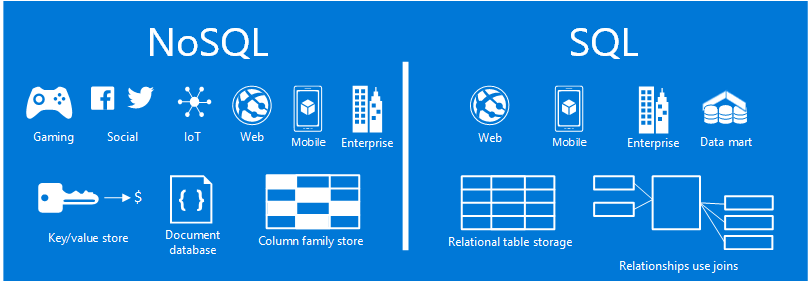
# NoSQL

Cơ Sở dữ liệu quan hệ được thiết kế cho những mô hình cần đảm bảo tính chặt chẽ và dữ liệu không quá lớn, trong khi các dịch vụ mạng xã hội lại có một lượng lớn dữ liệu và được cập nhật liên tục do số lượng người dùng quá nhiều. Do đó cơ sở dữ liệu NOSQL sinh ra để giải quyết các vấn đề mà RDBMS (A relational database management system) đã bộc lộ những yếu kém như: tốc độ thực thi, khả năng lưu trữ, các nghiệp vụ phức tạp (như phân trang, đánh chỉ mục …) Nhờ vậy giải pháp sử dụng cơ sở dữ liệu NOSQL sẽ mang lại một chi phí thấp hơn nếu so sánh với RDBMS truyền thống.



## Định Nghĩa

NoSQL là một xu hướng cơ sở dữ liệu mà không dùng dữ liệu quan hệ để quản lý dữ liệu trong lĩnh vực phần mềm. NOSQL có nghĩa là Non-Relational – không ràng buộc. Tuy nhên, thuật ngữ đó ít phổ biến hơn và ngày nay người ta thường dịch thành Not Only SQL – không chỉ SQL.

NOSQL được xem như thế hệ database kế tiếp của RDBMS, là một thế hệ cơ sở dữ liệu Non-relational (không ràng buộc), distributed (phân tán), open source (mã nguồn mở), horizontal scalable (khả năng mở rộng theo chiều ngang), có độ chịu tải, lỗi cao.

## Một Số Thuật Ngữ Trong NoSQL

* Tính ràng buộc (Relational): Thuật ngữ để mô tả tính ràng buộc giữa các bảng trong cơ sở dữ liệu quan hệ (MySQL, SqlSever, PostgreSQL…).
* Không ràng buộc (Non-relational): Dữ liệu sẽ không có các ràng buộc giữa các bảng nữa, mà dữ liệu sẽ ở dạng Json hoặc Bson.
* Khả năng mở rộng (High Scalability): Khi hệ thống lớn lên, ta có thể bổ sung thêm các Nodes mới, các Sever mới để chia tải hoặc chia dữ liệu. Để hệ thống không bị quá tải.
* Khả năng mở rộng theo chiều dọc (Vertical Scalable / Scale Up): Là việc nâng cấp phần cứng hệ thống bằng việc nâng cấp RAM, hay bộ nhớ.
* Khả năng mở rộng theo chiều ngang (Horizontal Scalable / Scale-Out): Là việc bổ sung phần cứng tránh sự hoạt động quá tải của hệ thống.
* Phân tán dữ liệu (Distributed Data): Là việc mô tả dữ liệu được phân tán ở các địa điểm khác nhau.
* Triển khai linh hoạt (Deployment Flexibilitty): Dễ dàng mở rộng thêm các nodes (Severs) mà không ảnh hưởng đến hoạt động của hệ thống.
* Tính sẵn sàng (High Availability): Hệ thống sẽ không bị ảnh hưởng khi một node bị trục trặc.
* Nhất quán cuối (Eventual Consistency): Khi ta đưa một dữ liệu mới vào một node của hệ thống, dữ lệu sẽ được lan truyền sang các node khác của hệ thống và cuối cùng tất cả node sẽ được đồng bộ.
* Lưu trữ tốt (Durability).

**Khi làm việc với Nosql ta sẽ gặp một số khái niệm sau:**

* FIELDS: Tương đương với khái niệm Columns trong SQL
* Documents: Thay thế khái niệm Rows trong SQL. Đây cũng chính là khái niệm làm nên sự khác biệt giữa NOSQL và SQL, 1 document chưa số cột (fields) không cố định trong khi 1 row thì số cột (columns) là định sẵn trước.
* Collection: Tương đương với khái niệm table trong SQL. Một Collection là tập hợp các documents.
* Key-Value: Cặp khóa - giá trị được dùng để lưu trữ dữ liệu trong NOSQL
* Cursor: Tạm dịch là con trỏ, sử dụng cursor để lấy dữ liệu từ database.

Trong các hệ cơ sở dữ liệu quan hệ, các cột được định nghĩa theo bảng, còn với hệ cơ sở dữ liệu không ràng buộc, các cột được định nghĩa ở mỗi document. Bởi thế, các document quản lý gần như tất cả, các collection không cần quản lý chặt chẽ những gì đang xảy ra trong nó nữa.

|  |  |
| --- | --- |
| RDBMS | NOSQL |
| Columns  Row  Table  Schema | Fields  Documents  Collection  Free Schema |

## Đặc Điểm

- NoSQL lưu trữ dữ liệu của mình theo dạng cặp giá trị “key – value”. Sử dụng số lượng lớn các node để lưu trữ thông tin.

- Chấp nhận dữ liệu bị trùng lặp do một số node sẽ lưu cùng thông tin giống nhau.

- Phi quan hệ – không có ràng buộc nào cho việc nhất quán dữ liệu.

- Có hiệu suất cao (High performance) và tính sẵn sàng cao (High availability).

## So Sánh Giữa SQL và NoSQL

|  |  |  |
| --- | --- | --- |
| Tính Năng | SQL | NoSQL |
| Hiệu xuất | Kém hơn NoSQL vì khi truy vấn nó phải tính toán, kiểm tra và xử lý mối quan hệ các bảng | Tốt hơn SQL do nó bỏ qua xử lý các ràng buộc giữa các bảng |
| Mở rộng theo chiều ngang | Có thể thực hiện được nhưng sẽ rất phức tạp nếu đã tồn tại dữ liệu trong database | Mở rộng dễ dàng mà không ảnh hưởng đến hệ thống |
| Tốc độ Đọc/Ghi | Kém hơn NoSQL do phải đảm bảo tính ràng buộc giữa các bảng | Tốc độ nhanh hơn SQL nhiều vì bỏ qua cơ chế ràng buộc giữa các bảng  Vì dữ liệu được lưu trong RAM ,sau đó đẩy xuống ổ cứng và cuối cùng là nhất quán cuối |
| Phần cứng | Đòi hỏi phần cứng cao | Không đòi hỏi cao về phần cứng |
| Thay đổi số node trong hệ thống | Vì tính nhất quán về dữ liệu nên khi thêm hoặc xóa một node thì hệ thống cần phải shutdown trong một khoảng thời gian | Vì tính nhất quán cuối nên không cần phải shutdown hệ thống |
| Truy vấn và Báo cáo | Dễ dàng, sử dụng ngôn ngữ SQL Query để truy vấn trực tiếp dữ liệu từ Database hoặc dùng công cụ hỗ trợ để lấy báo cáo | Việc lấy báo cáo dữ liệu trực tiếp từ NoSQL chưa được hỗ trợ tốt, thực hiện chủ yếu thông qua giao diện ứng dụng |
| Mở rộng dữ liệu | Khi muốn bổ sung thêm cột cho một bảng, cần phải khai báo trước | Không cần khai báo trước khi muốn bổ sung dữ liệu |
| Ứng Dụng | Sử dụng để xây dựng những hệ thống có quan hệ chặt chẽ và cần tính đồng nhất về dữ liệu như: tài chính, ngân hàng, chứng khoán… | Sử dụng xây dựng những hệ thống lưu trữ thông tin lớn, không quá quan trọng trong vấn đề đồng nhất dữ liệu trong 1 thời gian nhất định. |
| Ví dụ | **Facebook**: Sử dụng Hive(một dạng biến thể của SQL)  **Twitter:** sử dụng MySQL  **Stack Overflow**: sử dụng SQL Sever  **Youtube**: sử dụng MySQL  **Wikipedia:** sử dụng MySQL | **Facebook**: Sử dụng Cassandra(cơ sở dữ liệu đa chiều, lưu trữ phân tán)  **Twitter:** sử dụng cassandra |

## Ưu Và Nhược Điểm Của Cơ Sở Dữ Liệu NoSQL

**Hiệu suất hoạt động cao:** NoSQL có hiệu suất hoạt động cao, lưu trữ lượng lớn dữ liệu để đáp ứng nhu cầu lưu trữ ngày càng tăng hiện nay. Tuy nhiên để đạt được này, cần phải loại bỏ đi một số thứ như: các ràng buộc giữa các bảng, tính nhất quán dữ liệu, ngôn ngữ truy vấn SQL. Đồng thời NoSQL có một số cải tiến mới như sử dụng tốt index, khả năng phân tán dễ dàng đã giúp NoSQL có một hiệu suất hoạt động cao.

**Khả năng phân trang:** phân trang trong cơ sở dữ liệu khá khó khăn khi không có một phương pháp chính thống nào để phục vụ cho việc này. Người lập trình phải dùng các phương pháp khác nhau để có thể lấy đúng số item cần lấy.

**NoSQL là nguồn mở:** các sản phẩm nguồn mở đưa ra cho những người phát triển với nhiều lợi ích to lớn, ví dụ như việc sử dụng miễn phí. Ngoài ra, phần mềm mã nguồn mở có xu hướng tin cậy hơn, an ninh hơn và nhanh hơn để triển khai so với các lựa chọn thay thế sở hữu độc quyền. Ví dụ như các hệ quản trị cơ sở dữ liệu NoSQL như: Cassandra, CouchDB, Hbase, MongoDB, RavenDB và Redis.

**Việc mở rộng phạm vi là mềm dẻo:** Thay vì bổ sung thêm các máy chủ lớn hơn để điều khiển nhiều dữ liệu hơn, thì NoSQL cho phép một công ty phân tán dữ liệu qua nhiều máy chủ khi mà tải gia tăng.

**Cassandra**

Cassandra là một cơ sở dữ liệu hướng cột, phân tán mã nguồn mở được thiết kế để xử lý một khối lượng lớn dữ liệu giàn trải trên nhiều node mà vẫn đảm bảo tính sẵn sàng cao (Highly Availability), khả năng mở rộng hay thu giảm số node linh hoạt (Elastic Scalability) và chấp nhận một số lỗi (Fault Tolerant). Nó được phát triển bởi Facebook và vẫn còn tiếp tục phát triển và sử dụng cho mạng xã hội lớn nhất thới giới này. Năm 2008, Facebook chuyển nó cho cộng đồng mã nguồn mở và được Apache tiếp tục phát triển đến ngày hôm nay. Cassandra được coi là sự kết hợp của Amazon’s Dynamo và Google’s BigTable. Các nút máy chủ trong cụm Cassandra là đồng nhất theo thiết kế ngang hàng (peer-to-peer), không có bất cứ thành phần nào trong hệ thống là điểm hỏng thắt cổ chai (bottle-neck).



*Hình 1:*

Thiết kế của Cassandra là thiết kế phân tán trên hàng ngàn máy chủ mà không có bất cứ điểm chết tập trung nào. Cassandra có thiết kế dựa trên kiến trúc mạng ngang hàng (Peer - to - Peer) tất cả các nút máy chủ trong hệ thống đều có vai trò như nhau và không có nút máy chủ nào đóng vai trò là máy chủ trung tâm mà việc hỏng hóc của máy chủ này có thể kéo theo đánh sập hoàn toàn hệ thống như các kiến trúc chủ - khách truyền thống. Các nút máy chủ của Cassandra là độc lập và tham gia vào kết nối với các nút máy chủ khác trong hệ thống. Mỗi nút đều có thể xử lý các thao tác ghi và đọc dữ liệu, không phân biệt là dữ liệu được lưu trữ một cách vật lý trên máy chủ nào trong hệ thống. Khi một nút trong hệ thống bị hỏng hóc và dừng hoạt động, các thao tác đọc ghi dữ liệu có thể được xử lý bởi các nút khác trong hệ thống. Quá trình này hoàn toàn trong suốt với ứng dụng cho phép ẩn đi hỏng hóc của hệ thống đối với các ứng dụng đó. Trong Cassandra, mỗi đối tượng dữ liệu có thể được nhân bản và lưu giữ trên nhiều máy chủ. Nếu một trong các máy chủ lưu một phiên bản dữ liệu bị lỗi hoặc không phải là phiên bản được cập nhật dữ liệu mới nhất, Cassandra có cơ chế đồng bộ để luôn đảm báo các thao tác đọc sẽ luôn trả về dữ liệu mới nhất. Cơ chế này được thực thi trong quá trình đọc dữ liệu (read repair) thay vì đồng bộ ngay trong thao tác ghi dữ liệu, điều này cho phép tăng hiệu năng cho thao tác ghi dữ liệu.

**Phân tán dữ liệu trong Cassandra**

Cassandra sử dụng cơ chế hàm băm nhất quán phân tán (Distributed consistent hashing) tổ chức các nút máy chủ thành cụm theo định dạng vòng tròn và dữ liệu được phân tán theo vòng tròn này theo hàm băm nhất quán. Mỗi vòng tròn được coi là một Datacenter.



*Hình 1: phân tán dữ liệu trên cassandra*

Các nút trong một cụm Cassandra sẽ được phân bố trên một vòng tròn gọi là **ring** *(như hình trên)*. Mỗi nút sẽ được gán với 1 giá trị **key**, Cassandra dùng 127 bit để tạo ra key này. Mỗi nút trong **ring** sẽ quản lý một phạm vi giá trị của các **key**. Phạm vi của **key** được xác định trải đều từ giá trị của chính nút đó nắm giữ, đi ngược lại chiều kim đồng hồ cho đến khi gặp nút đầu tiên thì dừng lại. Đối chiếu lên hình, ta sẽ thấy rằng phạm vi các **key** mà nút ***T-1*** quản lý nằm trong vùng ***(T-0; T-1].*** Khi một bản ghi được ghi vào cụm Cassandra. Trường khóa của bản ghi đó sẽ được đi qua một hàm băm nhất quán, trả về một giá trị **key** 127bit, giá trị **key** này nằm trong vùng kiểm soát của nút nào thì bản ghi đó sẽ được ghi vào nút đấy. Ví dụ ta có giá trị trên các trường name được băm ra như bảng sau:

|  |  |
| --- | --- |
| Name | Hash Value |
| Phúc | -2245452657672322382 |
| Ngọc | 7723358928203680754 |
| Quân | -6756552657672322382 |
| Trâm | 1168658928203680754 |

Và ta có một cụm Cassandra với 4 nút được phân bố trên **ring** như hình sau:



*Hình 2: Cụm Cassandra với 4 nút được phân bố trên ring*

Với Cassandra, chúng ta có hai chiến lược phân mảnh (xác định vị trí của từng nút trong ***ring***)

**Random partitioning:** Đây là chiến lược mặc định và được đề xuất của Cassandra, vị trí của các node được xác định hoàn toàn thông qua mảng băm MD5. Phạm vi khóa nằm trong khoảng từ **0** tới **.**

**Ordered partitioning:** Chiến thuật phân mảnh đảm bảo các nút được sắp xếp theo thứ tự và phạm vi key mà mỗi nút sở hữu là như nhau.

Với chiến lược phân mảnh thứ nhất, nếu như các giá trị băm xuất ra giúp cho việc đặt các nút trong vòng phù hợp thì tất cả các bản ghi sẽ được phân bố đều trên toàn cụm. Việc thêm hay bớt mỗi nút ra khỏi cụm cũng dễ dàng hơn do không phải phân bố lại vị trí các nút khác. Với chiến lược phân mảnh thứ hai, khi mà các nút được phân bố đều vả phạm vi quản lý **key** là như nhau, những điều đó lại mang lại những bất lợi khá rõ ràng: Khó cân bằng trong cụm: Mỗi khi thêm hay bớt một nút khỏi cụm, người quản trị sẽ phải tự tái cân bằng cụm lại một cách thủ công để đảm bảo các nút phân bố đều. Nếu dữ liệu được ghi tuần tự, có thể xảy ra trường hợp hàng loạt dữ liệu được ghi vào một nút. Gây mất cân bằng trong cụm.

**Nhận xét:** Với cả hai chiến lược phân mảnh trên, vẫn để lộ ra những điểm yếu, khi số lượng nút trong vòng quá ít, hoặc các nút phân bố không đều theo giá trị băm của các bảng ghi đưa vào, rất dễ đưa đến hiện tượng mất cân bằng, quá tải trong cụm. Ngoài ra, khi thêm hay xóa một nút khỏi vòng, thì sẽ phải mất công tái cân bằng lại cụm.

**Nút ảo**

Để giải quyết vấn đề này, ta có một giải pháp đó là sử dụng ***nút ảo***. Nút ảo trông giống như một thành phần của vòng tròn trong hệ thống, nhưng bản chất nút ảo chỉ là ánh xạ của một nút vật lý đến một địa chỉ khác trong vòng. Khi dữ liệu đi vào vùng quản lý của nút ảo, nó sẽ được đưa về lưu trữ tại nút vật lý của nút ảo đó. Mỗi nút vật lý khi tham gia vào vòng sẽ được gán một vị trí của chính nút đó và gán thêm một số lượng các vị trí khác (được coi như là nút ảo của nút đó). Cassandra cấu hình mặc định mỗi một nút tham gia vòng sẽ được gán 256 nút ảo trong vòng.



*Hình 3:*

Hình trên thể hiện một vòng trong có 4 nút vật lý, mỗi nút được gán thêm 7 nút ảo, như vậy tổng cộng trên vòng tròn sẽ có 32 phân vùng ***key***. Vậy tác dụng của nút ảo là gì, khi việc phân tán đều các nút ảo ra khắp vòng, số lượng nút tăng lên khiến cho các phân vùng key bé lại, việc phân vùng key bé lại mang ý nghĩa rất lớn trong việc phân bổ dữ liệu của cụm Cassandra, việc phân vùng nhỏ lại và các nút sát nhau hơn đưa hệ thống càng gần đến với việc tất cả dữ liệu sẽ được phân bổ đều khắm các nút, xác suất dữ liệu được đưa vào các nút là cân bằng nhau khi mà trên một khoảng key nhỏ ta có đầy đủ các nút ảo hoặc nút vật lý. Trường hợp hoàn hảo nhất là các nút vật lý đều có thành phần hiện diện của mình đêu khắp trên vòng. Phân vùng key quản lý khi có và không có nút ảo.

**Nhân bản dữ liệu trong Cassandra**

Để thỏa mãn tính sẵn sàng và liên tục trong Cassandra, mỗi đối tượng dữ liệu có thể được nhân bản và lưu giữ trên nhiều máy chủ. Nếu một trong các máy chủ lưu một phiên bản dữ liệu bị lỗi hoặc là phiên bản cũ, không phải là phiên bản được cập nhật dữ liệu mới nhất, Cassandra có cơ chế đồng bộ để luôn đảm báo các thao tác đọc sẽ luôn trả về dữ liệu mới nhất. Đồng thời với việc này Cassandra tiến hành thao tác sửa lỗi đọc ***(read repair)*** là tiến trình ngầm để cập nhật trạng thái mới nhất cho tất cả các máy chủ lưu trữ nhân bản của dữ liệu. Cassandra tổ chức các nút máy chủ thành cụm theo định dạng vòng tròn và dữ liệu được phân tán theo vòng tròn này theo bảng hàm băm nhất quán ***(Distributed consistent hashing)***. Nếu mỗi dữ liệu của Cassandra được sao lưu trên **N** nút, khi một khóa **k** được quyết định sẽ lưu vào một nút nào đó, nút đó sẽ được coi là nút điều phối. Nút điều phối có nhiệm vụ phân phối bản ghi đấy cho **N-1** nút còn lại theo nguyên tắc: từ nút điều phối, đi theo chiều kim đồng hồ, dữ liệu sẽ được ghi lên 2 nút tiếp theo được gặp. Hình trên mô tả khi khóa **k** được xác định là sẽ ghi vào nút **B**, nút **B** sẽ đóng vai trò điều phối, luân chuyển khóa đấy cho 2 nút tiếp theo là nút **C** và nút **D**. như vậy, nút **D** sẽ lưu trữ các khóa nằm trong vùng **[A; D]**. Danh sách các khóa trong vùng này được gọi là danh sách liên kết của nút D. Việc đưa các giá trị của khóa k sang các nút khác áp dụng cho tất cả các tác vụ ghi, cập nhật hay xóa. Vì việc việc quyết định số lượng nút được luân chuyển ngay lập tức mỗi khi có tác vụ ghi diễn ra ảnh hưởng trực tiếp đến mức độ nhất quán của hệ thống.

Trong cấu hình của Cassandra Apache ta có một chỉ số "***replicatioon\_factor***" và "***w***". Chỉ số "***replication\_factor***" sẽ được cài đặt ngay khi khởi tạo một ***key\_space***, đó là số lượng nút trong vòng sẽ được dùng để sao lưu dữ liệu. Chỉ số "***w***" khi cấu hình Cassandra là số lượng nút trả về kết quả khi thực hiện tác vụ ghi bắt buộc để tác vụ đấy được coi là thành công.

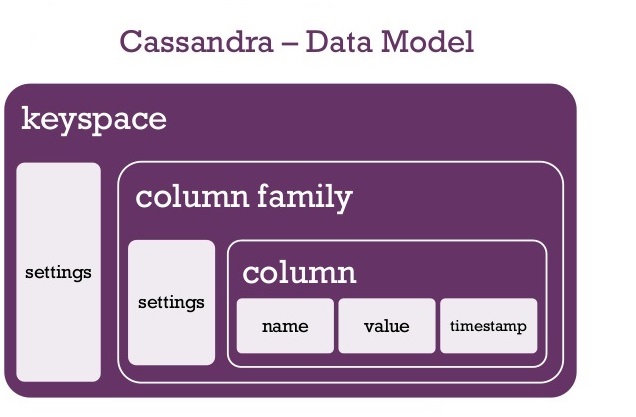
**Giao tiếp giữa các nút trong Cassandra**

****

Mỗi khi cụm Cassandra bổ sung hoặc loại bỏ một nút ra khỏi cụm, dữ liệu trong cụm sẽ phải được phân bố lại. Khi bổ sung một nút, nút đó sẽ lấy đi 1 phần dữ liệu của các nút khác khi nó được cấp cho 256 nút ảo. Khi một nút bị loải khỏi cụm, dữ liệu của nút đó sẽ phải được rải đều cho các nút khác. Trong Cassandra, các nút giao tiếp với nhau thông qua giao thức ***Gossip***. Gossip là một giao thức dùng để cập nhật thông tin về trạng thái của các node khác đang tham gia vào cluster. Đây là một giao thức liên lạc dạng peer-to-peer trong đó mỗi node trao đổi định kỳ thông tin trạng thái của chúng với các node khác mà chúng có liên kết. Tiến trình gossip chạy mỗi giây và trao đổi thông tin với nhiều nhất là ba node khác trong cluster. Các node trao đổi thông tin về chính chúng và cả thông tin với các node mà chúng đã trao đổi, bằng cách này toàn bộ những node có thể nhanh chóng hiểu được trạng thái của tất cả các node còn lại trong cluster. Một gói tin gossip bao gồm cả version đi kèm với nó, như thế trong mỗi lần trao đổi gossip, các thông tin cũ sẽ bị ghi đè bởi thông tin mới nhất ở một số node. Khi một node được khởi động, nó sẽ xem file cấu hình *cassandra.yaml* để xác định tên cluster chứa nó và các nút khác trong cluster được cấu hình trong file, được biết với tên là seed node. Để ngăn chặn sự đứt đoạn trong truyền thông gossip, tất cả các nút trong cluster phải có cùng 1 danh sách các seed node được liệt kê trong file cấu hình. Bởi vì, phần lớn các xung đột được sinh ra khi 1 node được khởi động. Mặc định, 1 node sẽ phải nhớ những node mà nó đã từng gossip kể cả khi khởi động lại và seed node sẽ không có mục đích nào khác ngoài việc cập nhật 1 node mới khi nó tham gia vào cluster. Tức là, khi một node tham gia vào cluster, nó sẽ liên lạc với các seed node để cập nhật trạng thái của tất cả các node khác trong cluster. Trong những cluster có nhiều data center, danh sách seed node nên chứa ít nhất một seed node trên mỗi data center, nếu không thì khi có 1 nút mới tham gia vào cluster, thì nó sẽ liên lạc với một seed node nằm trên data center khác. Cũng không nên để mọi node đều là seed node vì nó sẽ làm giảm hiệu năng của gossip và gây khó duy trì. Việc tối ưu gossip là không quan trọng như khuyến khích nên sử dụng một danh sách nhỏ các seed node, thông thường 3 seed node trên một data center.

**Mô hình dữ liệu của Cassandra**

Mô hình dữ liệu của Cassandra tuân theo cách tiếp cận hệ thống cột, mà có thể dễ dàng được hiểu tương tự như một cấu trúc bảng quan hệ nhưng theo cách NoSQL. Mô tả dưới đây sẽ làm rõ ràng hơn:



*Hình: mô hình dữ liệu trên cassandra*

**Keyspace:** Một khóa có thể được xem như là kho chứa ngoài cùng nhất cho dữ liệu trong Cassandra. Tất cả dữ liệu trong Cassandra sẽ đặt trong một keyspace. Nó có thể được xem như là một cơ sở dữ liệu trong RDBMS, nó là một tập hợp các bảng. Trong trường hợp của Cassandra, một keyspace là một tập hợp hệ thống cột.

**Column family (Hệ thống Cột):** Một hệ thống cột có thể được xem như là một bộ sưu tập các hàng, và mỗi hàng có một tập hợp các cột. Nó tương tự như một bảng trong RDBMS nhưng có một số khác biệt. Hệ thống cột được xác định, nhưng không cần thiết mỗi hàng phải có tất cả các cột và các cột có thể được thêm vào hoặc xoá bỏ khỏi một hàng khi cần thiết.

**Column (Cột):** Cột là đơn vị dữ liệu cơ bản trong Cassandra. Nó có 3 giá trị*: khoá hoặc tên cột, giá trị cột, và một mốc thời gian.*

**Quy trình khoa học dữ liệu**

Quy trình tiêu chuẩn liên ngành cho khai thác dữ liệu, thường được biết đến với từ viết tắt CRISP-DM (Cross-industry standard process for Data Mining) là một quy trình lặp, có khả năng quay lui (backtracking) gồm 6 giai đoạn:

* Tìm hiểu nghiệp vụ (Business understanding)
  + Hiểu mục tiêu nghiệp vụ
  + Đánh giá tình huống
  + Quy đổi từ mục tiêu nghiệp vụ sang mục tiêu khai phá dữ liệu
  + Xây dựng kế hoạch dự án
* Tìm hiểu dữ liệu (Data understanding)
  + Xem xét các yêu cầu về dữ liệu
  + Thu thập, thăm dò và đánh giá chất lượng ban đầu
* Chuẩn bị dữ liệu (Data preparation)
  + Lựa chọn dữ liệu
  + Thu thập dữ liệu
  + Tích hợp và định dạng dữ liệu (ETL)
  + Làm sạch dữ liệu
  + Chuyển đổi dữ liệu
* Mô hình hoá (Modeling)
  + Lựa chọn kỹ thuật mô hình thích hợp
  + Tách tập dữ liệu thành tập huấn và thử nghiệm tập hợp con cho mục đích đánh giá
  + Phát triển và kiểm tra các thuật toán mô hình thay thế và cài đặt tham số
  + Tinh chỉnh cài đặt mô hình theo đánh giá ban đầu về hiệu suất của mô hình
* Đánh giá mô hình (Evaluation Model)
  + Đánh giá mô hình trong bối cảnh ứng với các tiêu chí nghiệp vụ
  + Phê duyệt mẫu
* Triển khai (Deployment)
  + Tạo báo cáo kết quả
  + Lập kế hoạch và phát triển quy trình triển khai
  + Triển khai mô hình
  + Phân phối kết quả mô hình và tích hợp trong hệ thống của tổ chức
  + Lập kế hoạch bảo trì / cập nhật
  + Đánh giá dự án



*Hình 3: Mô hình CRISP-DM*

**Các loại phân tích dữ liệu:**

1. **Thống kê mô tả (Descriptive Statistics):**

Là các phương pháp liên quan đến việc thu thập số liệu, tóm tắt, trình bày, tính toán và mô tả các đặc trưng khác nhau để phản ánh một cách tổng quát đối tượng nghiên cứu.

* Các nguyên tắc định lượng mô tả các features chính của một tập dữ liệu. Về bản chất, nó mô tả một bộ dữ liệu.
* Quy trình mô tả và diễn dịch là các bước khác nhau
* Đơn biến và 2 biến là 2 loại phân tích thống kê mô tả

1. **Thống kê suy luận (Inferential Statistics):**

Là bao gồm các phương pháp ước lượng các đặc trưng của tổng thể, phân tích mối liên hệ giữa các hiện tượng nghiên cứu, dự đoán hoặc ra quyết định trên cơ sở thu thập thông tin từ kết quả quan sát mẫu. Nghĩa là, thống kê từ một lượng sample để dự đoán về population.

* Suy luận thường là mục tiêu của các mô hình thống kê (Statistical models)
* Suy luận phụ thuộc rất nhiều vào population và sample

1. **Phân tích dữ liệu thăm dò (Exploratory Data Analysis - EDA)**

Là một cách tiếp cận để phân tích các tập dữ liệu để tóm tắt các đặc điểm chính của chúng, tìm các mối quan hệ chưa được biết trước đó, thường với các phương pháp visual. EDA khác với phân tích dữ liệu ban đầu (Initial Data Analysis - IDA) tập trung hẹp hơn trong việc kiểm tra các giả định cần thiết cho việc kiểm tra mô hình và giả thuyết, và xử lý các giá trị thiếu và thực hiện các biến đổi khi cần. EDA bao gồm IDA.

* + Khám phá các model tốt cho việc khám phá các kết nối mới.
  + Có ích cho việc định nghĩa các nghiên cứu và câu hỏi trong tương lai.
  + Không nên sử dụng phân tích thăm dò để khái quát hoặc dự đoán.

1. **Phân tích dự đoán (Predictive Analysis):**

Phân tích dự đoán là nhánh của phân tích nâng cao được sử dụng để đưa ra dự đoán về các sự kiện không xác định trong tương lai. Phân tích dự báo sử dụng nhiều kỹ thuật từ khai phá dữ liệu, thống kê, mô hình hóa, học máy và trí thông minh nhân tạo để phân tích dữ liệu hiện tại để đưa ra dự đoán về tương lai. Nó sử dụng một số khai thác dữ liệu, mô hình tiên đoán và các kỹ thuật phân tích để tập hợp quản lý, công nghệ thông tin và quy trình nghiệp vụ mô hình hóa để đưa ra dự đoán về tương lai. Các mẫu tìm thấy trong dữ liệu lịch sử và giao dịch có thể được sử dụng để xác định các rủi ro và cơ hội cho tương lai. Các mô hình phân tích dự đoán nắm bắt các mối quan hệ giữa nhiều yếu tố để đánh giá rủi ro với một tập hợp các điều kiện cụ thể để gán điểm hoặc trọng số. Bằng cách áp dụng thành công các phân tích dự báo, các doanh nghiệp có thể giải thích một cách hiệu quả các dữ liệu lớn vì lợi ích của họ.

* Việc độ chính xác của dự đoán thì phụ thuộc rất nhiều vào việc đo lường đúng các biến.

1. **Phân tích nguyên nhân (Causal analysis)**

Tìm hiểu chuyện gì sẽ xảy ra đối với một biến khi thay đổi một biến khác

1. **Phân tích cơ chế (Mechanistic analysis)**

Hiểu được chính xác những thay đổi ở các biến dẫn tới những thay đổi các biến khác

* Khó để suy luận, trừ những tình huống đơn giản.
* Thường mô hình hóa bởi một tập các phương trình xác định.
* Nếu các phương trình được biết nhưng các thông số thì không, chúng có thể được suy ra cùng với phân tích dữ liệu.

**Feature Engineering**

1. **Giới thiệu:**

Khi làm việc với các bài toán Machine Learning thực tế, nhìn chung chúng ta chỉ có được dữ liệu thô (raw) chưa qua chỉnh sửa, chọn lọc. Chúng ta cần phải tìm một phép biến đổi để loại ra những dữ liệu nhiễu (noise), và để đưa dữ liệu thô với số chiều khác nhau về cùng một chuẩn (cùng là các vector hoặc ma trận). Dữ liệu chuẩn mới này phải đảm bảo giữ được những thông tin đặc trưng (features) cho dữ liệu thô ban đầu. Không những thế, tùy vào từng bài toán, ta cần thiết kế những phép biến đổi để có những features phù hợp. Quá trình quan trọng này được gọi là Feature Extraction, hoặc Feature Engineering, tiếng Việt gọi là trích chọn đặc trưng.

Trích một câu nói của thầy Andrew Ng (Giáo sư tại Standford) - Nguồn Feature Engineering - Wiki:

*Coming up with features is difficult, time-consuming, requires expert knowledge. “Applied machine learning” is basically feature engineering.*

*Tạm dịch:*

*Làm việc với features là một việc không dễ dàng, tốn nhiều thời gian, cần rất nhiều kiến thức, áp dụng máy học cơ bản chỉ là feature engineering.*

1. **Mô hình chung cho các bài toán Machine Learning**

Phần lớn các bài toán Machine Learning có thể được thể hiện trong hình vẽ dưới đây:



*Hình 1: Mô hình chung cho các bài toán Machine Learning.*

**TRAINING PHASE** Có hai khối có nền màu xanh lục chúng ta cần phải thiết kế:

1. **Feature Extractor**

* **Đầu vào**

Mục đích của Feature Engineering là tạo ra một Feature Extractor biến dữ liệu thô ban đầu thành dữ liệu phù hợp với từng mục đích khác nhau.

* **Đầu ra**
  + **Raw training input**:

Raw input là tất cả các thông tin ta biết về dữ liệu. Ví dụ: với ảnh thì là giá trị của từng pixel; với văn bản thì là từng từ, từng câu; với file âm thanh thì nó là một đoạn tín hiệu; với cơ sở dữ liệu Iris thì nó là độ dài các cánh hoa và đài hoa,…Dữ liệu thô này thường không ở dạng vector, không có số chiều như nhau. Thậm chí có thể có số chiều như nhau nhưng số chiều quá lớn, như một bức ảnh màu 1000 pixel **x** 1000 pixel thì số elements đã là **3x**(3 vì ảnh màu thường có 3 channels: Red, Green, Blue). Đây là một con số quá lớn, không lợi cho lưu trữ và tính toán.

* + **Output của training set**

Trong các bài toán Unsupervised learning, ta không biết output nên hiển nhiên sẽ không có đầu vào này. Trong các bài toán Supervised learning, có khi dữ liệu này cũng không được sử dụng. Ví dụ: nếu raw input đã có cùng số chiều rồi nhưng số chiều quá lớn, ta muốn giảm số chiều của nó thì cách đơn giản nhất là chiếu vector đó xuống một không gian có số chiều nhỏ hơn bằng cách lấy một ma trận ngẫu nhiên nhân với nó. Ma trận này thường là ma trận béo (số hàng ít hơn số cột, tiếng Anh - fat matrices) để đảm bảo số chiều thu được nhỏ hơn số chiều ban đầu. Việc làm này mặc dù làm mất đi thông tin, trong nhiều trường hợp vẫn mang lại hiệu quả vì đã giảm được lượng tính toán ở phần sau. Đôi khi ma trận chiếu không phải là ngẫu nhiên mà có thể được học dựa trên toàn bộ raw input, ta sẽ có bài toán tìm ma trận chiếu để lượng thông tin mất đi là ít nhất. Trong nhiều trường hợp, dữ liệu output của training set cũng được sử dụng để tạo ra Feature Extractor. Ví dụ: trong bài toán classification, ta không quan tâm nhiều đến việc mất thông tin hay không, ta chỉ quan tâm đến việc những thông tin còn lại có đặc trưng cho từng class hay không. Ví dụ, dữ liệu thô là các hình vuông và hình tam giác có màu đỏ và xanh. Trong bài toán phân loại đa giác, các output là tam giác và vuông, thì ta không quan tâm tới màu sắc mà chỉ quan tâm tới số cạnh của đa giác. Ngược lại, trong bài toán phân loại màu, các class là xanh và đỏ, ta không quan tâm tới số cạnh mà chỉ quan tâm đến màu sắc thôi.

* + **Prior knowledge about data (kiến thức về data)**

Đôi khi những giả thiết khác về dữ liệu cũng mang lại lợi ích. Ví dụ, trong bài toán classification, nếu ta biết dữ liệu là (gần như) linearly separable thì ta sẽ đi tìm một ma trận chiếu sao cho ở trong không gian mới, dữ liệu vẫn đảm bảo tính linearly separable, việc này thuận tiện hơn cho phần classification vì các thuật toán linear, nhìn chung, đơn giản hơn.

Sau khi học được feature extractor thì ta cũng sẽ thu được extracted features cho raw input data. Những extracted features này được dùng để huấn luyện các thuật toán Classification, Clustering, Regression… ở phía sau.

1. **Main Algorithms (các thuật toán chính)**

Khi có được extracted features rồi, chúng ta sử dụng những thông tin này cùng với training output và prior knowledge để tạo ra các mô hình phù hợp, điều mà chúng ta đã làm ở những bài trước.

**Chú ý**: Trong một số thuật toán cao cấp hơn, việc huấn luyện feature extractor và main algorithm được thực hiện cùng lúc với nhau chứ không phải từng bước như trên.

**Một điểm rất quan trọng**: khi xây dựng bộ feature extractor và main algorithms, chúng ta không được sử dụng bất kỳ thông tin nào trong tập test data. Ta phải giả sử rằng những thông tin trong test data chưa được nhìn thấy bao giờ.

**TESTING PHASE**

Bước này đơn giản hơn nhiều. Với raw input mới, ta sử dụng feature extractor đã tạo được ở trên (tất nhiên không được sử dụng output của nó vì output là cái ta đang đi tìm) để tạo ra feature vector tương ứng. Feature vector được đưa vào main algorithm đã được học ở training phase để dự đoán output.

1. **Một số ví dụ về Feature Engineering**

* **Feature Selection**

Giả sử rằng các điểm dữ liệu có số features khác nhau (do kích thước dữ liệu khác nhau hay do một số feature mà điểm dữ liệu này có nhưng điểm dữ liệu kia lại không thu thập được), và số lượng features là cực lớn. Chúng ta cần chọn ra một số lượng nhỏ hơn các feature phù hợp với bài toán.

**Một số cách trong Feature Selection như:**

**Cách 1: Loại bỏ các features có giá trị phương sai thấp:**

Ngưỡng phương sai (*VarianceThreshold*) là một phương pháp cơ bản của Feature Selection. Nó loại bỏ tất cả các features có phương sai không đáp ứng giá trị ngưỡng. Theo mặc định, nó loại bỏ tất cả các tính năng zero-variance (phương sai bằng 0), tức là các features có cùng giá trị trong tất cả các sample.

Trong đó: **p** là số phần trăm mà bạn muốn loại bỏ của một sample, như ví dụ dưới đây, ta chọn 80% = 0.8

**from** sklearn.feature\_selection **import** VarianceThreshold  
**import** numpy **as** np  
X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]  
sel = VarianceThreshold(threshold=(.8 \* (1 - .8)))  
print(**"Ma trận ban đầu:\n{}\n\nMa trận sau khi transform:\n{}\n"**.format(X, sel.fit\_transform(X)))**def** column(matrix, i):  
 **return** [row[i] **for** row **in** matrix]  
print(**"Giá trị của varianceThreshold là: {}"**.format(sel))  
print(**"Giá trị variance của cột 0 là: {}\nGiá trị variance của cột 1 là: {}"**.format(np.var(column(X,0)),np.var(column(X,1))))

*Kết Quả:*

Ma trận ban đầu:

[[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]

Ma trận sau khi transform:

[[0 1] [1 0] [0 0] [1 1] [1 0] [1 1]]

#Ta có thể thấy VarianceThreshold đã xóa cột đầu tiên của ma trận vì variance của cột 0 < giá trị VarianceThreshold

#để kiểm chứng, ta sẽ thử xem giá trị của cột 0 và cột 1 của ma trận

Giá trị của varianceThreshold là: VarianceThreshold(threshold=0.15999999999999998)

Giá trị variance của cột 0 là: 0.13888888888888892

Giá trị variance của cột 1 là: 0.22222222222222224

* **Dimentionality selection**
* **Bag-of-word**

**Feature Scaling and Normalization:**

Các điểm dữ liệu đôi khi được đo đạc với những đơn vị khác nhau, m và feet chẳng hạn. Hoặc có hai thành phần (của vector dữ liệu) chênh lệch nhau quá lớn, một thành phần có khoảng giá trị từ 0 đến 1000, thành phần kia chỉ có khoảng giá trị từ 0 đến 1 chẳng hạn. Lúc này, chúng ta cần chuẩn hóa dữ liệu trước khi thực hiện các bước tiếp theo.

**Rescaling**

Phương pháp đơn giản nhất là đưa tất cả các thành phần về cùng một khoảng, [0,1] hoặc [-1,1]. Nếu muốn đưa một thành phần (feature) về khoảng [0,1] công thức sẽ là:

Trong đó, **x** là giá trị ban đầu, **x’** là giá trị sau khi chuẩn hóa. **min(x), max(x)** được tính trên toàn bộ dữ liệu training data ở cùng một thành phần. Việc này được thực hiện trên từng thành phần của vector dữ liệu **x**.

Ở Python, thư viện sklearn cho phép thực hiện scale khoảng [0, 1] đơn giản với hàm: min\_max\_scaler()

*#thêm thư viện xử lý ma trận***import** numpy **as** np  
*#thư viện sklearn để lấy hàm min\_max\_scaler***from** sklearn **import** preprocessing  
*#Tạo một ma trận tên X\_Train*X\_train = np.array([[1.,-1.,2.],[2.,0.,0.],[0.,1.,-1.]])  
*#gán biến min\_max\_scaler từ hàm MinMaxScaler() cho khoảng [0,1]  
#gán biến max\_abs\_scaler từ hàm MaxAbsScaler() cho khoảng [-1,1]*min\_max\_scaler = preprocessing.MinMaxScaler()  
max\_abs\_scaler = preprocessing.MinMaxScaler()  
*#Thực hiện chuyển đổi về khoảng [0,1] và [-1,1] bằng hàm fit\_transform*X\_train\_minmax = min\_max\_scaler.fit\_transform(X\_train)  
X\_train\_maxabs = max\_abs\_scaler.fit\_transform(X\_train)  
*#in ra màn hình ma trận sau khi scaled!*print(**"Ma trận ban đầu:\n{}\n\nMa trận sau khi scaling:\nĐối với khoảng [0,1]:\n{}\n\n""Đối với khoảng [-1,1]:\n{}"** .format(X\_train,X\_train\_minmax,X\_train\_maxabs))

*Kết Quả:*

Ma trận ban đầu:

[[ 1. -1. 2.]

[ 2. 0. 0.]

[ 0. 1. -1.]]

Ma trận sau khi scale:

Đối với khoảng [0,1]:

[[ 0.5 0. 1. ]

[ 1. 0.5 0.33333333]

[ 0. 1. 0. ]]

Đối với khoảng [-1,1]:

[[ 0.5 -1. 1. ]

[ 1. 0. 0. ]

[ 0. 1. -0.5]]

**Standardization**

Một phương pháp nữa cũng hay được sử dụng là giả sử mỗi thành phần đều có phân phối chuẩn với kỳ vọng là 0 và phương sai là 1. Khi đó, công thức chuẩn hóa sẽ là:

Trong đó: **,**  lần lượt là kỳ vọng và phương sai (standard deviation) của thành phần đó trên toàn bộ training data.

*#thêm thư viện xử lý ma trận***import** numpy **as** np  
*#thư viện sklearn để lấy hàm min\_max\_scaler***from** sklearn **import** preprocessing  
*#Tạo một ma trận tên X\_Train*X\_train = np.array([[1.,-1.,2.],[2.,0.,0.],[0.,1.,-1.]])  
*#gán biến scaler từ hàm StandardScaler() sau khi fit ma trận*scaler = preprocessing.StandardScaler().fit(X\_train)  
print(**"Ma trận ban đầu:\n{}\n\nMa trận sau khi scaling:\n{}"** .format(X\_train,scaler.transform(X\_train)))

Kết Quả:

Ma trận ban đầu:

[[ 1. -1. 2.]

[ 2. 0. 0.]

[ 0. 1. -1.]]

Ma trận sau khi scaling:

[[ 0. -1.22474487 1.33630621]

[ 1.22474487 0. -0.26726124]

[-1.22474487 1.22474487 -1.06904497]]

**Scaling to unit length**

Một lựa chọn khác nữa cũng được sử dụng rộng rãi là chuẩn hóa các thành phần của mỗi vector dữ liệu sao cho toàn bộ vector có độ lớn (**Euclid**, tức norm 2) bằng 1. Việc này có thể được thực hiện bằng: