bình thường; Nó chỉ đơn thuần cho chúng ta biết liệu một hệ thống ủng hộ tính khả dụng hay tính nhất quán khi đối mặt với phân vùng Anetwork. Trái ngược với Định lý FLP, định lý CAP giả định một mô hình thất bại cho phép bỏ các thông điệp tùy ý, sắp xếp lại hoặc trì hoãn vô thời hạn. Theo giả định yếu hơn về các kênh liên lạc đáng tin cậy (tức là tin nhắn luôn đến nhưng không đồng bộ và có thể được sắp xếp lại) một hệ thống CAP trên thực tế có thể sử dụng thuật toán Attiya, Bar-Noy, Dolev [2], miễn là phần lớn các nút hoạt động.

PACELC Sự thiếu định lý CAP này được đề cập trong một bài báo của Daniel Abadi [1] trong đó ông chỉ ra rằng Định lý CAP không nắm bắt được sự đánh đổi giữa độ trễ và tính nhất quán trong quá trình hoạt động bình thường, mặc dù nó đã được chứng minh là có ảnh hưởng nhiều hơn đến thiết kế của các hệ thống phân tán so với sự đánh đổi tính nhất quán sẵn có trong các tình huống thất bại. Ông xây dựng PACELC thống nhất cả hai sự đánh đổi và do đó miêu tả không gian thiết kế của các hệ thống phân tán chính xác hơn. Từ PACELC, chúng tôi biết rằng trong trường hợp có phân vùng, có sự đánh đổi Tính nhất quán về tính khả dụng; Khác, tức là trong hoạt động bình thường, có sự đánh đổi Tính nhất quán về độ trễ

Sự phân loại này về cơ bản cung cấp hai lựa chọn khả thi cho kịch bản phân vùng (A/C) và cũng có hai lựa chọn cho hoạt động bình thường (L/C) và do đó có vẻ chi tiết hơn so với phân loại CAP. Tuy nhiên, nhiều hệ thống không thể được gán riêng cho một lớp PACELC duy nhất và một trong bốn lớp PACELC, cụ thể là PC/EL, khó có thể được gán cho bất kỳ hệ thống nào.

**3. Techniques(Kỹ thuật)**

Mọi cơ sở dữ liệu thành công đáng kể đều được thiết kế cho một lớp ứng dụng cụ thể hoặc để đạt được sự kết hợp cụ thể của các thuộc tính hệ thống mong muốn. Lý do đơn giản giải thích tại sao có rất nhiều hệ thống cơ sở dữ liệu khác nhau là không thể có bất kỳ hệ thống nào đạt được tất cả các thuộc tính mong muốn cùng một lúc. Các cơ sở dữ liệu SQL truyền thống như PostgreSQL đã được xây dựng để cung cấp gói chức năng đầy đủ: mô hình dữ liệu rất linh hoạt, khả năng truy vấn tinh vi bao gồm các liên kết, ràng buộc toàn vẹn toàn cầu và đảm bảo giao dịch. Ở phía bên kia của quang phổ thiết kế, có các kho lưu trữ khóa-giá trị như Dynamo mở rộng theo dữ liệu và khối lượng yêu cầu và cung cấp thông lượng đọc và ghi cao cũng như độ trễ thấp, nhưng hầu như không có bất kỳ chức năng nào ngoài tra cứu đơn giản. Trong phần này, chúng tôi nêu bật không gian thiết kế của các hệ thống cơ sở dữ liệu phân tán, tập trung vào phân mảnh, sao chép, quản lý lưu trữ và xử lý truy vấn. Chúng tôi khảo sát các kỹ thuật sẵn có và thảo luận xem chúng liên quan như thế nào đến các thuộc tính (mục tiêu) chức năng và phi chức năng khác nhau củacác hệ thống quản lý dữ liệu. Để minh họa kỹ thuật nào phù hợp để đạt được thuộc tính hệ thống nào, chúng tôi cung cấp Hộp công cụ NoSQL (Hình 4) trong đó mỗi kỹ thuật được kết nối với các thuộc tính chức năng và phi chức năng mà nó cho phép (chỉ các cạnh tích cực).

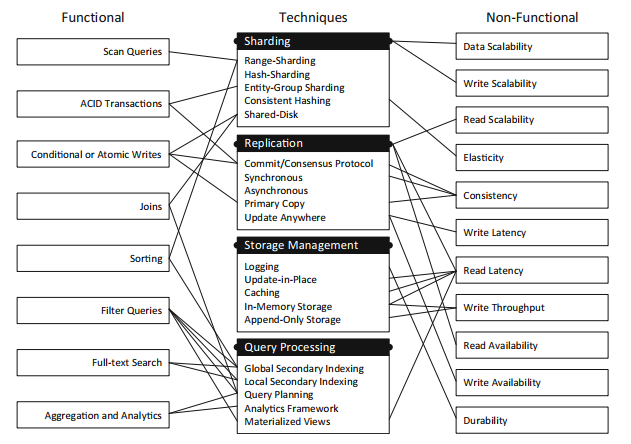
**3.1 Sharding (Phân mảnh)**

Một số hệ thống cơ sở dữ liệu quan hệ phân tán như Oracle RAC hoặc IBM DB2 pureScale dựa trên kiến trúc đĩa dùng chung, nơi tất cả các nút cơ sở dữ liệu truy cập vào cùng một kho lưu trữ dữ liệu trung tâm (ví dụ: NAS hoặc SAN). Do đó, các hệ thống này cung cấp dữ liệu nhất quán mọi lúc, nhưng cũng vốn dĩ khó mở rộng quy mô. Ngược lại, các hệ thống cơ sở dữ liệu (NoSQL) tập trung vào bài báo này được xây dựng dựa trên kiến trúc không có gì dùng chung, có nghĩa là mỗi hệ thống bao gồm nhiều máy chủ có bộ nhớ riêng và đĩa riêng được kết nối thông qua mạng. Do đó, khả năng mở rộng cao về thông lượng và khối lượng dữ liệu đạt được bằng cách phân mảnh (phân vùng) dữ liệu trên các nút (phân đoạn) khác nhau trong hệ thống.

Có ba kỹ thuật phân phối cơ bản: phân mảnh phạm vi, phân mảnh băm và phân mảnh nhóm thực thể. Để có thể quét hiệu quả, dữ liệu có thể được phân vùng thành các phạm vi giá trị được sắp xếp theo thứ tự và liền kề bằng cách chia nhỏ phạm vi. Tuy nhiên, cách tiếp cận này đòi hỏi một số sự phối hợp thông qua một bậc thầy quản lý các nhiệm vụ. Để đảm bảo tính linh hoạt, hệ thống phải có khả năng phát hiện và giải quyết các điểm nóng một cách tự động bằng cách chia nhỏ thêm một phân đoạn quá tải.

Bảo vệ phạm vi được hỗ trợ bởi các kho lưu trữ nhiều cột như BigTable, HBase hoặc Hypertable [49] và các kho lưu trữ tài liệu, ví dụ: MongoDB, RethinkDB, Espresso [43] và DocumentDB [46]. Một cách khác để phân vùng dữ liệu trên một số máy là phân đoạn băm trong đó mọi mục dữ liệu được gán cho máy chủ phân đoạn theo một số giá trị băm được tạo từ khóa chính. Cách tiếp cận này không yêu cầu người điều phối và cũng đảm bảo dữ liệu được phân phối đồng đều trên các phân đoạn, miễn là hàm băm được sử dụng tạo ra sự phân phối đồng đều. Tuy nhiên, nhược điểm rõ ràng là nó chỉ cho phép tra cứu và khiến việc quét không khả thi. Phân đoạn hàm băm được sử dụng trong các cửa hàng khóa-giá trị và cũng có sẵn trong một số cột rộngcác cửa hàng như Cassandra [34] hoặc Azure Tables [8].

Ví dụ: máy chủ phân đoạn chịu trách nhiệm về bản ghi có thể được xác định asserverid = hash(id)mod servers. Tuy nhiên, sơ đồ băm này yêu cầu tất cả các bản ghi phải được chỉ định lại mỗi khi một máy chủ mới tham gia hoặc rời đi, bởi vì nó thay đổi theo số lượng máy chủ phân đoạn (máy chủ). Do đó, nó không khả thi để sử dụng trong các hệ thống đàn hồi như Dynamo, Riak hoặc Cassandra, cho phép thêm các tài nguyên bổ sung theo yêu cầu và lại bị loại bỏ khi cần thiết. Để tăng tính linh hoạt, các hệ thống đàn hồi thường sử dụng hàm băm nhất quán [30] trong đó các bản ghi không được gán trực tiếp cho máy chủ mà thay vào đó là các phân vùng logic, sau đó được phân phối trên tất cả các máy chủ phân đoạn. Như vậy, chỉ một phần nhỏ củadữ liệu phải được gán lại khi có những thay đổi trong cấu trúc liên kết hệ thống. Ví dụ: một hệ thống đàn hồi có thể được thu nhỏ bằng cách giảm tải tất cả các phân vùng logic nằm trên một máy chủ cụ thể sang các máy chủ khác và sau đó tắt máy hiện đang ở chế độ chờ. Để biết chi tiết về cách sử dụng hàm băm nhất quán trong các hệ thống NoSQL, xem [18].



**Hình 4.** Hộp công cụ NoSQL: Nó kết nối các kỹ thuật của cơ sở dữ liệu NoSQL với các thuộc tính hệ thống chức năng và phi chức năng mong muốn mà chúng hỗ trợ

Phân mảnh nhóm thực thể là một sơ đồ phân vùng dữ liệu với mục tiêu cho phép các giao dịch phân vùng đơn lẻ trên dữ liệu đồng vị trí. Các phân vùng được gọi là nhóm thực thể và được ứng dụng khai báo rõ ràng (ví dụ: trong G Store [14] và MegaStore [4]) hoặc bắt nguồn từ các mẫu hình truy cập của giao dịch (ví dụ: trong Đám mây quan hệ [13] và Cloud SQL Server [5]). Nếu một giao dịch truy cập dữ liệu trải dài trên nhiều nhóm, quyền sở hữu dữ liệu có thể được chuyển giữa các nhóm thực thể hoặc người quản lý giao dịch phải dự phòng các giao thức giao dịch đa nút đắt tiền hơn

**3.2 Replication (Nhân rộng)**

Về mặt CAP, các RDBMS thông thường thường là các hệ thống CA chạy ở chế độ một máy chủ: Toàn bộ hệ thống sẽ không khả dụng khi máy bị lỗi. Và do đó, các nhà khai thác hệ thống đảm bảo tính toàn vẹn và tính khả dụng của dữ liệu thông qua phần cứng cao cấp đắt tiền nhưng đáng tin cậy. Ngược lại, các hệ thống NoSQL như Dynamo, BigTable hoặc Cassandra được thiết kế cho khối lượng dữ liệu và yêu cầu mà một máy đơn lẻ không thể xử lý được và do đó chúng chạy trên các cụm bao gồm hàng nghìn máy chủ. Vì lỗi là không thể tránh khỏi và sẽ xảy ra thường xuyên trong bất kỳ hệ thống phân tán quy mô lớn nào, nên phần mềm phải đối phó với chúng hàng ngày [24].

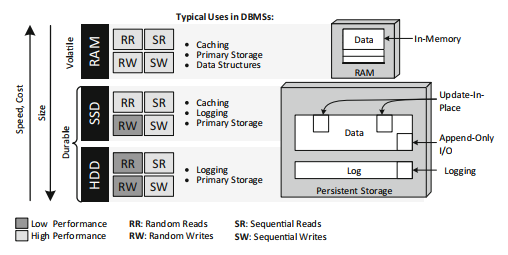
Vào năm 2009, Jeff Dean, đồng nghiệp của Google, đã tuyên bố [16] rằng một cụm mới điển hình tại Google gặp phải hàng nghìn lỗi ổ cứng, 1.000 lỗi máy đơn, 20 lỗi giá đỡ và một số phân vùng mạng do các trường hợp dự kiến và không mong muốn chỉ trong năm đầu tiên. Nhiều trường hợp phân vùng mạng và ngừng hoạt động gần đây trong các trung tâm dữ liệu đám mây lớn đã được báo cáo [3]. Sao chép cho phép hệ thống duy trì tính khả dụng và độ bền khi đối mặt với các lỗi như vậy. Nhưng việc lưu trữ các bản ghi giống nhau trên các máy khác nhau (máy chủ bản sao) trong cụm sẽ gây ra vấn đề về đồng bộ hóa giữa chúng và do đó, một bên là sự đánh đổi giữa tính nhất quán với độ trễ và tính khả dụng trên cái khác.

Grey và cộng sự. [22] đề xuất phân loại hai lớp các chiến lược sao chép khác nhau tùy theo thời điểm các bản cập nhật được truyền tới các bản sao và nơi các bản cập nhật được chấp nhận. Có hai lựa chọn khả thi ở tầng một (“khi”): Bản sao háo hức (đồng bộ) truyền các thay đổi đến một cách đồng bộ tới tất cả các bản sao trước khi một cam kết có thể được trả lại cho máy khách, trong khi bản sao lười biếng (không đồng bộ) chỉ áp dụng các thay đổi ở bản nhận. replica và chuyển chúng không đồng bộ. Ưu điểm lớn của sao chép háo hức là tính nhất quán giữa các bản sao, nhưng nó phải trả giá bằng độ trễ ghi cao hơn do phải chờ các bản sao khác và tính khả dụng bị suy giảm [22]. Sao chép lười biếng nhanh hơn, bởi vì nó cho phép các bản sao phân kỳ; do đó, dữ liệu cũ có thể được cung cấp. Ở tầng thứ hai ("ở đâu"), một lần nữa, có thể có hai cách tiếp cận khác nhau: Sơ đồ chủ-tớ (bản sao chính) được theo đuổi khi các thay đổi chỉ có thể được chấp nhận bởi một bản sao (bản chính) hoặc, trong một bản cập nhật ở bất cứ đâu (multi-master), mọi bản sao đều có thể chấp nhận ghi. Trong các giao thức chủ-tớ, kiểm soát đồng thời không phức tạp hơn trong một hệ thống phân tán không có bản sao, nhưng toàn bộ bộ bản sao sẽ không khả dụng ngay khi chủ bị lỗi. Các giao thức đa chủ yêu cầu các cơ chế phức tạp để ngăn ngừa hoặc phát hiện và điều chỉnh các thay đổi xung đột. Các kỹ thuật thường được sử dụng cho các mục đích là lập phiên bản, đồng hồ véc-tơ, sửa lỗi sai và đọc (ví dụ: trong Dynamo [18]) và các kiểu dữ liệu hội tụ hoặc giao hoán [45] (ví dụ: trong Riak).

Về cơ bản, tất cả bốn kết hợp của phân loại hai tầng đều có thể thực hiện được. Các hệ thống quan hệ phân tán thường thực hiện sao chép chủ-tớ háo hức để duy trì tính nhất quán mạnh mẽ. Háo hức cập nhật bất cứ nơi nào sao chép như ví dụ đặc trưng trong Google's Megastore phải chịu chi phí liên lạc lớn do đồng bộ hóa tạo ra và có thể gây ra bế tắc phân tán rất tốn kém để phát hiện. Các hệ thống cơ sở dữ liệu NoSQL thường dựa vào sao chép lười biếng, kết hợp với master-slave (các hệ thống CP, ví dụ HBase và MongoDB) hoặc cập nhật ở mọi nơi (các hệ thống AP, ví dụ Dynamo và Cassandra). Nhiều hệ thống NoSQL để lại sự lựa chọn giữa độ trễ và con-tính ổn định đối với máy khách, nghĩa là đối với mọi yêu cầu, máy khách quyết định nên chờ phản hồi từ bất kỳ bản sao nào để đạt được độ trễ tối thiểu hay để có phản hồi nhất quán chắc chắn (bởi phần lớn các bản sao hoặc bản chính) để ngăn dữ liệu cũ. Một khía cạnh của bản sao không được đề cập trong sơ đồ hai lớp là khoảng cách giữa các bản sao. Lợi thế rõ ràng của việc đặt các bản sao gần nhau là độ trễ thấp, nhưng việc đặt các bản sao ở gần nhau cũng có thể làm giảm tác động tích cực đến tính khả dụng; ví dụ: nếu hai bản sao của cùng một mục dữ liệu được đặt trong cùng một giá, thì mục dữ liệu đó không khả dụng khi giá bị lỗi mặc dù đã sao chép. Nhưng hơn thếNgoài khả năng không có sẵn tạm thời, việc đặt các bản sao gần đó còn có nguy cơ mất tất cả các bản sao cùng một lúc trong một kịch bản thảm họa. Một kỹ thuật thay thế để giảm độ trễ được sử dụng trong Orestes [20], trong đó dữ liệu được lưu vào bộ đệm ẩn gần với các ứng dụng sử dụng cơ sở hạ tầng bộ nhớ đệm web và các giao thức kết hợp bộ đệm.

Sao chép địa lý có thể bảo vệ hệ thống khỏi mất dữ liệu hoàn toàn và cải thiện độ trễ đọc đối với quyền truy cập phân tán từ máy khách. Háo hức sao chép địa lý, như được triển khai trong Google's Megastore [4], Spanner [12], MDCC [32] và Mạnh Tử [38] đạt được tính nhất quán cao với chi phí là độ trễ ghi cao hơn (thường là 100 mili giây [12] đến 600 mili giây [4]). Với sao chép địa lý lười biếng như trong Dynamo [18], PNUTS [11], Walter [47], COPS [36], Cassandra [34] và BigTable [9] những thay đổi gần đây có thể bị mất, nhưng hệ thống hoạt động tốt hơn và vẫn khả dụng - có thể trong quá trình phân vùng. Charron-Bost và cộng sự. [10, Chương 12] và Öszu và Valduriez [41, Chương 13] cung cấp một cuộc thảo luận toàn diện về sao chép cơ sở dữ liệu.

Để có hiệu suất tốt nhất, các hệ thống cơ sở dữ liệu cần được tối ưu hóa cho phương tiện lưu trữ mà chúng sử dụng để phục vụ và duy trì dữ liệu. Đây thường là bộ nhớ chính (RAM), ổ đĩa thể rắn (SSD) và ổ đĩa quay (HDD) có thể được sử dụng trong mọi kết hợp. Không giống như RDBMS trong các thiết lập doanh nghiệp, cơ sở dữ liệu NoSQL phân tán tránh các kiến trúc đĩa chia sẻ chuyên dụng để ưu tiên các cụm không chia sẻ dựa trên máy chủ hàng hóa (sử dụng phương tiện lưu trữ hàng hóa). Các thiết bị lưu trữ thường được hình dung như một “ kim tự tháp lưu trữ” (xem Hình 5) [25]. Ngoài ra còn có một tập hợp các bộ đệm trong suốt (ví dụ: bộ đệm CPU L1-L3 và bộ đệm đĩa, không được hiển thị trong Hình), chỉ được tận dụng hoàn toàn thông quacác thuật toán cơ sở dữ liệu được thiết kế để thúc đẩy dữ liệu cục bộ. Các đặc điểm hiệu suất và chi phí rất khác nhau của bộ lưu trữ RAM, SSD và HDD cũng như các chiến lược khác nhau để tận dụng điểm mạnh của chúng (quản lý lưu trữ) là một lý do cho sự đa dạng của cơ sở dữ liệu NoSQL. Quản lý lưu trữ có khía cạnh không gian (nơi lưu trữ dữ liệu) và khía cạnh thời gian (thời điểm lưu trữ dữ liệu). Update-in-place và append-only-IO là hai kỹ thuật tổ chức dữ liệu không gian bổ sung; trong bộ nhớ quy định RAM là vị trí của dữ liệu, trong khi ghi nhật ký là một kỹ thuật tạm thời tách rời bộ nhớ chính và bộ lưu trữ liên tục và do đó cung cấp quyền kiểm soát khi dữ liệu thực sự được lưu giữ.Trong bài báo chuyên đề “sự kết thúc của một kỷ nguyên kiến trúc” [48], Stonebraker et al. đã phát hiện ra rằng trong các RDBMS điển hình, chỉ có 6,8% thời gian thực thi được dành cho “công việc hữu ích”, trong khi phần còn lại được dành cho:



**Hình 5** Tháp lưu trữ và vai trò của nó trong các hệ thống NoSQL

* quản lý bộ đệm (34,6%), tức là lưu vào bộ nhớ đệm để giảm tốc độ truy cập đĩa chậm hơn
* chốt (14,2%), để bảo vệ cấu trúc dữ liệu được chia sẻ khỏi các điều kiện chạy đua do đa luồng gây ra
* khóa (16,3%), để đảm bảo sự cô lập logic của các giao dịch
* ghi nhật ký (11,9%), để đảm bảo độ bền khi đối mặt với lỗi
* tối ưu hóa mã hóa thủ công (16,2 %)

Điều này thúc đẩy rằng có thể mong đợi những cải thiện lớn về hiệu suất nếu RAM được sử dụng làm bộ lưu trữ chính (cơ sở dữ liệu trong bộ nhớ [50]). Nhược điểm là chi phí lưu trữ cao và thiếu độ bền—một sự cố mất điện nhỏ có thể phá hủy trạng thái cơ sở dữ liệu. Điều này có thể được giải quyết theo hai cách: Trạng thái có thể được sao chép trên n nút máy chủ trong bộ nhớ để bảo vệ chống lại n - 1 lỗi nút đơn (ví dụ: HStore, VoltDB [29]) hoặc bằng cách đăng nhập vào bộ lưu trữ lâu bền (ví dụ: Redis hoặc SAP Hana). Thông qua ghi nhật ký, một mẫu truy cập ghi ngẫu nhiên có thể được chuyển đổi thành một mẫu tuần tự bao gồm các thao tác đã nhận và các thuộc tính liên quan của chúng (ví dụ: làm lại thông tin). Trong hầu hết các hệ thống NoSQL, quy tắc cam kết để ghi nhật ký làđược tôn trọng, điều này yêu cầu mọi thao tác ghi được xác nhận là thành công phải được ghi lại và nhật ký được chuyển sang bộ lưu trữ liên tục. Để tránh độ trễ quay của ổ cứng phát sinh bằng cách ghi nhật ký từng hoạt động riêng lẻ, các lần xóa nhật ký có thể được nhóm lại với nhau (cam kết nhóm), điều này làm tăng nhẹ độ trễ của từng thao tác ghi, nhưng cải thiện đáng kể thông lượng.

SSD và nói chung là tất cả các thiết bị lưu trữ dựa trên bộ nhớ flash NAND về cơ bản khác với HDD ở nhiều khía cạnh: “(1) tốc độ đọc và ghi không đối xứng, (2) không ghi đè tại chỗ – toàn bộ khối phải được xóa trước khi ghi đè bất kỳ trang trong khối đó, và (3) chu kỳ chương trình/xóa hạn chế” [40]. Do đó, Đối với cơ sở dữ liệu trong bộ nhớ, mẫu truy cập cập nhật tại chỗ là lý tưởng: Nó đơn giản hóa việc triển khai và ghi ngẫu nhiên vào RAM về cơ bản nhanh như ghi tuần tự, với những khác biệt nhỏ bị ẩn bởi đường ống dẫn và hệ thống phân cấp bộ đệm CPU. Tuy nhiên, RDBMS và nhiều hệ thống NoSQL (ví dụ: MongoDB) cũng sử dụng mẫu cập nhật tại chỗ để lưu trữ liên tục. Để giảm thiểu tốc độ truy cập ngẫu nhiên chậm vào bộ lưu trữ liên tục, bộ nhớ chính thường được sử dụng làm bộ đệm và được bổ sung bằng cách ghi nhật ký để đảm bảo độ bền. Trong các RDBMS, điều này đạt được thông qua một nhóm bộ đệm phức hợp không chỉ sử dụng các thuật toán thay thế bộ đệm thích hợp cho các mẫu truy cập dựa trên SQL điển hình mà cònđảm bảo ngữ nghĩa ACID. Cơ sở dữ liệu NoSQL có vùng đệm đơn giản hơn, thu lợi từ các truy vấn đơn giản hơn và thiếu các giao dịch ACID. Giải pháp thay thế cho mô hình vùng đệm là để lại bộ nhớ đệm cho HĐH thông qua bộ nhớ ảo (ví dụ: được sử dụng trong công cụ lưu trữ MMAP của MongoDB). Điều này đơn giản hóa kiến trúc cơ sở dữ liệu, nhưng có nhược điểm là cung cấp ít quyền kiểm soát hơn đối với mục hoặc trang dữ liệu nào nằm trong bộ nhớ và khi nào chúng bị loại bỏ. Ngoài ra, tính năng đọc trước (đọc suy đoán) và ghi sau (đệm ghi) được thực hiện minh bạch với bộ đệm hệ điều hành thiếu sự phức tạp vì chúng dựa trên logic hệ thống tệp thay vì truy vấn cơ sở dữ liệu. bộ phận quản lý lưu trữ của hệ thống cơ sở dữ liệu không được coi SSD và HDD là RAM liên tục, chậm hơn một chút, vì ghi ngẫu nhiên vào SSD chậm hơn gần một bậc so với ghi tuần tự. Mặt khác, các lần đọc ngẫu nhiên có thể được thực hiện mà không có bất kỳ hình phạt nào về hiệu suất. Có một số hệ thống cơ sở dữ liệu (ví dụ: Oracle Exadata, Aerospike) được thiết kế rõ ràng cho các đặc tính hiệu suất này của SSD. Trong ổ cứng, cả đọc và ghi ngẫu nhiên đều chậm hơn 10-100 lần so với truy cập tuần tự. Do đó, việc ghi nhật ký phù hợp với điểm mạnh của SSD và HDD, cả hai đều cung cấp thông lượng cao hơn đáng kể cho ghi tuần tự.

Lưu trữ chỉ nối thêm (còn được gọi là cấu trúc nhật ký) cố gắng tối đa hóa thông lượng bằng cách ghi tuần tự. Mặc dù các hệ thống tệp có cấu trúc nhật ký đã có một lịch sử nghiên cứu lâu dài, nhưng I/O chỉ nối thêm mới chỉ được phổ biến gần đây cho cơ sở dữ liệu bằng cách BigTable sử dụng các cây Hợp nhất có cấu trúc nhật ký (LSM) [9] bao gồm một bộ đệm trong bộ nhớ, một bộ lưu trữ - nhật ký lều và các tệp lưu trữ được ghi định kỳ, không thay đổi. Các cây LSM và các biến thể như Sorted Array Merge Trees (SAMT) và Cache-Oblivious Look-ahead Arrays (COLA) đã được áp dụng trong nhiều hệ thống NoSQL (Cassandra, CouchDB, LevelDB, Bitcask, RethinkDB, WiredTiger, RocksDB, InfluxDB, TokuDB) [31]. Thiết kế cơ sở dữ liệuđể đạt được hiệu suất ghi tối đa bằng cách luôn ghi vào nhật ký khá đơn giản, khó khăn nằm ở việc cung cấp các lần đọc tuần tự và ngẫu nhiên nhanh. Điều này yêu cầu một cấu trúc chỉ mục phù hợp được cập nhật vĩnh viễn dưới dạng cấu trúc dữ liệu sao chép khi ghi (COW) (ví dụ: COW B-tree của CouchDB) hoặc chỉ được duy trì định kỳ dưới dạng cấu trúc dữ liệu bất biến (ví dụ: trong các hệ thống kiểu BigTable). Một vấn đề của tất cả các phương pháp lưu trữ có cấu trúc nhật ký là bộ sưu tập rác (nén) tốn kém để lấy lại không gian của các mục đã cập nhật hoặc đã xóa.

Trong các môi trường ảo hóa như các đám mây Cơ sở hạ tầng dưới dạng Dịch vụ, nhiều đặc điểm được thảo luận của lớp lưu trữ bên dưới bị ẩn đi.

**3.3 Query processing (Xử lý truy vấn)**

Khả năng truy vấn của cơ sở dữ liệu NoSQL chủ yếu tuân theo mô hình phân phối, đảm bảo tính nhất quán và mô hình dữ liệu của nó. Tra cứu khóa chính, tức là truy xuất các mục dữ liệu bằng một ID duy nhất, được mọi hệ thống NoSQL hỗ trợ, vì nó tương thích với phân vùng phạm vi cũng như phân vùng băm. Truy vấn bộ lọc trả về tất cả các mục (hoặc phép chiếu) đáp ứng một vị từ được chỉ định trên các thuộc tính của các mục dữ liệu từ một bảng. Ở dạng đơn giản nhất, chúng có thể được thực hiện dưới dạng quét toàn bộ bảng được lọc. Đối với cơ sở dữ liệu được phân vùng băm, điều này ngụ ý một mẫu thu thập phân tán trong đó mỗi phân vùng thực hiện quá trình quét dự đoán và kết quả được hợp nhất. Đối với các hệ thống phân vùng phạm vi, bất kỳ điều kiện nào trên phạm vithuộc tính có thể được khai thác để chọn phân vùng.

Để tránh sự thiếu hiệu quả của các lần quét O(n), các chỉ mục phụ có thể được sử dụng. Đây có thể là các chỉ mục phụ cục bộ được quản lý trong mỗi phân vùng hoặc các chỉ mục phụ toàn cầu lập chỉ mục dữ liệu trên tất cả các phân vùng [4]. Vì bản thân chỉ mục toàn cầu phải được phân phối trên các phân vùng, nên việc duy trì chỉ mục thứ cấp nhất quán sẽ yêu cầu các giao thức cam kết chậm và có khả năng không khả dụng. Do đó, trên thực tế, hầu hết các hệ thống chỉ cung cấp tính nhất quán cuối cùng cho các chỉ mục này (ví dụ: Megastore, Google AppEngine Datastore, DynamoDB) hoặc hoàn toàn không hỗ trợ chúng (ví dụ: HBase, Azure Tables). Khi thực hiện các truy vấn toàn cầu trên các chỉ mục phụ cục bộ, truy vấn chỉ có thể được nhắm mục tiêu đến một chỉ mục phụ. Tập hợp các phân vùng nếu vị từ truy vấn và các quy tắc phân vùng giao nhau. Mặt khác, kết quả phải được tập hợp thông qua tập hợp phân tán. Ví dụ: một bảng người dùng có phân vùng theo phạm vi trên một trường tuổi có thể phục vụ các truy vấn có điều kiện bình đẳng về tuổi từ một phân vùng trong khi các truy vấn về tên cần được đánh giá ở mỗi phân vùng. Một trường hợp đặc biệt của lập chỉ mục phụ toàn cầu là tìm kiếm toàn văn bản, trong đó các trường đã chọn hoặc các mục dữ liệu hoàn chỉnh được đưa vào chỉ mục đảo ngược bên trong cơ sở dữ liệu (ví dụ: MongoDB) hoặc vào một nền tảng tìm kiếm bên ngoài như ElasticSearch hoặc Solr (Riak Search, DataStax Cassandra).

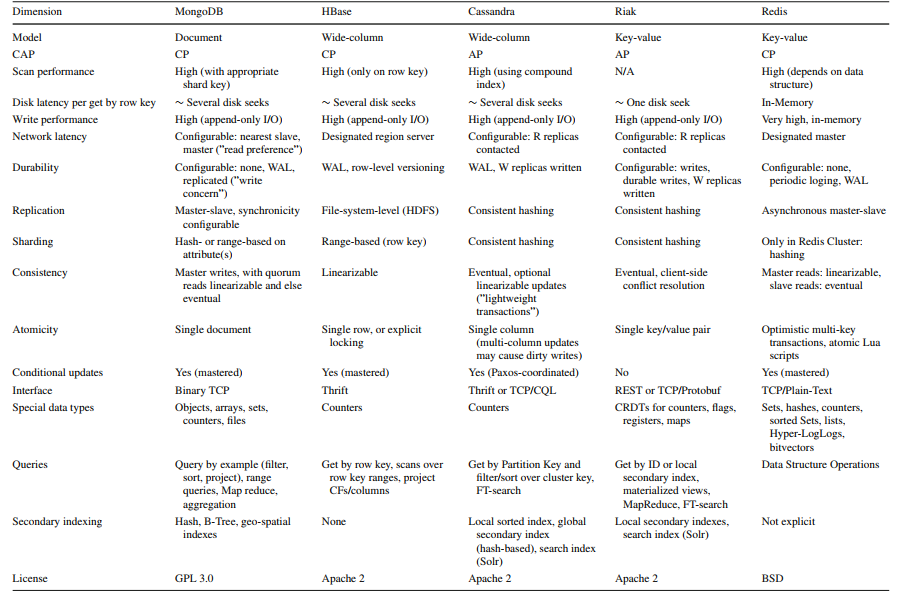
Lập kế hoạch truy vấn là nhiệm vụ tối ưu hóa kế hoạch truy vấn để giảm thiểu chi phí thực hiện [25]. Đối với các tập hợp và liên kết, lập kế hoạch truy vấn là cần thiết vì các truy vấn này rất kém hiệu quả và khó thực hiện trong mã ứng dụng. Sự phong phú của tài liệu và kết quả xử lý truy vấn quan hệ phần lớn bị bỏ qua trong các hệ thống NoSQL hiện tại vì hai lý do. Đầu tiên, mô hình khóa-giá trị và cột rộng được tập trung xung quanh CRUD và quét các hoạt động trên các khóa chính để lại ít chỗ cho việc tối ưu hóa truy vấn. Thứ hai, hầu hết công việc về xử lý truy vấn phân tán tập trung vào khối lượng công việc OLAP (xử lý phân tích trực tuyến) ưu tiên thông lượng hơn độ trễ trong khi việc tối ưu hóa truy vấn một nút không dễ dàngáp dụng cho cơ sở dữ liệu được phân vùng và sao chép. Tuy nhiên, nó vẫn là một thách thức nghiên cứu mở để khái quát hóa phần lớn các kỹ thuật tối ưu hóa truy vấn có thể áp dụng, đặc biệt là trong bối cảnh cơ sở dữ liệu tài liệu.

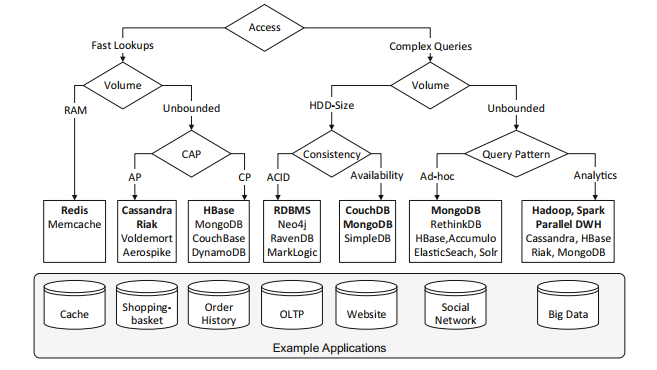
Phân tích trong cơ sở dữ liệu có thể được thực hiện nguyên bản (ví dụ: trong MongoDB, Riak, CouchDB) hoặc thông qua các nền tảng phân tích bên ngoài như Hadoop, Spark và Flink (ví dụ: trong Cassandra và HBase). Sự trừu tượng hóa phân tích hàng loạt gốc phổ biến được các hệ thống NoSQL đưa ra là MapReduce [17]. Do I/O, chi phí giao tiếp và tối ưu hóa kế hoạch thực hiện hạn chế, các phương pháp tiếp cận theo định hướng lô và vi mô này có thời gian phản hồi cao. Chế độ xem cụ thể hóa là một giải pháp thay thế với thời gian phản hồi truy vấn thấp hơn. Chúng được khai báo tại thời điểm thiết kế và được cập nhật liên tục khi thay đổi hoạt động (ví dụ: trong CouchDB và Cassandra). Tuy nhiên, tương tự như lập chỉ mục thứ cấp toàn cầu, tính nhất quán của chế độ xem thường làthoải mái để ghi nhanh, khả dụng cao, khi hệ thống được phân phối. Vì chỉ có một số hệ thống cơ sở dữ liệu có hỗ trợ tích hợp để nhập và truy vấn các luồng dữ liệu không giới hạn, nên các đường dẫn phân tích gần thời gian thực thường triển khai Kiến trúc Lambda [39] hoặc Kiến trúc Kappa [33]: kiến trúc trước đây bổ sung cho một khung xử lý hàng loạt như Hadoop MapReduce với bộ xử lý luồng chẳng hạn như Storm [6] và khung sau hoàn toàn dựa vào xử lý luồng và bỏ qua xử lý hàng loạt hoàn toàn.

**Bảng 1**. So sánh trực tiếp các yêu cầu chức năng, yêu cầu và kỹ thuật phi chức năng giữa MongoDB, Redis, HBase, Riak, Cassandra và MySQL theo Hộp công cụ NoSQL của chúng tôi



**Bảng 2** So sánh định tính của MongoDB, HBase, Cassandra, Riak và Redis





**Hình 6** Một cây quyết định để ánh xạ các yêu cầu đến hệ thống cơ sở dữ liệu (NoSQL)

**4. Nghiên cứu điển hình về hệ thống**

Trong phần này, chúng tôi cung cấp một so sánh định tính của một số giá trị chính nổi bật nhất, tài liệu và cột rộng. Chúng tôi trình bày kết quả trong các so sánh cô đọng mạnh mẽ và tham khảo các tài liệu của các hệ thống riêng lẻ để biết thông tin không chính xác. NoSQL được đề xuất Hộp công cụ (xem Hình 4, trang 5) là một phương tiện trừu tượng có thể được sử dụng để phân loại các hệ thống cơ sở dữ liệu theo ba chiều: yêu cầu chức năng, yêu cầu phi chức năng và các kỹ thuật được sử dụng để thực hiện chúng. Chúng tôi lập luận rằng phân loại này đặc trưng tốt cho nhiều hệ thống cơ sở dữ liệu và do đó có thể được sử dụng để đối chiếu có ý nghĩa các hệ thống cơ sở dữ liệu khác nhau: Bảng 1 cho thấy sự so sánh trực tiếp của MongoDB, Redis, HBase, Riak, Cassandra và MySQL trong cấu hình mặc định tương ứng của chúng. Một so sánh chi tiết hơn về các thuộc tính hệ thống trung tâm được trình bày trong Bảng 2 (xem trang 11).

Phương pháp được sử dụng để xác định các thuộc tính hệ thống cụ thể bao gồm phân tích sâu sắc các tài liệu và tài liệu có sẵn công khai về các hệ thống. Hơn nữa, một số thuộc tính phải được đánh giá bằng cách nghiên cứu các cơ sở mã nguồn mở, giao tiếp cá nhân với các nhà phát triển cũng như phân tích tổng hợp các báo cáo và điểm chuẩn của các học viên.

So sánh làm sáng tỏ cách cơ sở dữ liệu SQL và NoSQL được thiết kế để đáp ứng các nhu cầu rất khác nhau: RDBMS cung cấp mức chức năng chưa từng có trong khi cơ sở dữ liệu NoSQL vượt trội về mặt phi chức năng thông qua khả năng mở rộng, tính khả dụng, độ trễ thấp và thông lượng cao. Tuy nhiên, cũng có sự khác biệt lớn giữa các cơ sở dữ liệu NoSQL. Ví dụ, Riak và Cassandra có thể được cấu hình để đáp ứng nhiều yêu cầu phi chức năng, nhưng cuối cùng chỉ nhất quán và không có nhiều khả năng chức năng ngoài phân tích dữ liệu và, trong trường hợp Cassandra, cập nhật có điều kiện. MongoDB và HBase, mặt khác, cung cấp tính nhất quán mạnh mẽ hơn và các khả năng chức năng phức tạp hơn như truy vấn quét và chỉ MongoDB: truy vấn bộ lọc, nhưng không duy trì tính khả dụng đọc và ghi trong các phân vùng và có xu hướng hiển thị độ trễ đọc cao hơn. Redis, với tư cách là hệ thống không phân vùng duy nhất trong so sánh này ngoài MySQL, cho thấy một tập hợp các đánh đổi đặc biệt tập trung vào khả năng duy trì thông lượng cực cao ở độ trễ thấp bằng cách sử dụng cấu trúc dữ liệu trong bộ nhớ và sao chép master-slave không đồng bộ.

**5. Kết luận**

Chọn một hệ thống cơ sở dữ liệu luôn có nghĩa là chọn một tập hợp các thuộc tính mong muốn hơn một thuộc tính khác. Để phá vỡ sự phức tạp của lựa chọn này, chúng tôi trình bày một cây quyết định nhị phân trong Hình 6 lập bản đồ các quyết định đánh đổi để làm ví dụ các ứng dụng và hệ thống cơ sở dữ liệu có khả năng phù hợp. Các nút lá bao gồm các ứng dụng khác nhau, từ bộ nhớ đệm đơn giản (trái) đến phân tích BigData (phải). Đương nhiên, quan điểm này về vấn đề 123 F. Gessert et al. không gian chưa hoàn chỉnh, nhưng nó mơ hồ chỉ ra một giải pháp cho một vấn đề quản lý dữ liệu cụ thể.

Sự phân chia đầu tiên trong cây là dọc theo mẫu hình truy cập của các ứng dụng: chúng chỉ dựa vào tra cứu nhanh (nửa bên trái) hoặc yêu cầu khả năng truy vấn phức tạp hơn (nửa bên phải). Các ứng dụng tra cứu nhanh có thể được phân biệt hơn nữa bởi khối lượng dữ liệu mà chúng xử lý: nếu bộ nhớ chính của một máy duy nhất có thể chứa tất cả dữ liệu, một hệ thống nút đơn như Redis hoặc Memcache có lẽ là lựa chọn tốt nhất, tùy thuộc vào chức năng (Redis) hay đơn giản (Memcache) được ưa chuộng. Nếu khối lượng dữ liệu đang hoặc có thể tăng vượt quá dung lượng RAM hoặc thậm chí không bị giới hạn, một hệ thống đa nút có quy mô theo chiều ngang có thể phù hợp hơn. Quyết định quan trọng nhất trong trường hợp này là có nên ủng hộ tính khả dụng (AP) hay tính nhất quán (CP) như được mô tả trong định lý CAP. Các hệ thống như Cassandra và Riak có thể mang lại trải nghiệm luôn sẵn sàng, trong khi các hệ thống như HBase, MongoDB và DynamoDB mang lại tính nhất quán cao.

Nửa bên phải của cây bao gồm các ứng dụng yêu cầu truy vấn phức tạp hơn so với tra cứu đơn giản. Ở đây cũng vậy, trước tiên chúng ta phân biệt các hệ thống bằng khối lượng dữ liệu mà chúng phải xử lý theo việc liệu các hệ thống nút đơn có khả thi (kích thước ổ cứng) hay cần phân phối (khối lượng không giới hạn). Đối với khối lượng công việc OLTP (xử lý giao dịch trực tuyến) phổ biến trên khối lượng dữ liệu lớn vừa phải, RDBMS truyền thống hoặc cơ sở dữ liệu đồ thị như Neo4J là tối ưu, vì chúng cung cấp ngữ nghĩa ACID. Tuy nhiên, nếu tính khả dụng là bản chất, các hệ thống phân tán như MongoDB, CouchDB hoặc DocumentDB được ưu tiên hơn.

Nếu khối lượng dữ liệu vượt quá giới hạn của một máy, việc lựa chọn hệ thống phù hợp phụ thuộc vào mẫu truy vấn phổ biến: Khi các truy vấn phức tạp phải được tối ưu hóa cho độ trễ, ví dụ như trong các ứng dụng mạng xã hội, MongoDB rất hấp dẫn, vì nó tạo điều kiện cho các truy vấn đặc biệt là biểu cảm. HBase và Cassandra cũng hữu ích trong trường hợp như vậy, nhưng vượt trội về thông lượng phân tích Dữ liệu lớn được tối ưu hóa, khi kết hợp với Hadoop.

Tóm lại, chúng tôi tin rằng mô hình từ trên xuống được đề xuất là một hỗ trợ quyết định hiệu quả để lọc số lượng lớn các hệ thống cơ sở dữ liệu NoSQL dựa trên các yêu cầu trung tâm. Hộp công cụ NoSQL hơn nữa cung cấp ánh xạ từ các yêu cầu chức năng và phi chức năng đến các kỹ thuật triển khai phổ biến để phân loại không gian NoSQL không ngừng phát triển.

**Tham khảo**

1. Abadi D (2012) Consistency tradeoffs in modern distributed data-base system design: cap is only part of the story. Computer45(2)
2. Attiya H, Bar-Noy A et al (1995) Sharing memory robustly in message-passing systems. JACM 42(1)
3. Bailis P, Kingsbury K (2014) The network is reliable. Commun ACM 57(9)
4. Baker J, Bond C, Corbett JC et al (2011) Megastore: providing scalable, highly available storage for interactive services. In: CIDR, pp 223–234
5. Bernstein PA, Cseri I, Dani N et al (2011) Adapting microsoft sql server for cloud computing. In: 27th ICDE, pp 1255–1263 IEEE
6. Boykin O, Ritchie S, O’Connell I, Lin J (2014) Summingbird: a framework for integrating batch and online mapreduce computations. VLDB 7(13)
7. Brewer EA (2000) Towards robust distributed systems
8. Calder B, Wang J, Ogus A et al (2011) Windows azure storage: a highly available cloud storage service with strong consistency. In: 23th SOSP. ACM
9. Chang F, Dean J, Ghemawat S et al (2006) Bigtable: a distributed storage system for structured data. In: 7th OSDI, USENIX Asso- ciation, pp 15–15
10. Charron-Bost B, Pedone F, Schiper A (2010) Replication: theory and practice, lecture notes in computer science, vol. 5959. Springer
11. Cooper BF, Ramakrishnan R, Srivastava U et al (2008) Pnuts: Yahoo!’s hosted data serving platform. Proc VLDB Endow 1(2):1277–1288
12. Corbett JC, Dean J, Epstein M, et al (2012) Spanner: Google’s globally-distributed database. In: Proceedings of OSDI, USENIX Association, pp 251–264
13. Curino C, Jones E, Popa RA et al. (2011) Relational cloud: a database service for the cloud. In: 5th CIDR
14. Das S, Agrawal D, El Abbadi A et al (2010) G-store: a scalable data store for transactional multi key access in the cloud. In: 1st SoCC, ACM, pp 163–174
15. Davidson SB, Garcia-Molina H, Skeen D et al (1985) Consistency in a partitioned network: a survey. SUR 17(3):341–370
16. Dean J (2009) Designs, lessons and advice from building large distributed systems. Keynote talk at LADIS 2009
17. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. COMMUN ACM 51(1)
18. DeC andia G, Hastorun D et al (2007) Dynamo: amazon’s highly available key-value store. In: 21th SOSP, ACM, pp 205–220
19. Fischer MJ, Lynch NA, Paterson MS (1985) Impossibility of distributed consensus with one faulty process. J ACM 32(2):374–382
20. Gessert F, Schaarschmidt M, Wingerath W, Friedrich S, Ritter N (2015) The cache sketch: Revisiting expiration-based caching in the age of cloud data management. In: BTW, pp 53–72
21. Gilbert S, Lynch N (2002) Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2):51–59
22. Gray J, Helland P (1996) The dangers of replication and a solution. SIGMOD Rec 25(2):173–182
23. Haerder T, Reuter A (1983) Principles of transaction-oriented database recovery. ACM Comput Surv 15(4):287–317
24. Hamilton J (2007) On designing and deploying internet-scale ser-vices. In: 21st LISA. USENIX Association
25. Hellerstein JM, Stonebraker M, Hamilton J (2007) Architecture of a database system. Now Publishers Inc
26. Herlihy MP, Wing JM (1990) Linearizability: a correctness condition for concurrent objects. TOPLAS 12
27. Hoelzle U, Barroso LA (2009) The Datacenter As a Computer: an introduction to the design of warehouse-scale machines. Morgan and Claypool Publishers
28. Hunt P, Konar M, Junqueira FP, Reed B (2010) Zookeeper: wait- free coordination for internet-scale systems. In: USENIXATC. USENIX Association
29. Kallman R, Kimura H, Natkins J et al (2008) H-store: a high- performance, distributed main memory transaction processing system. VLDB Endowment
30. Karger D, Lehman E, Leighton T et al (1997) Consistent hashing and random trees: distributed caching protocols for relieving hotspots on the world wide web. In: 29th STOC, ACM
31. Kleppmann M (2016) Designing data-intensive applications. O Reilly, to appear
32. Kraska T, Pang G, Franklin MJ et al (2013) Mdcc: Multi-data center consistency. In: 8th EuroSys, ACM
33. Kreps J (2014) Questioning the lambda architecture. Accessed: 17 Dec 2015
34. Lakshman A, Malik P (2010) Cassandra: a decentralized structured storage system. SIGOPS Oper Syst Rev 44(2):35–40
35. Laney D (2001) 3d data management: Controlling data volume, velocity, and variety. Tech. rep, META Group
36. Lloyd W, Freedman MJ, Kaminsky, M et al (2011) Don’t settle for eventual: scalable causal consistency for wide-area storage with cops. In: 23th SOSP. ACM
37. Mahajan P, Alvisi L, Dahlin M et al (2011) Consistency, availability, and convergence. University of Texas at Austin Tech Report 11
38. Mao Y, Junqueira FP, Marzullo K (2008) Mencius: building effi- cient replicated state machines for wans. OSDI 8:369–384
39. Marz N, Warren J (2015) Big data: principles and best practices of scalable realtime data systems. Manning Publications Co
40. Min C, Kim K, Cho H et al (2012) Sfs: random write considered harmful in solid state drives. In: FAST
41. Özsu MT, Valduriez P (2011) Principles of distributed database systems. Springer Science & Business Media
42. Pritchett D (2008) Base: an acid alternative. Queue 6(3):48–55
43. Qiao L, Surlaker K, Das S et al (2013) On brewing fresh espresso: Linkedin’s distributed data serving platform. In: SIGMOD, ACM, pp 1135–1146
44. Sadalage PJ, Fowler M (2013) NoSQL distilled : a brief guide to the emerging world of polyglot persistence. Addison-Wesley, Upper Saddle River
45. Shapiro M, Preguica N, Baquero C et al (2011) A comprehensive study of convergent and commutative replicated data types. Ph.D. thesis, INRIA
46. Shukla D, Thota S, Raman K et al (2015) Schema-agnostic indexing with azure documentdb. PVLDB 8(12)
47. Sovran Y, Power R, Aguilera MK, Li J (2011) Transactional storage for geo-replicated systems. In: 23th SOSP, ACM, pp 385–400
48. Stonebraker M, Madden S, Abadi DJ et al (2007) The end of an architectural era: (it’s time for a complete rewrite). In: 33rd VLDB, pp 1150–1160
49. Wiese L et al (2015) Advanced Data Management: For SQL. Cloudand Distributed Databases. Walter de Gruyter GmbH & Co KG, NoSQL
50. Zhang H, Chen G et al (2015) In-memory big data management and processing: a survey. TKDE