Cassandra

Apache Cassandra là cơ sở dữ liệu dạng cột (column family) được phát triển bởi Facebook trong năm 2008, sau khi trở thành sản phẩm của Facebook trong một thời gian ngắn, Cassandra trở thành dự án open source tại Google code vào tháng 7 năm 2008. Tháng 3 năm 2009, Cassandra trở thành dự án tiềm năng của tổ chức Apache và không lâu sau, vào tháng 2 năm 2010 Cassandra đã trở thành một trong những project nổi bật nhất của Apache.

Cassandra Basics

Giới thiệu Apache Cassandra

Apache Cassandra là cơ sở dữ liệu dạng cột (column family) được phát triển bởi Facebook trong năm 2008, sau khi trở thành sản phẩm của Facebook trong một thời gian ngắn, Cassandra trở thành dự án open source tại Google code vào tháng 7 năm 2008. Tháng 3 năm 2009, Cassandra trở thành dự án tiềm năng của tổ chức Apache và không lâu sau, vào tháng 2 năm 2010 Cassandra đã trở thành một trong những project nổi bật nhất của Apache. (Theo wikipedia.org)

1. Là cơ sở dữ liệu hướng cột (column-family)

Cơ sở dữ liệu dạng cột (column-family) lưu trữ dữ liệu theo nhiều cột trong mỗi dòng (row key). Có thể coi column-family như là một table trong cơ sở dữ liệu quan hệ.

2. Khóa chính (Primary key)

Được thiết kế như partition key. Nếu primary key bao gồm nhiều column thì:

- Column đầu tiên sẽ là partition key: Chịu trách nhiệm phân tán dữ liệu tới các node trong hệ thống
- Các column phía sau là clustering key: Sắp xếp thứ tự dữ liệu trong mỗi partition key.

- Ưu điểm: So với cơ sở dữ liệu RDMS truyền thống (như MySQL, Oracle) dạng dòng (row), khi truy xuất dữ liệu cần thực hiện lấy nhiều row cần scan theo index key trên vùng rộng lớn (full-scan), trong khi đó, Cassandra tổ dữ dữ liệu hướng cột (column), khi đó cần truy cập dữ liệu theo partition key, chỉ cần đọc block dữ liệu duy nhất, do đó sẽ giảm thời gian và tải hệ thống khi truy vấn dữ liệu.

3. Lưu trữ và xử lý phân tán

Trong Cassandra, dữ liệu được lưu trữ phân tán trên các máy chủ của cụm để đảm bảo tính sẵn sàng cao, đồng thời tối ưu việc xử lý truy vấn dữ liệu trên các máy chủ mà dữ liệu được lưu trữ.

4. Đảm bảo an toàn

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.

5. Hỗ trợ đa dạng mô hình

Hỗ trợ lưu trữ các dạng dữ liệu:

- Key-value
- Cơ sở dữ liệu cấu trúc: Structure
- Cơ sở dữ liệu đồ thị: Graph

6. Khả năng chịu lỗi cao

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. 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.

7. Tính khả dụng, mở rộng cao

- Cassandra cho phép mở rộng cơ sở dữ liệu bằng cách bổ sung thêm node vào cluster một cách dễ dàng.
- Hỗ trợ nhiều ngôn ngữ lập trình ở client side: C++, Java, Python, PHP, Ruby, Erlang, C#...

Kiến trúc Cassandra

Thiết kế của Cassandra là thiết kế phân tán dựa trên kiến trúc mạng ngang hàng (Peer - to - Peer) tất cả các node máy chủ trong hệ thống đều có vai trò như nhau và không có node máy chủ nào đóng vai trò là máy chủ trung tâm (master), giảm thiểu sự cố của máy chủ này có thể kéo theo đánh sập hoàn toàn hệ thống như các kiến trúc master-slave truyền thống.

Các node máy chủ của Cassandra là độc lập và tham gia vào kết nối với các node máy chủ khác trong hệ thống. Mỗi node 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 node trong hệ thống bị sự cố và dừng hoạt động, các thao tác đọc ghi dữ liệu có thể được xử lý bởi các node khác trong hệ thống. Quá trình này hoàn toàn trong suốt với ứng dụng cho phép ẩn đi sự cố của máy chủ hệ thống đối với các ứng dụng.

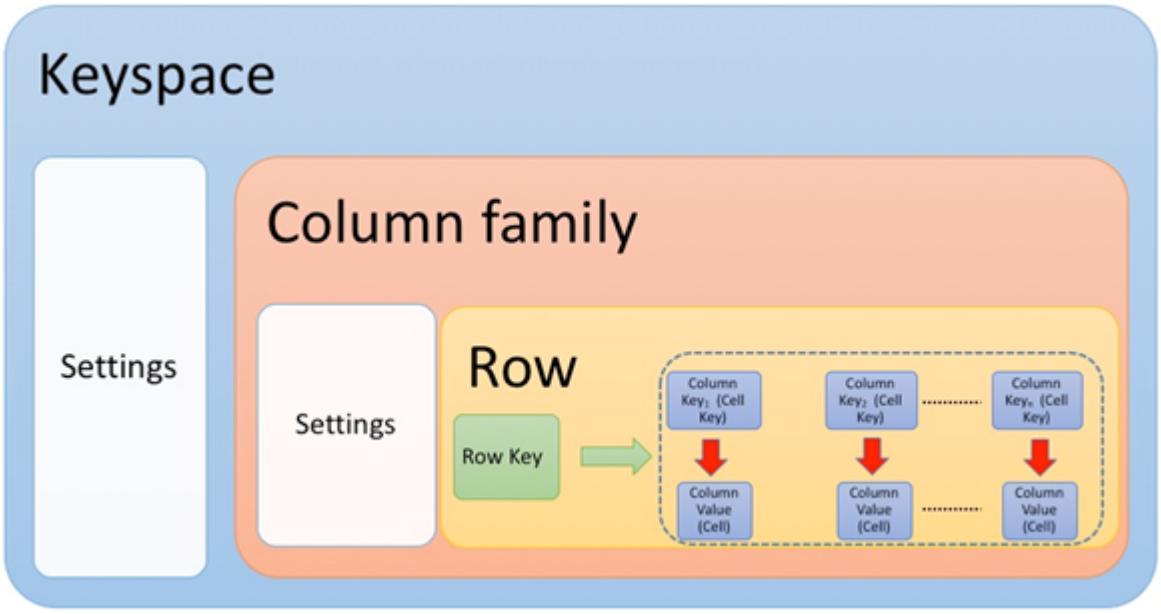
Trong Cassandra, mỗi đối tượng dữ liệu có thể được nhân bản và lưu giữ trên nhiều máy chủ. Nếu một trong các máy chủ lưu một phiên bản dữ liệu bị lỗi hoặc không phải là phiên bản được cập nhật dữ liệu mới nhất, Cassandra có cơ chế đồng bộ để luôn đảm báo các thao tác đọc sẽ luôn trả về dữ liệu mới nhất. 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.

Mô hình dữ liệu trong Cassandra

Mô hình dữ liệu Cassandra tuân theo quy tắc hệ thống cột (column family):

- Column family: là một đối tượng của NoSQL nơi chứa các cột dữ liệu. Nó là một tập hợp dữ liệu chứa các cặp “khóa – giá trị”. Trong đó “khóa” được ánh xạ đến một giá trị gồm tập hợp các cột. Tương tự với RDBMS, column family là một bảng, mỗi cặp “khóa – giá trị” là một hàng.
- Tất cả các Column family được lưu trữ trong “kho chứa” gọi là Keyspace.
- Column: là một tập hợp dữ liệu (bộ 3) gồm tên cột, giá trị, và mốc thời gian (timestamp).

u trữ cũng



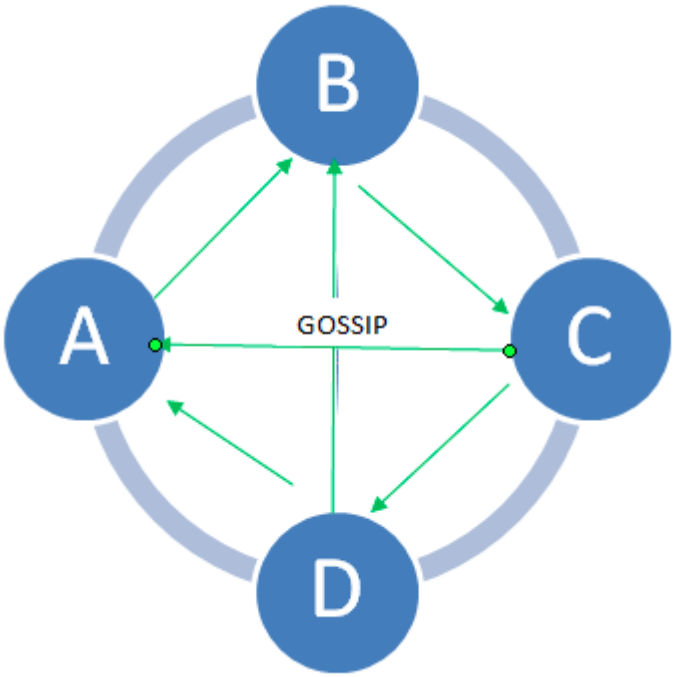
Mô hình lưu trữ dữ liệu trong Column family

Khác với mô hình dữ liệu trong cơ sở dữ liệu quan hệ được thiết kế nhằm đảm bảo cho việc lưu trữ hiệu quả, giảm tối đa việc dư thừa dữ liệu, xây dựng các mối quan hệ và liên kết giữa các bảng hoặc các thực thể với nhau. Mô hình dữ liệu trong Cassandra được thiết kế với mục đích hoàn toàn khác đó là giúp việc thao tác với dữ liệu hiệu quả và có thể lưu trữ được một khối lượng dữ liệu khổng lồ.

Giao thức Gossip trong Cassandra

Mỗi khi cụm Cassandra bổ sung hoặc loại bỏ một node ra khỏi cụm, dữ liệu trong cụm sẽ phải được phân bố lại. Khi bổ sung một node, node đó sẽ lấy đi 1 phần dữ liệu của các node, khi một node bị loại khỏi cụm, dữ liệu của node đó sẽ phải được lưu trữ đều trên các node khác.

Trong Cassandra, các node 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 [illegible] ổi định kỳ thông tin trạng thái của c



Giao thức Gossip

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 node khác trong cluster được cấu hình trong file, được biết với tên là seed node. Để ngăn chặn sự gián đoạn trong truyền thông gossip, tất cả các node trong cluster phải có cùng 1 danh

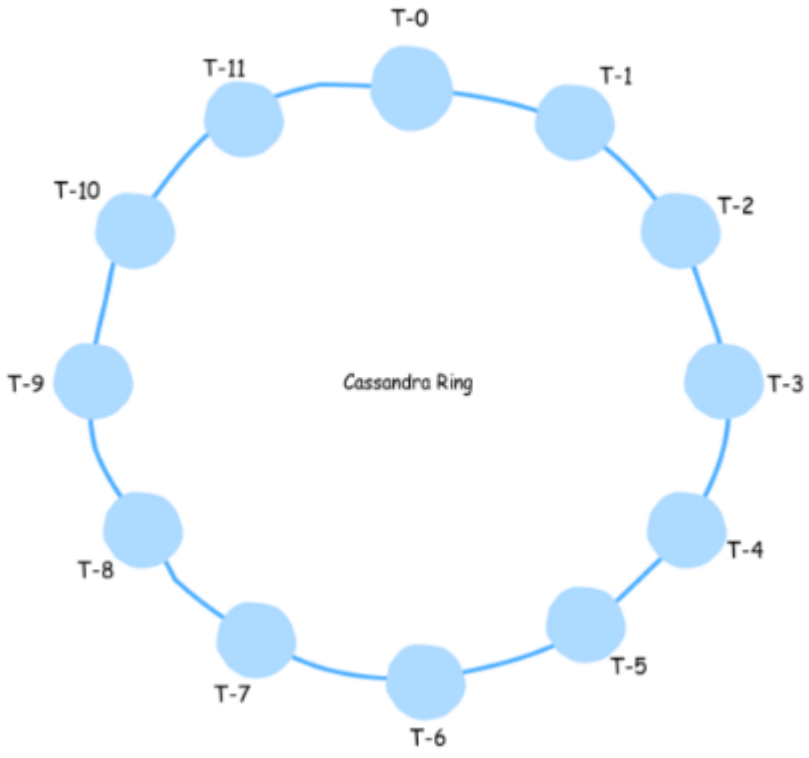
sách các seed node được liệt kê trong file cấu hình. Bởi vì, phần lớn các xung đột được sinh ra khi 1 node được khởi động.

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 node 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.

Phân tán dữ liệu trong Cassandra

Cas hashing) để tổ
chứ n tán theo vòng
trò



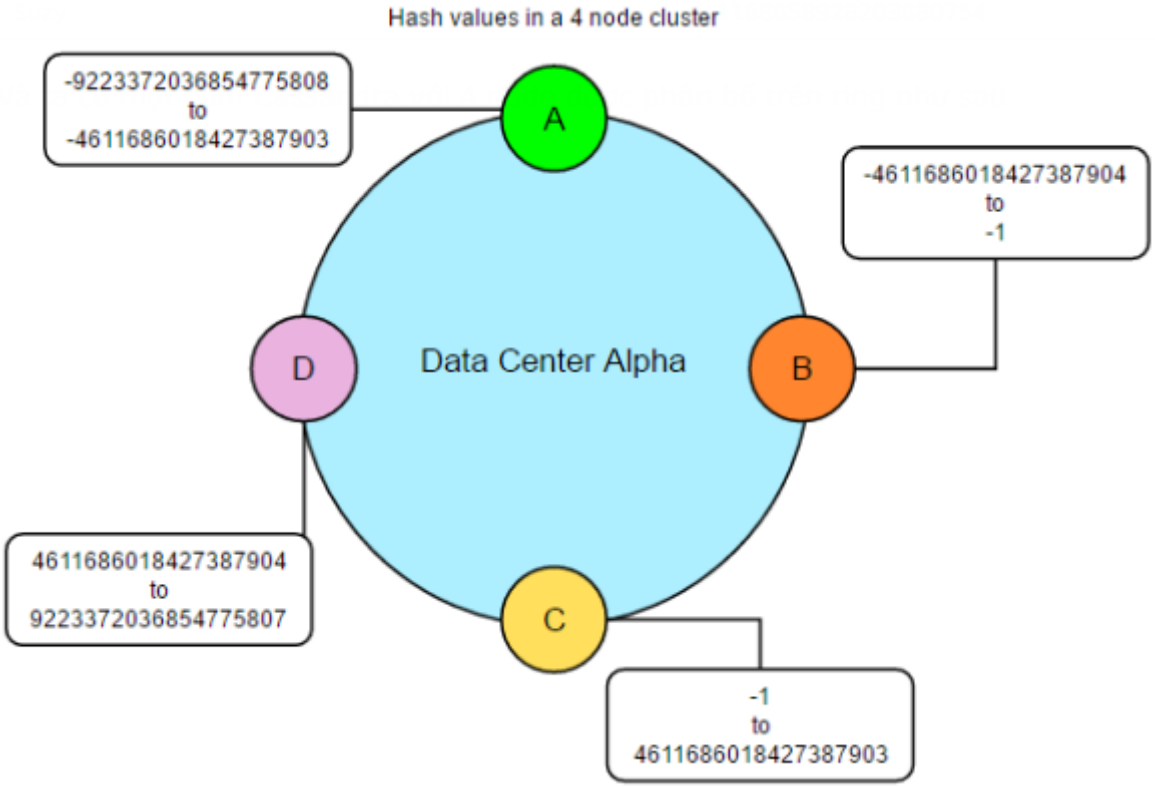
Các node trong Cassandra

Các node trong một cụm Cassandra sẽ được phân bố trên một vòng tròn gọi là ring (Hình trên). Mỗi node sẽ được gán với 1 giá trị key, Cassandra dùng 127 bit để tạo ra key này.

Mỗi node 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 node đó nắm giữ, đi ngược lại chiều kim đồng hồ cho đến khi gặp node đầu tiên thì dừng lại. Đối chiếu lên hình, ta sẽ thấy rằng phạm vi các key mà node 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 node nào thì bản ghi đó sẽ được ghi vào node đấy.

Ví dụ ta có giá trị trên các trường name được băm ra như bảng sau:

Partition Key	Hash value
Jim	-2245452657672322382
Carol	7723358928203680754
Johnny	-6756552657672322382
Suzy	



Với

Cassandra, chúng ta có hai cách để partition dữ liệu (xác định vị trí của từng node 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 hàm băm MD5. Phạm vi khóa nằm trong khoảng từ 0 tới 2^{127}- 1
- Ordered partitioning: Đây là chiến lược đảm bảo các node được sắp xếp theo thứ tự và phạm vi key mà mỗi node sở hữu là như nhau.

Với chiến lược partition thứ nhất, nếu như các giá trị băm xuất ra giúp cho việc đặt các node trong vòng phù hợp thì tất cả các bản ghi sẽ được phân bố đều trên toàn cụm. Việc thêm hay bớt mỗi node ra khỏi cụm cũng dễ dàng hơn do không phải phân bố lại vị trí các node khác.

Với chiến lược partition thứ hai, khi mà các node được phân bố đều vả phạm vi quản lý key là như nhau, nhưng điều đó lại mang lại nhược điểm: Khó cân bằng trong cụm. Mỗi khi thêm hay bớt một node khỏi cụm, người quản trị sẽ phải tự tái cân bằng lại cụm một cách thủ công để đảm bảo các node phân bố đều. Nếu dữ liệu được ghi tuần tự, có thể xảy ra trường hợp hàng loạt dữ liệu được ghi vào một node. Gây mất cân bằng trong cụm.

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

Snitch trong Cassandra

Snitch là protocol sử dụng để mapping IP với Racks và Datacenter, áp dụng các snitches khác nhau thì dữ liệu sẽ được lưu trữ tại các điểm khác nhau trên cluster, snitches giúp ta thiết kế sơ đồ để lưu trữ dữ liệu (sơ đồ mạng máy tính).

Tất cả các nodes trong một cluster thì được áp dụng cùng một snitch, nếu muốn thay đổi snitch cho cluster thì sửa đổi tên snitch cần áp dụng trong file cấu hình và sau đó restart toàn bộ cluster.

Thông tin cấu hình snitches được lưu trong file cassandra.yaml. Các snitches được sử dụng trong Cassandra: SimpleSnitch. Lợi thế của sử dụng SimpleSnitch là không cần hiểu sâu về cách cài đặt, cấu hình Cassandra, SimpleSnitch không đòi hỏi các thông tin về thiết lập data center or rack. Áp dụng loại Snitch này tốt khi triển khai Cassandra trên một máy đơn lẻ, khi thiết lập SimpleSnitch cần thiết lập replication_factor = # đối với strategy_options:

```
CREATE KEYSPACE IF NOT EXISTS demo WITH REPLICATION = { 'class' : 'SimpleStrategy',
'replication_factor' : # };
```

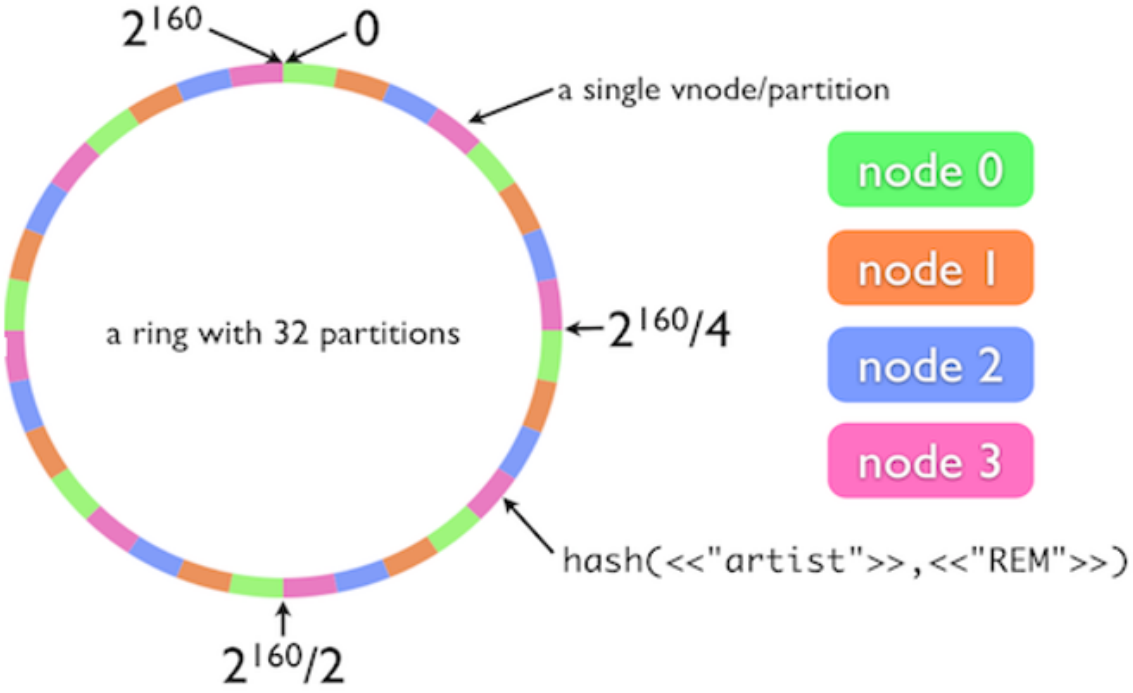
Ngoài ra còn có các kiểu snitches dưới đây:

- Dynamic Snitching
- RackInferringSnitch
- PropertyFileSnitch
- GossipingPropertyFileSnitch
- Ec2Snitch
- Ec2MultiRegionSnitch
- GoogleCloudSnitch
- CloudstackSnitch

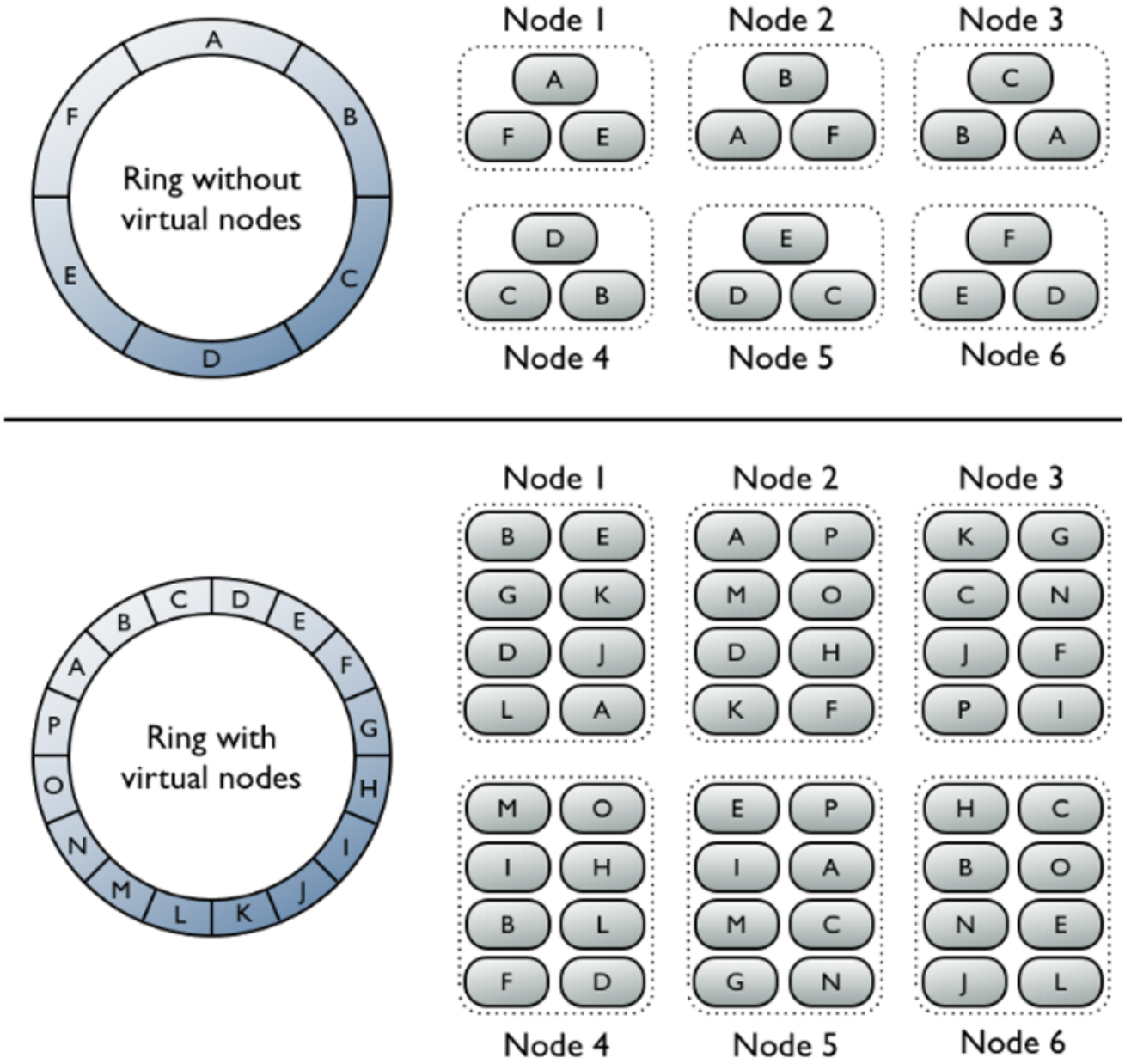
Virtual node - Node ảo trong Cassandra

Để giải quyết vấn đề tái cân bằng lại cụm, Cassandra đề ra một giải pháp đó là sử dụng node ảo. Node ảo giống như một thành phần của vòng tròn trong hệ thống, nhưng bản chất node ảo chỉ là ánh xạ của một node vật lý đến một địa chỉ khác trong vòng. Khi dữ liệu đi vào vùng quản lý của node ảo, nó sẽ được đưa về lưu trữ tại node vật lý của node ảo đó.

Mỗi node vật lý khi tham gia vào vòng sẽ được gán một vị trí của chính node đó và gán thêm một số lượng các vị trí khác (được coi như là node ảo của node đó). Cassandra cấu hình mặc định mỗi một node tham gia vòng sẽ được gán 256 node ảo trong vòng.



Hình trên thể hiện một vòng trong có 4 node vật lý, mỗi node được gán thêm 7 node ảo, như vậy tổng cộng trên vòng tròn sẽ có 32 phân vùng key. Khi việc phân tán đều các node ảo ra khắp vòng, số lượng node 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 node sát nhau hơn đưa hệ thống càng gần đến với việc tất cả dữ liệu sẽ được phân bổ đều khắp các node, xác suất dữ liệu được đưa vào các node là cân bằng nhau khi mà trên một khoảng key nhỏ ta có đầy đủ các node ảo hoặc node vật lý. Trường hợp hoàn hảo nhất là các node vật lý đều có thành phần hiện diện của mình đều khắp trên vòng.



Cân bằng của cụm khi có và không có node ảo

Partitioning trong Cassandra

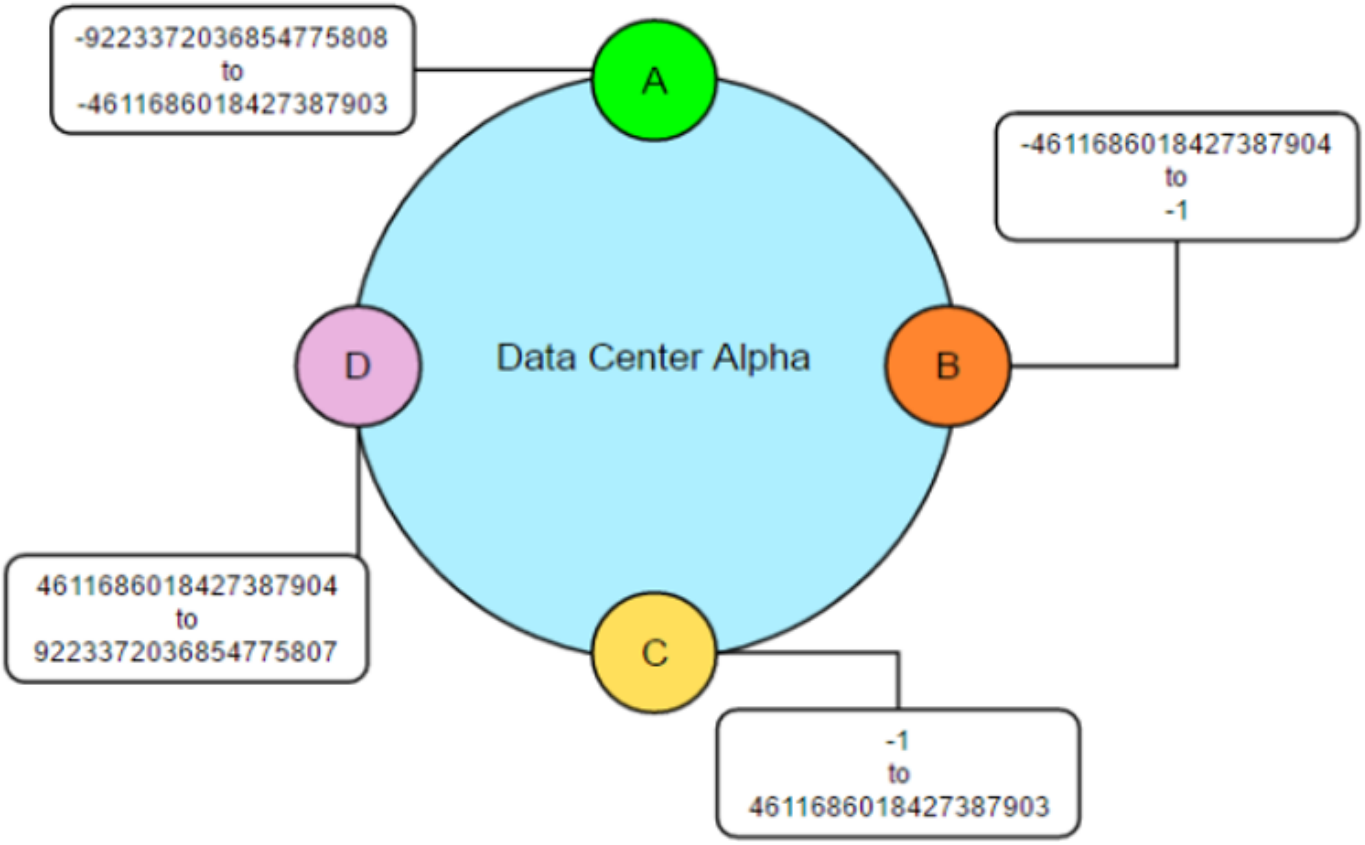
Partitioning là việc bạn quyết định việc dữ liệu được phân tán như thế nào trên các node trong cluster (bao gồm cả các bản sao).

- Trong Cassandra, dữ liệu được quản lý bởi một cluster được đại diện như một không gian dữ liệu hay một ring. Vòng tròn (ring) được chia tương ứng với phạm vi là số lượng các node, mỗi node quản lý một hoặc nhiều vùng của dữ liệu. Trước khi một node có thể tham gia vòng nó được gắn một giá trị token (thẻ bài). Token xác định vị trí của node trên ring và phạm vi dữ liệu mà nó quản lý.
- Application cần chỉ rõ giá trị nằm trong khoảng giữa các token và Cassandra sử dụng nó để điều hướng request tới node chứa dữ liệu đích.
- Cassandra phân vùng dữ liệu trên cluster sử dụng consistent hashing. Trong consistent hashing, phạm vi output trả về bởi hash function được chia ra trên một ring.
- Consistent hashing cho phép việc phân tán dữ liệu trên các cluster mà giảm tối thiểu việc tái cấu trúc lại khi có một hoặc nhiều node được thêm vào hoặc xóa đi trong cluster.
- Consistent hashing partitions data dựa trên partition key.

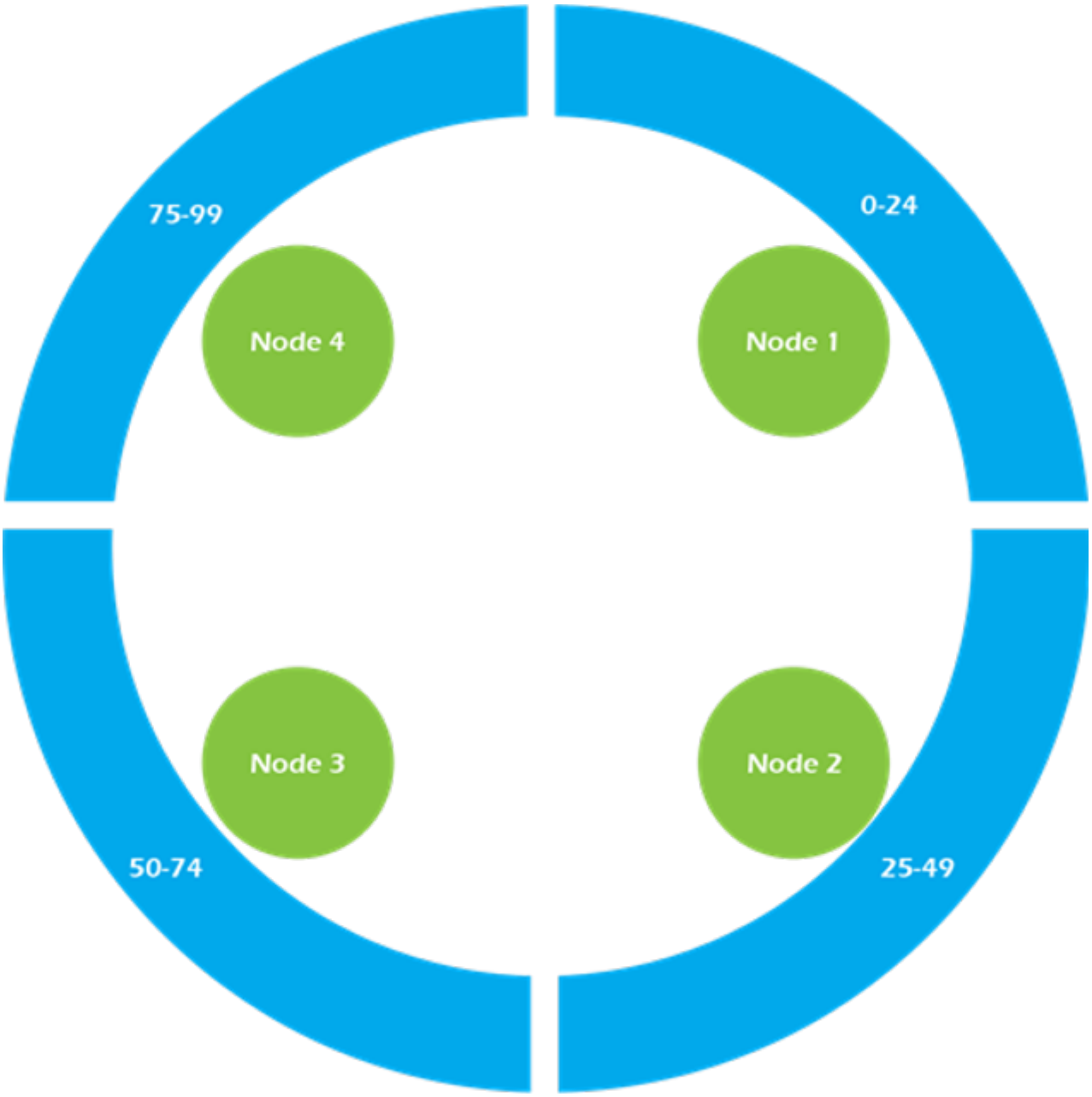
Ví dụ như dữ liệu với trường name được chọn là partition key. Các giá trị trên trường name được băm ra và sắp xếp theo thứ tự như bảng dưới đây:

Partition key	**Hash value**
Jim	-2245462676723222382 2
Carol	7723358927293689754 3
Johnny	-6723372898343w87439
Suzy	11247572943573487523

Sau đó mỗi node trên cluster sẽ quản lý một range dữ liệu dựa trên các giá trị băm được từ partition key.



Partitioning trong Cassandra



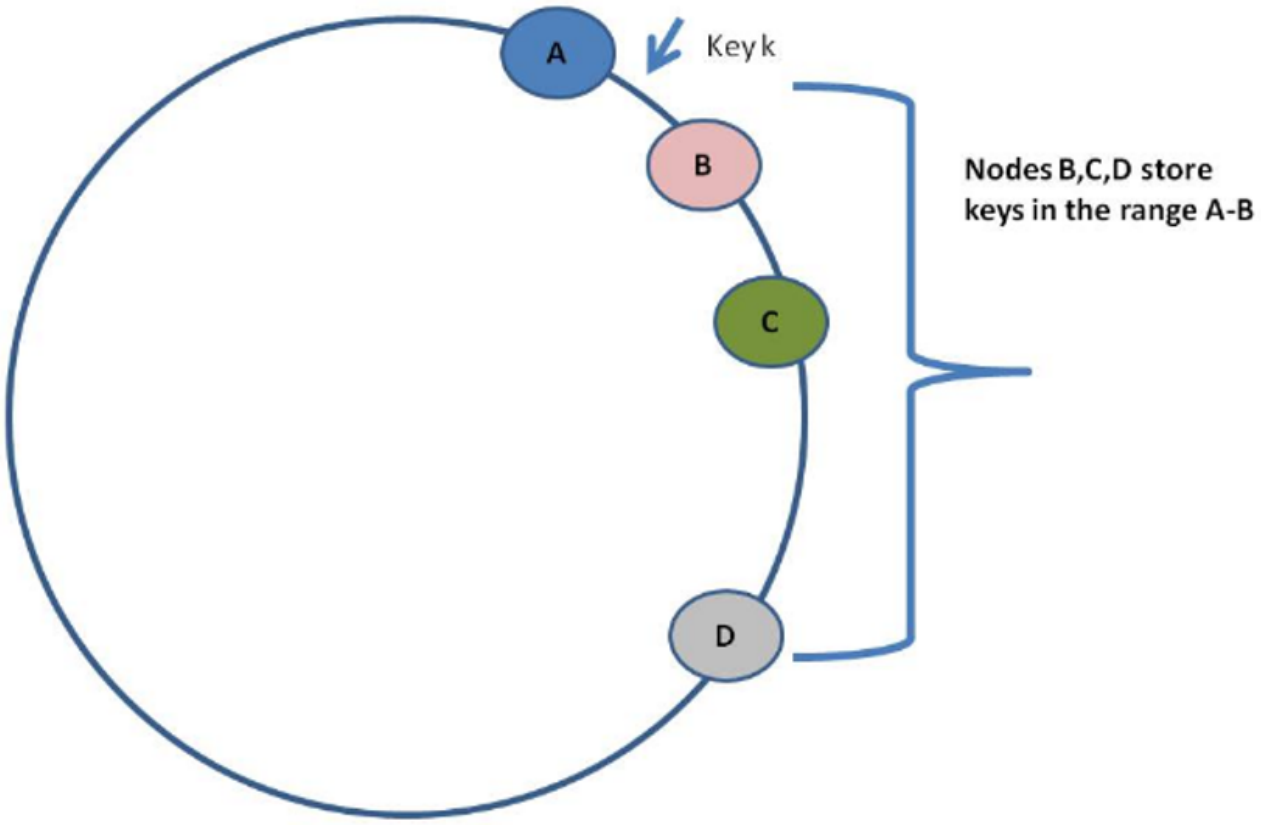
Data range trong Cassandra

Như hình trên chúng ta xem xét một cụm đơn giản gồm 4 node, nơi tất cả các dữ liệu được quản lý bởi 1 cụm được đánh số trong khoảng từ 0 đến 100. Mỗi node được gán một token đại diện cho một điểm trong phạm vi này. Trong ví dụ này, các token có giá trị là 0, 25, 50, và 75. Node đầu tiên, với token 0, chịu trách nhiệm về phạm vi gói (75-0). Node token thấp nhất cũng chấp nhận khóa hàng ít hơn so với các mã token thấp nhất và nhiều hơn với các mã token cao nhất.

Replication dữ liệu trong Cassandra

Để đảm bảo 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 node 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 node, khi một khóa k được quyết định sẽ lưu vào một node nào đó, node đó sẽ được coi là node điều phối. Node điều phối có nhiệm vụ phân phối bản ghi đấy cho N-1 node còn lại theo nguyên tắc: từ node điều phối, đi theo chiều kim đồng hồ, dữ liệu sẽ được ghi lên 2 node tiếp theo được gặp.



Hình trên mô tả khi khóa k được xác định là sẽ ghi vào node B, node B sẽ đóng vai trò điều phối, luân chuyển khóa đấy cho 2 node tiếp theo là node C và node D. Như vậy, node 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 node D. Việc đưa các giá trị của khóa k sang các node khác áp dụng cho tất cả các tác vụ ghi, cập nhật hay xóa.

Vì việc việc quyết định số lượng node 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 node trong vòng sẽ được dùng để sao lưu dữ liệu. Chỉ số "w" khi cấu hình Cassandra là số lượng node trả về kết quả khi thực hiện tác vụ ghi bắt buộc để tác vụ đấy được coi là thành công. Xét trên hình 8, khi ta đặt replication_factor = 3 và w = 2, khi khóa k được ghi vào thì cần phải có ít nhất 2 node trong 3 node B, C, D phản hồi lại ghi thành công thì tác vụ đấy mới được coi là thành công. Việc cài đặt chỉ số "w" cho ta thấy mức độ chi phí ta có thể bỏ ra để đảm bảo tính nhất quán của dữ liệu ngay lập tức.

Việc nhân bản dữ liệu cũng ảnh hưởng đến mức độ nhất quán của hệ thống. Mức độ nhất quán xét trên cả 2 phương diện đó là đọc và ghi dữ liệu. Để duy trì mức độ nhất quán của dữ liệu, Cassandra cung cấp cho người dùng nhiều mức độ nhất quán của các tác vụ đọc và ghi. Từ mức độ cao nhất đến thấp nhất, ta có thể điều chỉnh mức nhất quán dựa vào hai tham số cấu hình là "w" và "r" cùng với chỉ số "replication_factor". Trong đó, "w" là số node trả về khi ghi thành công, "r" là số node trả về khi đọc thành công. Nếu như tính nhất quán là sự ưu tiên, ta có thể đặt "w" và "r" sao cho đảm bảo

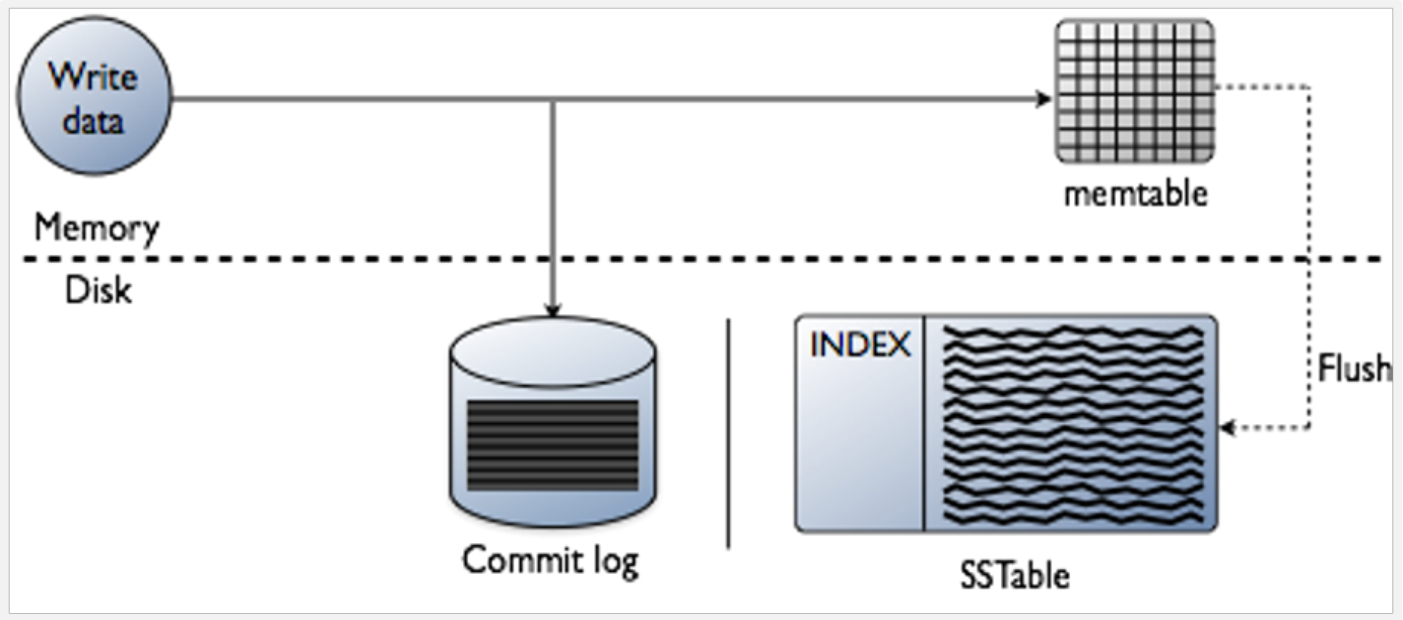
```
w + r > replication_factor
```

Và nên đảm bảo "w" hoặc "r" luôn nhỏ hơn replication_factor để cho được độ trễ tốt hơn. Giả sử như replication_factor = 3, vậy 2 giá trị của "w" và "r" tốt nhất sẽ là 2. Nghĩa là mỗi khi đọc và ghi dữ liệu, cần ít nhất 2 node trả về giá trị thì tác vụ đó coi là thành công. Và khi tác vụ đọc hoặc ghi thực hiện, sẽ luôn đảm bảo sẽ được thực hiện trên dữ liệu mới nhất mà tác vụ trước đó đã thực hiện.

Quản lý và truy xuất dữ liệu trong Cassandra

1. Ghi dữ liệu trong Cassandra

Cassandra được tối ưu để việc ghi dữ liệu luôn có sự sẵn sàng cao và nhanh chóng. Trong khi đó, RDBMS có cấu trúc sao cho dữ liệu dư thừa là ít nhất, thông tin cần cho câu truy vấn được lưu trữ ở nhiều bảng có quan hệ với nhau. Chính vì lưu trữ dữ liệu như thế nên việc ghi dữ liệu tốn nhiều chi phí. Server database phải thực hiện thêm nhiều tác vụ để đảm bảo toàn vẹn dữ liệu thông qua nhiều bảng quan hệ. Vì vậy RDBMS thường có hiệu suất không cao khi ghi dữ liệu.



Luồng ghi dữ liệu trong Cassandra

Cassandra đầu tiên ghi dữ liệu vào commit log, sau đó là ghi vào cấu trúc bảng trong bộ nhớ gọi là memtable. Tác vụ ghi là thành công khi nó ghi vào commit log và vào memtable. Vì vậy có rất ít tác vụ tương tác đĩa tại thời điểm đọc ghi dữ liệu.

Việc ghi dữ liệu được thực hiện định kì vào disk trong một cấu trúc bảng nhất quán gọi là SSTable. Memtable và SSTable được tổ chức theo column family. Memtable được tổ chức bằng việc sắp xếp theo row key và được đẩy xuống SSTable một cách tuần tự.

SSTable là inmutable (không thay đổi), chúng không được ghi lại sau khi đã flush. Có nghĩa rằng một row được lưu trữ thông qua nhiều file trong SSTable. Tại thời điểm đọc, một row phải được kết

hợp (combine) từ tất cả các SSTable trên đĩa để sinh ra dữ liệu được yêu cầu. Để tối ưu process này, Cassandra đã dùng một cấu trúc trong bộ nhớ gọi là Bloom filter. Mỗi SSTable có một Bloom filter kết hợp với nó, dùng để kiểm tra nếu một row key được yêu cầu tồn tại trong SSTable trước khi làm tác vụ tìm kiếm trên đĩa.

2. Truy xuất dữ liệu đồng thời trong Cassandra

Không giống với RDBMS, Cassandra không hỗ trợ đầy đủ các tính năng trong ACID như không có khóa (locking) hay độc lập thực thi khi update nhiều dòng hoặc column family. ACID là từ viết tắt dùng để miêu tả 4 hành vi mà transaction phải đạt được trong RDBMS:

- Atomic: Đảm bảo giao dịch transaction đó phải thành công hoặc được quay lại (roll back) khi thất bại.
- Consistent: Đảm bảo transaction không thể để cơ sở dữ liệu ở trạng tháng thiếu nhất quán.
- Isolated: Transaction này phải độc lập với những transaction khác.
- Durable: Đảm bảo dữ liệu không bị mất mát khi có sự cố với hệ thống hoặc node server.

Cassandra đánh đổi giữa isolation và atomic để có khả năng đáp ứng cao trong việc thực hiện việc ghi dữ liệu nhanh. Trong Cassandra, việc ghi là một tác vụ atomic ở cấp độ dòng, nghĩa là update hay insert column của một row key được xem như là một tác vụ đọc. Cassandra không hỗ trợ transaction của một update trên nhiều dòng nên sẽ không thể quay lại (roll back) khi mà một tác vụ thành công trên một bảng sao nhưng lại thất bại trên một bản sao khác.

Cassandra sử dụng timestamp để cập nhật thời gian tác động gần nhất vào column. Timestamp được hỗ trợ bởi ứng dụng client. Timestamp cuối cùng luôn được chọn khi truy vấn/cập nhật data. Vì thế nếu nhiều giao dịch cùng update lên cũng một cột, update gần nhất sẽ được chọn.

Tác vụ ghi trong Cassandra là bền bỉ (durable). Tất cả tác vụ sẽ được ghi lại trong bộ nhớ và trong commit log trước khi chúng được biết như là một sự thành công. Nếu có thất bại (crash) hay server bị sự cố trước khi memtable được đẩy xuống đĩa. Commit log được dùng để phục hồi lại (recover) tác vụ ghi.

3. Insert và Update dữ liệu trong Cassandra

Nhiều cột có thể được insert tại cùng một thời điểm. Khi insert hay update column trong column family, ứng dụng đặc tả rowkey để nhận dạng cột nào được update. Rowkey tương tự như primary key, phải là duy nhất trong mỗi dòng của column family. Mặc dù vậy không giống như primary key, insert một duplicate row key sẽ không báo lỗi như primary key, nó đơn giản xem như là một tác vụ update.

Column được ghi đè nếu timestamp trong version mới của column gần hơn so với timestamp hiện tại. Vì thế timestamp chính xác là cần thiết nếu việc update xảy ra thường xuyên.

4. Delete dữ liệu trong Cassandra

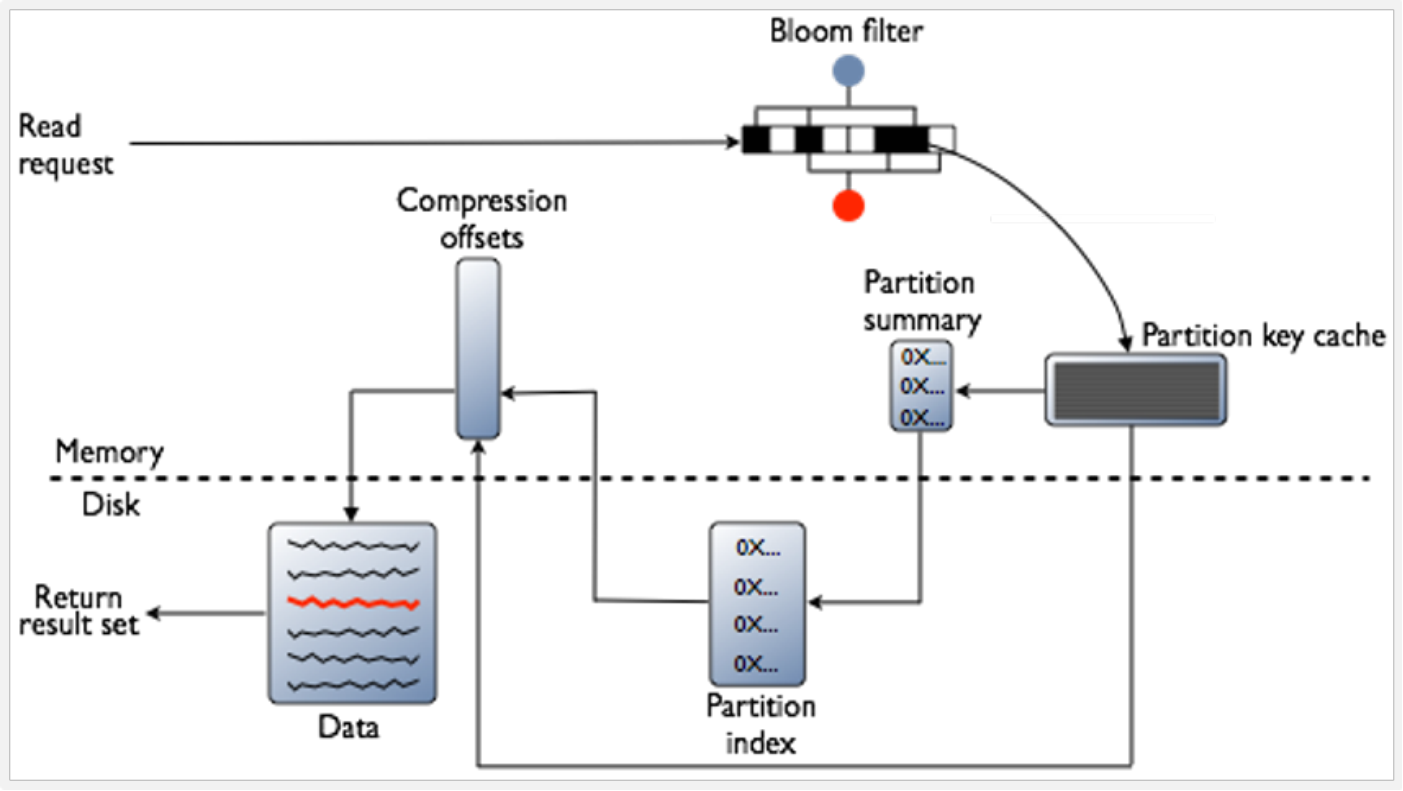
Khi thực hiện xóa (delete) dữ liệu trong Cassandra:

- Tác vụ xóa không thực hiện xóa dữ liệu trên đĩa ngay lập tức. Một lần SSTable được viết, nó sẽ không bị thay đổi (file không được update bởi DML), có nghĩa rằng cột bị xóa không bị xóa ngay lập tức. Hệ thống sẽ tạo một đánh dấu (maker) gọi là tombstone, để chỉ ra trạng thái của cột mới. Cột được đánh dấu bởi maker tồn tại trong khoảng thời gian được cấu hình trước, sau đó mới xóa vĩnh viễn bởi tác vụ compaction sau khi thời gian cấu hình bị quá hạn.
- Cột bị xóa có thể xuất hiện lại nếu routine node không chạy. Đánh dấu cột bị xóa bằng tombstone đảm bảo rằng một bản sao được đưa xuống tại thời điểm xóa sẽ bị xóa cuối cùng khi nó back up trở lại. Mặc dù vậy, nếu một node down lâu hơn thời điểm cấu hình để giữ tombstone thì node có thể mất dữ liệu bị xóa hoàn toàn, và nhân bản dữ liệu bị xóa 1 lần khi nó được back up trở lại. Để ngăn dữ liệu bị xóa xuất hiện lại, hệ thống cần phải điều chỉnh sửa chữa mỗi node.

Row key cho row bị xóa vẫn còn xuất hiện trong kết quả truy vấn (range query result). Khi bạn xóa một dòng trong Cassandra, nó đánh dấu tất cả các cột cho row key đó bằng tombstone. Cho đến khi nó bị xóa bởi compation bạn có một empty row key (1 row không có column). Những key bị xóa có thể được hiển thị trong get_range_slices(). Nếu ứng dụng client thực hiện câu truy vấn range lên row, ứng dụng có thể cũng cần phải loại bỏ đi những row không có cột nào.

5. Đọc dữ liệu trong Cassandra

Khi có một yêu cầu đọc lên row từ node, row phải được combine từ tất cả SSTable, cũng như là từ những memtable chưa flush xuống bộ nhớ để sinh ra dữ liệu yêu cầu. Để tối ưu quá trình này, Cassandra sử dụng cấu trúc gọi là Bloom filter cho mỗi SSTable, nó dùng để kiểm tra nếu dữ liệu cho những row được request tồn tại trong SSTable trước khi làm một số tác vụ truy vấn trên đĩa. Do đo, Cassandra có hiệu suất cao trong việc đọc khi so sánh với những hệ thống lưu trữ khác ngay cả khi có nhiều request đọc.



Luồng đọc dữ liệu trong Cassandra

Cũng giống như với một số cơ sở dữ liệu khác, tác vụ đọc là nhanh nhất khi dữ liệu yêu cầu nằm trong bộ nhớ memory. Mặc dù tất cả hệ thống lưu trữ mới thực hiện cache để cho phép truy xuất nhanh dữ liệu, nhưng không phải tất cả chúng đều được đảm bảo khi mà số tác vụ I/O vượt quá dung lượng của cache. Hiệu suất đọc dữ liệu của Cassandra cũng vậy, tuy nhiên, Cassandra dễ dàng khắc phục vấn đề này bằng cách thêm nhiều node cho cluster.

So sánh và đánh giá Cassandra và HBase

Cassandra và HBase là cơ sở dữ liệu phổ biến thuộc loại Column family.

1. Giống nhau

#	Thành phần	Nội dung
1	Database	Cả Cassandra và HBase đều là CSDL mã nguồn mở thuộc NoSQL. Có thể lưu trữ và xử lý lượng lớn dữ liệu bao gồm cả dữ liệu không có quan hệ.
2	Scalability (khả năng mở rộng)	Cả Cassandra và HBase đều có khả năng mở rộng cao, bằng cách tăng số lượng node trong cluster. Bởi vậy, cả hai đều là lựa chọn tốt trong việc xử lý dữ liệu lớn
3	Replication (nhân bản)	Luôn có khả năng xảy ra lỗi trong một chương trình hoặc ứng dụng, do đó, dữ liệu có thể bị mất. Nhưng cả Cassandra và HBase đều có giải pháp để bảo vệ và ngăn chặn việc mất dữ liệu ngay cả khi máy chủ bị lỗi. Điều này được thực hiện thông qua replication. Dữ liệu được ghi trên một node được sao chép đến nhiều node trong cluster. Do đó, nếu một node bị lỗi, luôn có một node dự phòng để truy cập dữ liệu.

2. Khác nhau

#	Thành phần	Cassandra	HBase
1	Infrastructure (Hạ tầng)	Cassandra được thiết kế để triển khai độc lập. Tuy nhiên, một số ứng dụng có thể kết hợp Cassandra với các DBMS khác hoặc cũng có thể triển khai tích hợp với Storm hoặc Hadoop.	HBase sử dụng cơ sở hạ tầng Hadoop. Cơ sở hạ tầng HBase-Hadoop này bao gồm một số thành phần như Zookeeper, HBase master, data node và name node.
2	Support	Cassandra hỗ trợ ordered partitioning (sắp xếp trên partition). Cassandra có một số hạn chế khi scan các row theo khoảng, và bộ đồng xử lý (coprocessor)	HBase không hỗ trợ ordered partitioning (sắp xếp trên partition) HBase cung cấp khả năng đồng xử lý. Khả năng này hỗ trợ các trình kích hoạt. Trong HBase, một row được phục vụ bởi chính xác một máy chủ tại một thời điểm. Do đó, nó không hỗ trợ cân bằng tải đọc đối với chỉ một row.
3	Node	Trong Cassandra, có các seed node. Các nút này đóng vai trò là các điểm giao tiếp giữa các cụm. Do đó, khả năng mở rộng và tính khả dụng cao trong Cassandra được đảm bảo bằng cách cho phép nhiều seed node trong một cụm.	Trong HBase, có các master node. Các nút chủ này giám sát và điều phối hoạt động của các máy chủ khu vực. Do đó, khả năng này được đảm bảo bởi các standby node trong HBase. Trong trường hợp master node bị lỗi, standby node sẵn sàng thế chỗ.
4	Giao tiếp giữa các node	Cassandra sử dụng Gossip Protocol để giao tiếp giữa các node, dữ liệu được sao chép từ node này sang node khác	HBase dựa vào giao thức Zookeeper để xác định các node, dữ liệu được sao chép, đọc ghi dữ liệu thông qua master node.
5	Query language (ngôn ngữ truy vấn)	Cassandra hỗ trợ sử dụng ngôn ngữ CQLSH tương tự ngôn ngữ SQL	HBase chỉ hỗ trợ HBase shell

Trong thực tế, nhà phát triển có thể kết hợp một hoặc một vài hệ cơ sở dữ liệu cho ứng dụng. Mỗi cơ sở dữ liệu sẽ có những điểm mạnh, điểm yếu riêng. Do đó, các nhà phát triển cần có sự hiểu biết về các hệ cơ sở dữ liệu và linh hoạt trong việc lựa chọn nền tảng nhằm đưa ra giải pháp tốt nhất cho bài toán.

Hướng dẫn cài đặt Cassandra Cluster

Môi trường cài đặt Cassandra Cluster:

- Centos 7
- Java 8

1. Cài đặt JDK

Thực hiện tải phiên bản JDK 8 trên trang chủ của Oracle:

```
https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
```

Thực hiện cài đặt JDK:

```
rpm -ivh jdk-8u211-linux-x64.rpm
```

Kiểm tra phiên bản JDK sau khi cài đặt:

```
$ java -version
java version "1.8.0_211"
Java(TM) SE Runtime Environment (build 1.8.0_211-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.211-b12, mixed mode)
```

2. Cài đặt python 2.7

```
# yum install epel-release
# wget https://centos6.iuscommunity.org/ius-release.rpm
# rpm -Uvh ius-release*.rpm
# yum install python27
```

3. Cài đặt Cassandra

Thực hiện cài đặt Cassandra trên các node của hệ thống.

```
# vi /etc/yum.repos.d/cassandra.repo

[cassandra]
name=Apache Cassandra
baseurl=https://www.apache.org/dist/cassandra/redhat/311x/
gpgcheck=0
enabled = 1
repo_gpgcheck=0
gpgkey=https://www.apache.org/dist/Cassandra/KEYS
#  yum install cassandra
```

- Cấu hình cluster:

```
cluster_name: 'Cluster_Name'
seed_provider:

  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
            parameters:
            - seeds: "<IP_LIST>"
listen_address: <IP_ADDRESS>
rpc_address: <IP_ADDRESS>
endpoint_snitch: GossipingPropertyFileSnitch
data_file_directories:
            - /data01/cassandra/data
commitlog_directory: /data01/cassandra/commitlog
saved_caches_directory: /data1/cassandra/saved_caches
```

- Start Cassandra:

```
#  service cassandra start
```

- Thiết lập tự chạy khi khởi động hệ điều hành:

```
#  service cassandra on
```