-

**ĐẠI HỌC QUỐC GIA TP.HCM**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA HỆ THỐNG THÔNG TIN**



****

**TIỂU LUẬN**

**CƠ SỞ DỮ LIỆU PHÂN TÁN**

**ĐỀ TÀI: SCYLLADB**

**Giáo viên hướng dẫn: Nguyễn Thị Kim Yến**

**Sinh viên thực hiện:**

**Nguyễn Hữu Thắng – 1952 2208**

**Hoàng Trí Tâm – 1952 2159**

**Nguyễn Quốc Lưu – 1952 1816**

**Phạm Minh Thắng – 1952 2215**

TP.HCM, tháng 12 năm 2021

MỤC LỤC

[BẢNG PHÂN CÔNG CÔNG VIỆC 3](#_Toc89900051)

[PHẦN 1: GIỚI THIỆU 4](#_Toc89900052)

[**1.** **ScyllaDB là gì** 4](#_Toc89900053)

[**2.** **Lịch sử hình thành** 4](#_Toc89900054)

[**3.** **Tác giả, tổ chức quản lý** 5](#_Toc89900055)

[**3.1 Tác giả** 5](#_Toc89900056)

[**3.2 Tổ chức quản lý** 5](#_Toc89900057)

[**4.** **Mô hình lưu trữ** 5](#_Toc89900058)

[**4.1 Kiến trúc Scylla** 5](#_Toc89900059)

[**4.2 Kiến trúc Scylla Shard-per-Core** 10](#_Toc89900060)

[**4.3 Kiến trúc ngang hàng** 11](#_Toc89900061)

[**4.4 Giao tiếp qua mạng của Scylla** 14](#_Toc89900062)

[**4.5 Quản lý bộ nhớ Scylla** 15](#_Toc89900063)

[**5.** **Ngôn ngữ thao tác với dữ liệu** 16](#_Toc89900064)

[**5.1 Các kiểu dữ liệu** 16](#_Toc89900065)

[**5.1 Định nghĩa dữ liệu** 17](#_Toc89900066)

[**5.3 CQL Shell (CQLSh)** 22](#_Toc89900067)

[**5.4 Thao tác dữ liệu** 23](#_Toc89900068)

[**6.** **Cơ chế phân tán** 24](#_Toc89900069)

[**6.1 Push** 25](#_Toc89900070)

[**6.2 Pull** 25](#_Toc89900071)

[PHẦN 2: HƯỚNG DẪN CÀI ĐẶT 25](#_Toc89900072)

[**1.** **Cài đặt Docker** 25](#_Toc89900073)

[**2.** **Cài đặt trên một máy** 25](#_Toc89900074)

[**2.1.** **Tải xuống Scylla** 25](#_Toc89900075)

[**2.2.** **Cài đặt Scylla** 26](#_Toc89900076)

[**2.3.** **Nodetool status** 27](#_Toc89900077)

[**2.4.** **Cấu hình Container** 28](#_Toc89900078)

[**3.** **Cài đặt trên cụm máy** 28](#_Toc89900079)

[**3.1 Cấu hình** 28](#_Toc89900080)

[**3.2 Cài đặt** 32](#_Toc89900081)

[**4. Cài đặt đa trung tâm dữ liệu** 32](#_Toc89900082)

[**4.1 Cấu hình** 32](#_Toc89900083)

[**4.2 Cài đặt** 34](#_Toc89900084)

[**4.3 Cấu hình các node** 34](#_Toc89900085)

[PHẦN 3: THỰC NGHIỆM MÔ PHỎNG PHÂN TÁN 35](#_Toc89900086)

[**1.** **Mô tả bài toán đặt ra với dữ liệu** 35](#_Toc89900087)

[**2.** **Mô tả cấu trúc dữ liệu sử dụng** 35](#_Toc89900088)

[**2.2 Nút** 35](#_Toc89900089)

[**2.3 Mảnh vỡ** 35](#_Toc89900090)

[**2.4 Bảng ghi nhớ** 36](#_Toc89900091)

[**3.** **Các bước thực nghiệm chi tiết** 36](#_Toc89900092)

[**3.1 Tạo dữ liệu cho máy** 36](#_Toc89900093)

[**3.2 Truy vấn dữ liệu ở các máy trong 1 cụm** 37](#_Toc89900094)

[**3.3 Phân tán dữ liệu ra các datacenter khác** 37](#_Toc89900095)

[**3.4 Configuring keyspaces cho Multi-DC** 38](#_Toc89900096)

[**3.5 Truy vấn dữ liệu ở các máy trong các datacenter** 40](#_Toc89900097)

# BẢNG PHÂN CÔNG CÔNG VIỆC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Hữu Thắng | Trí Tâm | Minh Thắng | Quốc Lưu |
| Tên HQT CSDL |  | x |  |  |
| Lịch sử hình thành |  | x |  |  |
| Tác giả, tổ chức quản lý |  |  |  | x |
| Mô hình lưu trữ |  |  |  | x |
| Ngôn ngữ thao tác với dữ liệu |  |  |  | x |
| Cơ chế phân tán |  |  | x |  |
| Cài đặt trên một máy | x |  |  |  |
| Cài đặt đa trung tâm dữ liệu | x |  |  |  |
| Cài đặt trên cụm máy phân tán |  |  | x |  |
| Mô tả bài toán đặt ra với dữ liệu |  |  |  | x |
| Mô tả cấu trúc dữ liệu sử dụng | x |  |  |  |
| Các bước thực nghiệm chi tiết |  | x |  |  |
| Hỗ trợ | x | x | x | x |
| Phần trăm công việc | 25% | 25% | 25% | 25% |

# PHẦN 1: GIỚI THIỆU

1. **ScyllaDB là gì**

* ScyllaDB là một cơ sở dữ liệu NoSQL dựa trên Cassandra.
* ScyllaDB là cơ sở dữ liệu lớn theo thời gian thực, với hiệu suất mở rộng 1.000.000 giây OPS cho mỗi nút, mở rộng quy mô lên 100 giây và độ trễ 99% dưới 1 mili giây.
* Lợi thế của Scylla:

Chúng tôi đã xây dựng cơ sở dữ liệu NoSQL của mình với phương pháp tiếp cận gần với phần cứng, không chia sẻ gì để tối ưu hóa hiệu suất, sử dụng đầy đủ các máy chủ đa lõi hiện đại và giảm thiểu chi phí cho DevOps. Scylla tương thích với API với cả Cassandra và DynamoDB, nhưng nhanh hơn, nhất quán hơn và có TCO thấp hơn.

1. **Lịch sử hình thành**

* ScyllaDB được thành lập bởi nhóm đã thiết kế và phát triển phần mềm giám sát máy ảo(hypervisor) KVM, phần mềm giám sát máy ảo (hypervisor) mặc định trong nhiều môi trường điện toán đám mây, bao gồm Google Compute Engine, Amazon Web Services và OpenStack.
* Nguồn gốc tên**:** Được lấy cảm hứng từ truyền thuyết quái vật Scylla, Scylla vốn là một nàng tiên cá xinh đẹp bị nữ phù thủy Circe ganh ghét nên đã lập mưu khiến Scylla uống phải thuốc độc biến thành quái vật biển.
* 2005: Avi Kivity và Dor Laor gặp nhau tại Qumranet, nơi Avi đã phát minh ra một mô hình phần mềm giám sát máy ảo (hypervisor) mới, một mô hình sử dụng khả năng tăng tốc phần cứng và phù hợp với thiết kế mọi thứ của Linux. Kernel Virtual Machine (KVM) sau đó đã trở thành tiêu chuẩn công nghiệp.
* 2014: Sau khi làm việc tại RedHat, nhóm đã khởi chạy ScyllaDB. Viết lại kiến trúc Cassandra từ đầu bằng mã gốc với tất cả kinh nghiệm nhiều năm về mã hóa hạt nhân / siêu giám sát, điều này đã trở thành cơ sở dữ liệu của Scylla.
* 2015: Bản beta Scylla được phát hành
* 2016: Scylla 1.0 được phát hành
* 2017:
* Scylla Enterprise ra mắt
* Mã nguồn mở Scylla 2.0 phát hành.
* 2018:
* Scylla Manager ra mắt
* Mã nguồn mở Scylla 3.0 ra mắt hỗ trợ OLTP và OLAP đồng thời
* 2020: Scylla 4.0 phát hành.

1. **Tác giả, tổ chức quản lý**

### **3.1 Tác giả**

ScyllaDB được thành lập bởi nhóm đã thiết kế và phát triển phần mềm giám sát máy ảo(hypervisor) KVM.

### **3.2 Tổ chức quản lý**

Đội quản lý ScyllaDB:

* CTO & Co-founder: Avi Kivity
* CEO & Co-founder: Dor Laor
* CFO: Danit Livne
* Giám đốc tài chính: Danit Livne
* Giám đốc doanh thu: Randall Jackson
* Giám đốc tiếp thị: Wayne Ariola
* Vice president of Research & Development: Shlomi Livne
* Vice president of Product: Tzach Livyatan
* Vice president of  Sales: Rich Nigro

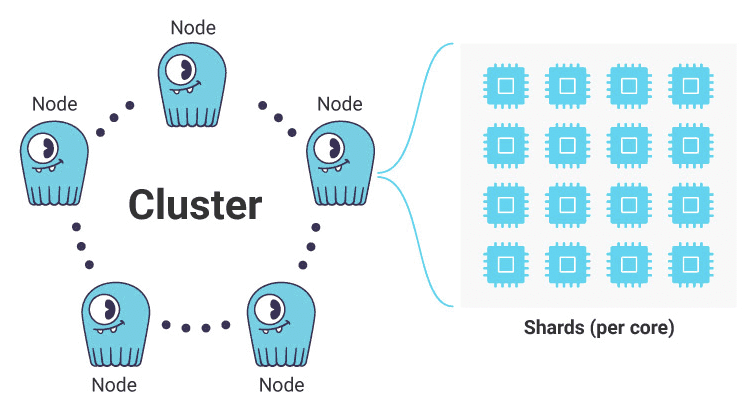
1. **Mô hình lưu trữ**

### **4.1 Kiến trúc Scylla**

Scylla được xây dựng trên kiến ​​trúc cơ sở dữ liệu NoSQL hiện đại được thiết kế từ đầu để có hiệu suất vượt trội và tính khả dụng cao đồng thời cung cấp khả năng tương thích API với Apache Cassandra và Amazon DynamoDB. Tuy nhiên, nó cung cấp những lợi thế đáng kể so với những cơ sở dữ liệu này.

Scylla là một cơ sở dữ liệu NoSQL phân tán thế hệ tiếp theo, được viết bằng C ++ để tận dụng tối đa các tính năng nguyên thủy cấp thấp của Linux. Nó có thể được xem xét theo nhiều cách khác nhau, cả về các lớp kiến ​​trúc vật lý và logic của nó.

* Kiến trúc máy chủ Scylla:

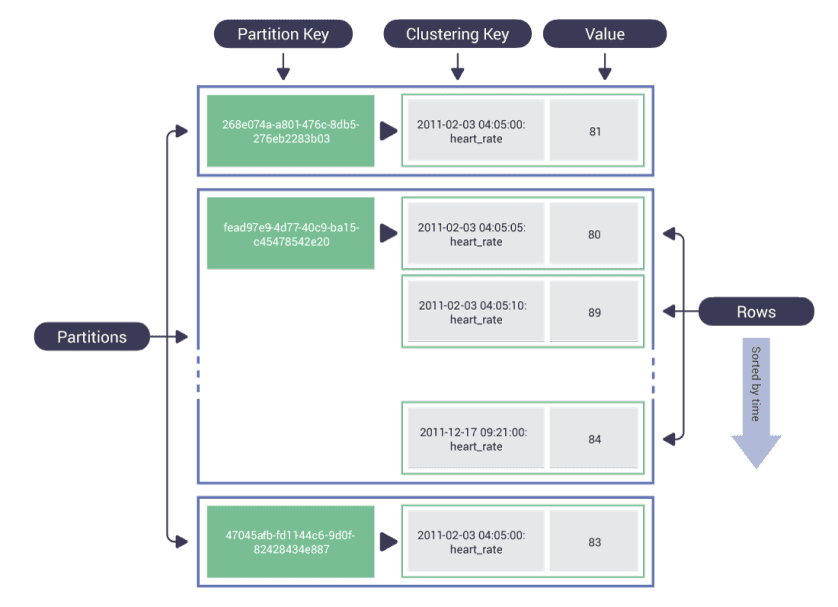


*Kiến trúc cụm và thiết kế phân đoạn cho mỗi lõi của Scylla*

|  |  |
| --- | --- |
| Cluster | Lớp đầu tiên được hiểu là một cụm, một tập hợp các nút được kết nối với nhau được tổ chức thành một kiến trúc vòng ảo mà qua đó dữ liệu được phân phối. Tất cả các nút được coi là ngang hàng nhau. Nếu không có một phân cấp xác định, cụm không có một điểm thất bại nào. Các cụm không cần phải lưu trữ tại một vị trí. Dữ liệu có thể được phân phối trên một cụm phân tán theo địa lý bằng cách sử dụng nhân rộng nhiều trung tâm dữ liệu. |
| Node | Đây có thể là các máy chủ tại chỗ riêng lẻ hoặc các máy chủ ảo (phiên bản public cloud) bao gồm một tập hợp con phần cứng trên một máy chủ vật lý lớn hơn. Dữ liệu của toàn bộ cụm được phân phối đồng đều nhất có thể trên các nút đó. Hơn nữa, Scylla sử dụng các đơn vị logic, được gọi là Nút ảo (vNodes), để phân phối dữ liệu tốt hơn cho hiệu suất đồng đều hơn. Một cụm có thể lưu trữ nhiều bản sao của cùng một dữ liệu trên các nút khác nhau để đảm bảo độ tin cậy. |
| Shard | Scylla phân chia dữ liệu hơn nữa, tạo ra các phân đoạn bằng cách gán một phân đoạn của tổng dữ liệu trong một nút cho một CPU cụ thể, cùng với bộ nhớ liên quan (RAM) và bộ lưu trữ liên tục (chẳng hạn như NVMe SSD). Shards hoạt động chủ yếu như các đơn vị hoạt động độc lập, được gọi là thiết kế “không có gì dùng chung”. Điều này làm giảm đáng kể sự tranh chấp và sự cần thiết của các khóa xử lý đắt tiền. |

* Kiến trúc dữ liệu

Mô hình dữ liệu của Scylla đã dẫn đến việc nó được gọi là cơ sở dữ liệu “cột rộng”, mặc dù đôi khi chúng tôi gọi nó là cơ sở dữ liệu “khóa-khóa-giá trị” để phản ánh các khóa phân vùng và phân cụm.

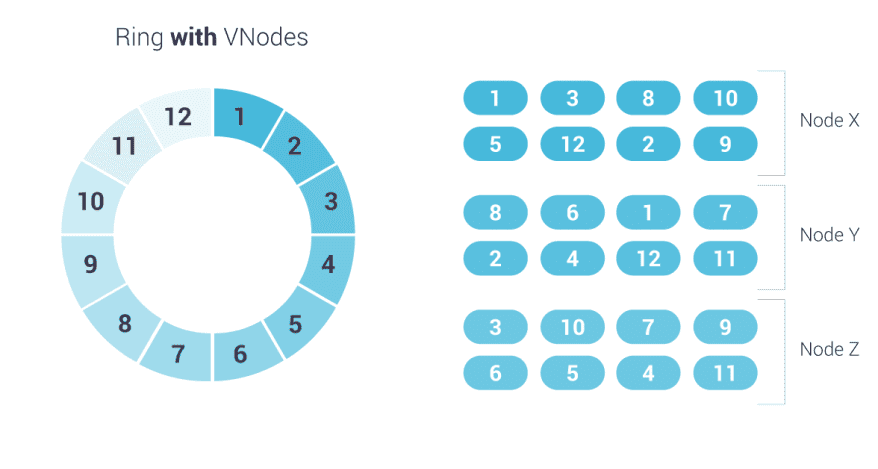


*Đây là một ví dụ về một vài phân vùng trong Scylla. Lưu ý rằng mỗi phân vùng có thể có số lượng hàng khác nhau.*

|  |  |
| --- | --- |
| Keyspace | Vùng chứa cấp cao nhất cho dữ liệu xác định chiến lược sao chép và hệ số sao chép (RF) cho dữ liệu được lưu giữ trong Scylla. Ví dụ: người dùng có thể muốn lưu trữ hai, ba hoặc thậm chí nhiều bản sao của cùng một dữ liệu để đảm bảo rằng nếu một hoặc nhiều nút bị mất, dữ liệu của họ sẽ vẫn an toàn. |
| Table | Trong một không gian khóa, dữ liệu được lưu trữ trong các bảng riêng biệt. Bảng là cấu trúc dữ liệu hai chiều bao gồm các cột và hàng. Không giống như các hệ thống SQL RDBMS, các bảng trong Scylla là độc lập; bạn không thể tạo THAM GIA trên các bảng. |
| Partition | Bàn trong Scylla có thể khá lớn, thường được đo bằng terabyte. Do đó, các bảng được chia thành các phần nhỏ hơn, được gọi là các phân vùng, để được phân phối đồng đều nhất có thể trên các phân đoạn. |
| Rows | Mỗi phân vùng chứa một hoặc nhiều hàng dữ liệu được sắp xếp theo một thứ tự cụ thể. Không phải mọi cột đều xuất hiện trong mỗi hàng. Điều này làm cho Scylla hiệu quả hơn để lưu trữ những gì được gọi là “dữ liệu thưa thớt”. |
| Columns | Dữ liệu trong một hàng của bảng sẽ được chia thành các cột. Mục nhập hàng và cột cụ thể sẽ được gọi là một ô. Một số cột sẽ được sử dụng để xác định cách dữ liệu sẽ được lập chỉ mục và sắp xếp, được gọi là các khóa phân vùng và phân cụm. |

Scylla bao gồm các phương pháp để tìm các phân vùng đặc biệt lớn và các hàng lớn có thể gây ra các vấn đề về hiệu suất. Scylla cũng lưu trữ các hàng được sử dụng thường xuyên nhất trong bộ nhớ đệm dựa trên bộ nhớ, để tránh việc tra cứu dựa trên đĩa đắt tiền trên cái gọi là “phân vùng nóng”.

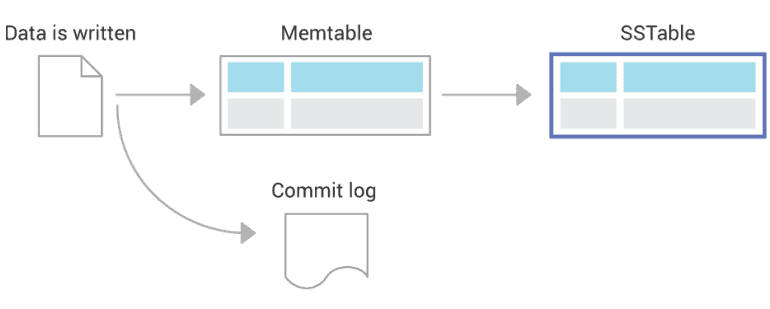
* Kiến trúc vòng



*Một ví dụ về phân phối vNode cho không gian khóa có hệ số sao chép (RF) là 2. Vì vậy, mỗi vNode được sao chép hai lần và không thể được lưu trữ trên cùng một nút vật lý.*

|  |  |
| --- | --- |
| Ring | Tất cả dữ liệu trong Scylla có thể được hình dung dưới dạng một vòng các phạm vi mã thông báo, với mỗi phân vùng được ánh xạ tới một mã thông báo băm duy nhất (tuy nhiên, ngược lại, mã thông báo có thể được liên kết với một hoặc nhiều phân vùng). Các mã thông báo này được sử dụng để phân phối dữ liệu xung quanh cụm, cân bằng nó đồng đều nhất có thể giữa các nút và phân đoạn. |
| vNode | Vòng được chia thành vNodes (Virtual Nodes) bao gồm một loạt các mã thông báo được chỉ định cho một nút hoặc phân đoạn vật lý. Các vNode này được sao chép nhiều lần trên các nút vật lý dựa trên hệ số sao chép (RF) được đặt cho không gian khóa. |

* Kiến trúc lưu trữ



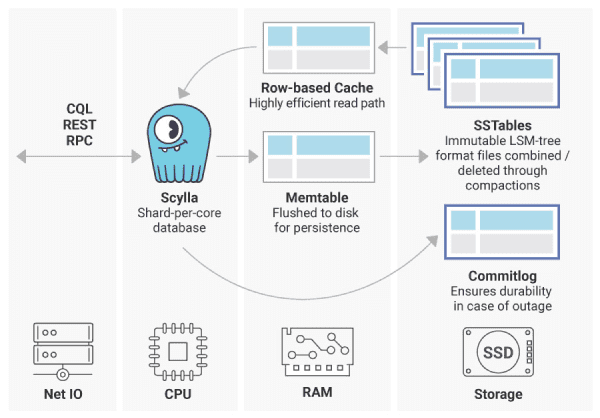
*Đường dẫn ghi của Scylla đặt dữ liệu vào một cấu trúc trong bộ nhớ được gọi là bảng ghi nhớ và cũng ghi cập nhật một cách hợp lý vào một cam kết liên tục. Theo thời gian, nội dung của bảng ghi nhớ được chuyển sang một tệp liên tục mới, bất biến trên đĩa, được gọi là Bảng chuỗi được sắp xếp (SSTable).*

|  |  |
| --- | --- |
| Memtable | Trong dữ liệu đường dẫn ghi của Scylla lần đầu tiên được đưa vào một bảng ghi nhớ, được lưu trữ trong RAM. Trong thời gian này, dữ liệu này sẽ được lưu vào đĩa để duy trì. |
| Commitlog | Một bản ghi chỉ nối thêm các hoạt động của nút cục bộ, được ghi đồng thời khi dữ liệu được gửi đến một bảng ghi nhớ. Điều này cung cấp sự bền bỉ (độ bền của dữ liệu) trong trường hợp nút tắt; khi máy chủ khởi động lại, bản ghi cam kết có thể được sử dụng để khôi phục một bảng ghi nhớ. |
| SSTables | Dữ liệu được lưu trữ liên tục trong Scylla trên cơ sở mỗi phân đoạn bằng cách sử dụng các tệp không thể thay đổi (không thể thay đổi, chỉ đọc) được gọi là Bảng chuỗi được sắp xếp (SSTables), sử dụng định dạng cây Hợp nhất có cấu trúc (LSM). Sau khi dữ liệu được chuyển từ một bảng ghi nhớ sang một bảng SSTable, bảng ghi nhớ (và đoạn commitlog được liên kết) có thể bị xóa. Các cập nhật cho các bản ghi không được ghi vào SSTable ban đầu, mà được ghi vào một SSTable mới. Scylla có các cơ chế để biết phiên bản nào của một bản ghi cụ thể là mới nhất. |
| Tombstones | Khi một hàng bị xóa khỏi SSTable, Scylla đặt một điểm đánh dấu được gọi là bia mộ vào một SSTable mới. Điều này phục vụ một lời nhắc cho cơ sở dữ liệu để bỏ qua dữ liệu gốc đang bị xóa. |
| Compactions | Sau khi một số SSTables được ghi vào đĩa, Scylla biết để chạy một nén, một quá trình chỉ lưu trữ bản sao mới nhất của một bản ghi và xóa bất kỳ bản ghi nào được đánh dấu bằng bia mộ. Khi SSTable nén mới được ghi, các SSTable cũ, lỗi thời sẽ bị xóa và giải phóng dung lượng trên đĩa. |
| Compaction Strategy | Scylla sử dụng các thuật toán khác nhau, được gọi là chiến lược, để xác định thời điểm và cách thức tốt nhất để chạy các giao dịch. Chiến lược xác định sự cân bằng giữa ghi, đọc và khuếch đại không gian. Scylla Enterprise thậm chí còn hỗ trợ một phương pháp duy nhất được gọi là Chiến lược nén gia tăng có thể tiết kiệm đáng kể chi phí trên đĩa. |

* Kiến trúc Client – Server

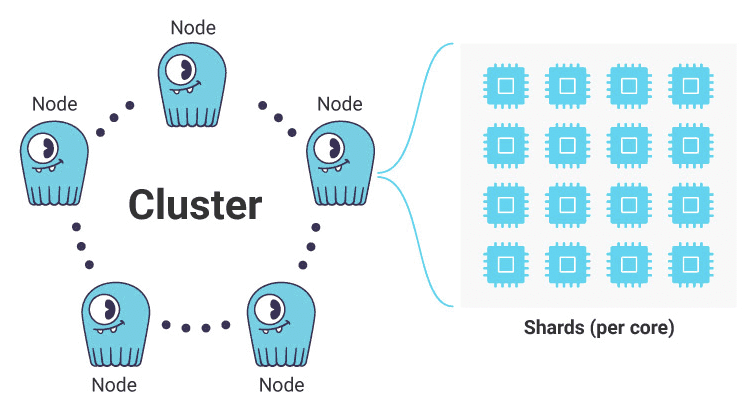
|  |  |
| --- | --- |
| Trình điều khiển | Scylla giao tiếp với các ứng dụng thông qua trình điều khiển cho giao diện CQL tương thích với Cassandra hoặc với SDK cho API DynamoDB của nó. Bạn có thể sử dụng nhiều trình điều khiển hoặc máy khách kết nối với Scylla cùng một lúc để cho phép thông lượng tổng thể lớn hơn. Các trình điều khiển được viết riêng cho Scylla có khả năng nhận biết phân đoạn, do đó cho phép giao tiếp trực tiếp với các nút trong đó dữ liệu được lưu trữ, do đó cung cấp khả năng truy cập nhanh hơn và hiệu suất tốt hơn. |

### **4.2 Kiến trúc Scylla Shard-per-Core**



*Mô tả đơn giản về thiết kế phân đoạn trên mỗi lõi của Scylla, với đường dẫn đọc ở trên cùng và đường dẫn ghi bên dưới.*

Scylla, cơ sở dữ liệu dữ liệu lớn NoSQL thời gian thực , được xây dựng hoàn toàn bằng C ++ với đầy đủ kiến ​​thức và kinh nghiệm về cách tận dụng phần cứng máy chủ NUMA đa lõi, đa xử lý hiện đại và các tính năng cơ bản của hệ điều hành Linux. Scylla là một công cụ cơ sở dữ liệu song song lớn, chạy phân mảnh trên mỗi lõi trong máy chủ của bạn, trên tất cả các máy chủ trong một cụm. Thiết kế của nó cho phép Scylla chạy hàng triệu thao tác mỗi giây ở độ trễ trung bình dưới mili giây.

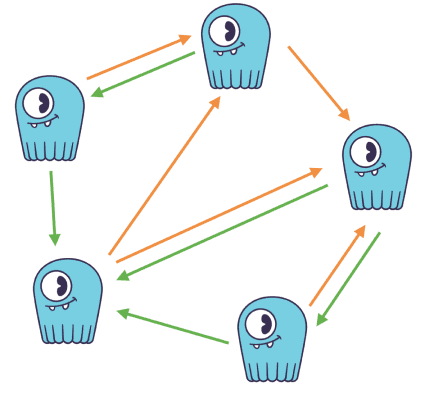


*Mỗi nút trong một cụm Scylla hoạt động ngang hàng; không có mô hình chính, bản sao để tạo ra tắc nghẽn hoặc các điểm lỗi đơn lẻ. Trong mỗi nút, dữ liệu được chia nhỏ cho mỗi lõi.*

Scylla sử dụng một thiết kế không đồng bộ cao, không chia sẻ, dựa trên khung Seastar cơ bản của nó. Mỗi phân đoạn dữ liệu được phân bổ CPU, RAM, tài nguyên mạng và lưu trữ liên tục mà nó sử dụng hiệu quả nhất có thể. Với các bộ lập lịch tùy chỉnh của riêng mình để xử lý CPU và I / O, Scylla biết cách đạt được hiệu quả tối đa từ cơ sở hạ tầng Dữ liệu lớn của bạn.

### **4.3 Kiến trúc ngang hàng**

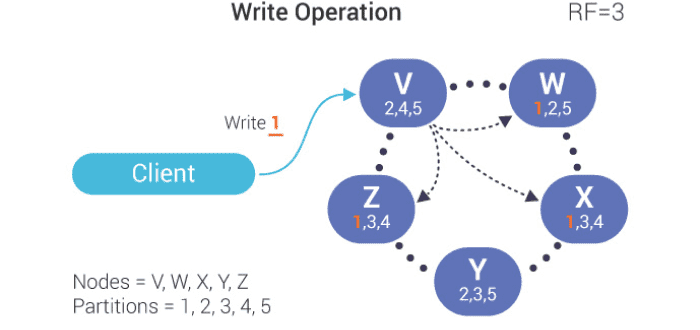
Khi Scylla khởi động, các nút sử dụng giao thức gossip để khám phá các nút ngang hàng để thiết lập cụm. Kiến ​​trúc cơ bản là không có phân cấp và không có bản sao. Trên thực tế, với Scylla, chúng tôi thậm chí đã loại bỏ khái niệm về các nút hạt giống được tìm thấy trong các triển khai đồn thổi khác. Nó hoàn toàn ngang hàng. Cơ chế tin đồn này cũng được sử dụng trong trường hợp thay đổi cấu trúc liên kết, chẳng hạn như thêm hoặc xóa một nút, hoặc trong trường hợp nút không mong muốn, cung cấp khả năng phục hồi mạnh mẽ cho một cụm Scylla.



*Trong những người đồng nghiệp nói chuyện phiếm hoạt động song song, mỗi người ngang hàng giao tiếp với một hoặc nhiều đối tác được chọn ngẫu nhiên để thiết lập nhóm.*

* Sao chép dữ liệu tự động

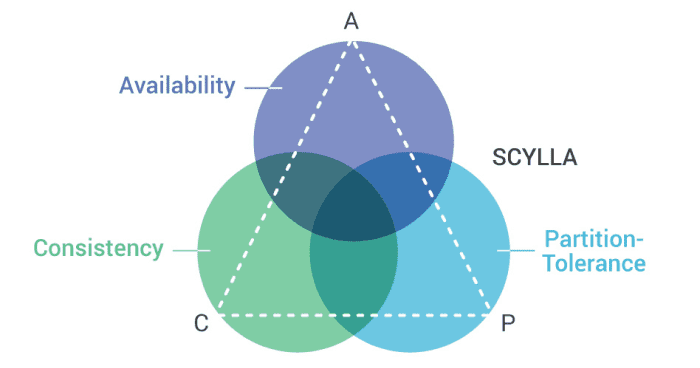
Scylla cho phép người dùng thiết lập hệ số sao chép (RF), có nghĩa là nhiều bản sao của cùng một dữ liệu có thể được lưu trữ trên nhiều nút trên toàn bộ cụm. Bằng cách này, ngay cả khi một nút bị mất, dữ liệu vẫn nằm ở đâu đó trong cụm.



*Trong thao tác ghi này, một bản cập nhật cho phân vùng 1, dữ liệu được chuyển cùng từ nút điều phối, V, đến ba nút nơi dữ liệu cư trú: W, X và Z.*

Đối với nhiều trường hợp sử dụng có tính khả dụng cao, thiết lập hệ số nhân rộng là ba (3) là đủ. Trong những trường hợp như vậy, ngay cả khi hai trong số ba bản sao của dữ liệu không có sẵn, dữ liệu sẽ nằm ở đâu đó trực tiếp trong cụm.

Với hệ số sao chép được thiết lập đúng cách, có thể đạt được thời gian chết bằng không. Người dùng có thể xác định hệ số sao chép của riêng họ, dựa trên trường hợp sử dụng của họ. Có những thời điểm khi hệ số nhân bản của hai có thể là đủ, và những thời điểm khi hệ số nhân bản của năm có thể được yêu cầu. Scylla tự động đảm nhận việc sao chép dữ liệu trong nền. Bạn chỉ cần đặt hệ số sao chép và cụm xử lý phần còn lại.



*Định lý CAP chỉ ra rằng cơ sở dữ liệu phân tán chỉ có thể duy trì đồng thời hai trong ba đặc tính quan trọng: tính nhất quán, tính khả dụng hoặc dung sai phân vùng. Scylla tập trung vào tính khả dụng cao ("A") và khả năng chịu đựng của phân vùng ("P"), do đó được gọi là hệ thống chế độ "AP".*

* Scylla và Định lý CAP

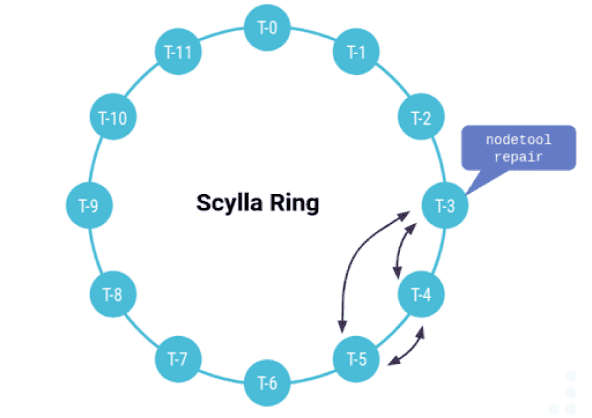
Định lý CAP dựa trên giả thuyết rằng các hệ thống có thể chọn cung cấp tính nhất quán, tính khả dụng hoặc dung sai phân vùng và rằng các nhà thiết kế cơ sở dữ liệu sẽ phải chọn hai trong ba đặc điểm đó. Bất kỳ cơ sở dữ liệu phân tán nào đều yêu cầu dung sai phân vùng - khả năng tiếp tục hoạt động ngay cả khi một phần của hệ thống bị gián đoạn do lỗi mạng hoặc máy chủ. Do đó, hai chế độ cơ sở dữ liệu phổ biến hiện nay là phổ biến:

* Hệ thống chế độ CP: ưu tiên tính nhất quán và dung sai phân vùng
* Hệ thống chế độ AP: ưu tiên tính khả dụng và dung sai phân vùng
* Tính nhất quán có thể điều chỉnh được

Tính nhất quán trong Scylla là có thể điều chỉnh được - người dùng có thể cho phép các giao dịch của họ có các mức độ nhất quán khác nhau

* Chống Entropy

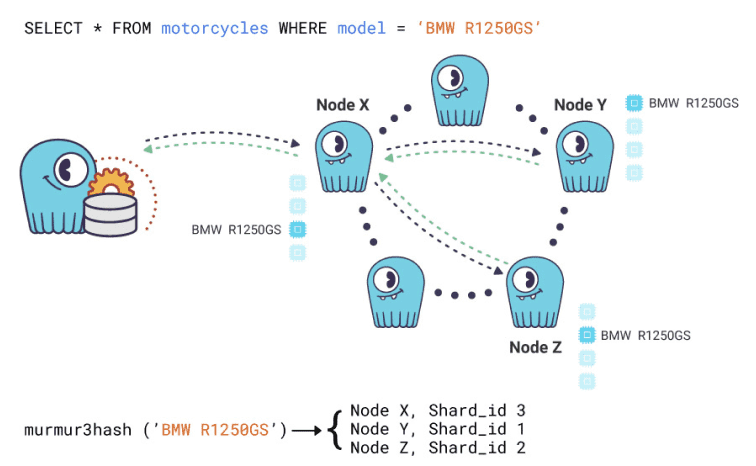
Scylla được thiết kế để hoạt động ngay cả trong trường hợp nút tạm thời không có sẵn (khi nó cuối cùng tham gia lại cụm) hoặc lỗi nút (khi nó phải được thay thế). Nhưng khi những tình huống đó xảy ra, hệ thống phải chiến đấu chống lại entropy và đưa cụm trở lại hoạt động hoàn toàn. Các quy trình và tính năng sau được thiết kế để giảm thiểu điều đó



*Sửa chữa Nodetool là một tiện ích chống entropy chạy ở chế độ nền và đồng bộ hóa dữ liệu giữa các nút.*

### **4.4 Giao tiếp qua mạng của Scylla**

* Scylla tối ưu hóa giao tiếp mạng bằng cách sử dụng các phương pháp như trình điều khiển nhận dạng mảnh và phát trực tuyến RPC, đồng thời duy trì bảo mật cơ sở dữ liệu NoSQL mạnh mẽ.
* Scylla cung cấp các trình điều khiển nhận biết phân đoạn bằng Python, Java và Go. Các trình điều khiển này cung cấp hiệu suất vượt trội bằng cách sử dụng băm Murmur3 và chính sách nhận dạng mã thông báo để biết nút nào và cụ thể là CPU nào trong dữ liệu nút đó.

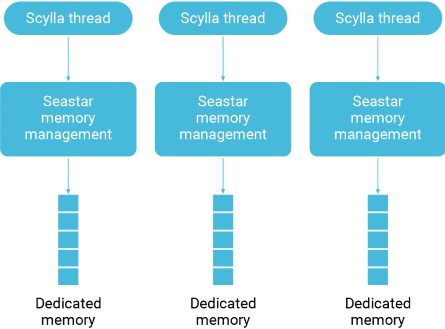


*Các trình điều khiển nhận biết phân đoạn của Scylla đảm bảo máy khách kết nối với nút điều phối là bản sao dữ liệu đã biết để có độ trễ thấp nhất và truyền dữ liệu nhanh nhất.*

* Mạng an toàn: Scylla coi trọng các mối đe dọa đối với mạng và áp dụng các phương pháp và giao thức bảo mật mạnh mẽ, bao gồm xác thực, kiểm soát truy cập dựa trên vai trò (RBAC), ủy quyền và mã hóa. Người dùng có thể áp dụng mã hóa cho dữ liệu đang chuyển giữa các máy khách và các nút máy chủ cũng như giữa các nút máy chủ.

### **4.5 Quản lý bộ nhớ Scylla**

* Scylla tận dụng tối đa tất cả bộ nhớ có sẵn để cung cấp thời gian phản hồi truy vấn cực nhanh với độ trễ thấp nhất quán.
* Cách Scylla tối đa hóa việc sử dụng bộ nhớ:



* Trong quá trình khởi động, Scylla kiểm tra phần cứng của nút và cố gắng lấy tất cả bộ nhớ có sẵn cho chính nó (ngoại trừ bộ nhớ dành riêng cho hệ điều hành), vì bộ nhớ là tài nguyên quan trọng nhất cho bất kỳ cơ sở dữ liệu NoSQL nào. Bộ nhớ được cấp phát được chia và gán cho từng phân đoạn luồng đơn chạy trong nút, mỗi phân đoạn được ghim vào một lõi CPU khác nhau. Cách tiếp cận này cho phép Scylla phân bổ hiệu quả bộ nhớ cho mỗi lõi của CPU theo cách thân thiện với NUMA không dùng chung và tránh mọi hoạt động chặn điển hình hoặc khóa bộ nhớ, và được gọi là quản lý bộ nhớ không khóa.

**LƯU Ý**: Trong môi trường chia sẻ, điều này có thể ảnh hưởng nghiêm trọng đến hiệu suất hoặc quá trình thực thi của các ứng dụng đang chạy khác. Trong các triển khai như vậy, người dùng có thể cấu hình giới hạn dung lượng bộ nhớ mà Scylla sử dụng.

1. **Ngôn ngữ thao tác với dữ liệu**

### **5.1 Các kiểu dữ liệu**

|  |  |  |
| --- | --- | --- |
| Kiểu dữ liệu | hằng số được hỗ trợ | Mô tả |
| ascii | string | Chuỗi ký tự ASCII |
| bigint | integer | có độ dài 64-bit |
| blob | blob | Các byte tùy ý (không xác định) |
| boolean | boolean | True hoặc false |
| counter | integer | Cột bộ đếm (giá trị có dấu 64-bit) |
| date | integer, string | Thời gian |
| decimal | integer, float | Số thập phân có độ chính xác thay đổi |
| double | integer, float | Số thực có độ dài 64-bit |
| duration | duration | Một khoảng thời gian với độ chính xác nano giây |
| float | integer, float | Số thực có độ dài 32-bit |
| inet | string | Địa chỉ IP, IPv4 (dài 4 byte) hoặc IPv6 (dài 16 byte). Lưu ý rằng không có inethằng số, địa chỉ IP phải được nhập dưới dạng chuỗi |
| int | integer | Có độ dài 16-bit |
| text | varchar | string | Chuỗi được mã hóa UTF8 |
| timestamp | integer, string | Thời gian (ngày và giờ) với độ chính xác mili giây |
| timeuuid | uuid | thường được sử dụng làm dấu thời gian "không có xung đột" |
| tinyint | integer | Số nguyên có độ dài 8-bit |
| uuid | uuid |  |
| varint | integer | Số nguyên chính xác tùy ý |

### **5.1 Định nghĩa dữ liệu**

CQL lưu trữ dữ liệu trong các bảng , có lược đồ xác định cách bố trí của dữ liệu đã nói trong bảng và các bảng đó được nhóm lại trong các keyspaces. Keyspaces xác định một số tùy chọn áp dụng cho tất cả các bảng mà nó chứa, trong đó nổi bật nhất là chiến lược sao chép được sử dụng bởi keyspaces. Một ứng dụng chỉ có thể có một keyspace. Tuy nhiên, cũng có thể có nhiều keyspaces trong trường hợp ứng dụng của bạn có các yêu cầu sao chép khác nhau.

Các câu lệnh để tạo, sửa, xoá keyspaces và bảng:

* Tạo KEYSPACE:
* Cú pháp:

**CREATE** **KEYSPACE** [**IF** **NOT** **EXISTS**] `keyspace\_name` **WITH** `**options**`

* Ví dụ:

**CREATE** **KEYSPACE** Excalibur

**WITH** replication = {'class': 'NetworkTopologyStrategy', 'DC1': 1, 'DC2': 3}

**AND** durable\_writes = true;

* Các hỗ trợ options là:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tên** | **Loại** | **Bắt buộc** | **Mặc định** | **Mô tả** |
| replication | map | có |  | Thực hiện sao chép và các tùy chọn để sử dụng cho keyspaces |
| durable\_writes | simple | không | true | Sử dụng commit log để cập nhật trên keyspace này |

Thuộc tính replication là bắt buộc và ít nhất phải chứa tùy chọn con 'class', định nghĩa class replication sẽ sử dụng. Phần còn lại của các tùy chọn phụ phụ thuộc vào replication nào được sử dụng. Theo mặc định, Scylla hỗ trợ 'class' sau

* Sử dụng KEYSPACE: **USE** `keyspace\_name`
* ALTER KEYSPACE: **ALTER** **KEYSPACE** `keyspace\_name` **WITH** `**options**`
* DROP KEYSPACE: **DROP** **KEYSPACE** [**IF** **EXISTS**] `keyspace\_name`
* Tạo TABLE:

create\_table\_statement: **CREATE** **TABLE** [**IF** **NOT** **EXISTS**] `table\_name`

: '('

: `column\_definition`

: (',' `column\_definition`) \*

: [',' **PRIMARY** **KEY** '(' `primary\_key` ')']

: ')' [ **WITH** `table\_options`]

column\_definition: `column\_name` `cql\_type` **STATIC**] [**PRIMARY** **KEY**]

primary\_key: `partition\_key` [ ',' `clustering\_columns`]

partition\_key: `column\_name`

: | '(' `column\_name` (',' `column\_name`) \*')'

clustering\_columns: `column\_name` (',' `column\_name`) \*

table\_options: **COMPACT** **STORAGE** [**AND** `table\_options`]

: | **CLUSTERING** **ORDER** **BY** '(' `clustering\_order` ')' [ **AND** `table\_options`]

: | scylla\_encryption\_options: '=' '{'[`cipher\_algorithm`: <hash>]','[`secret\_key\_strength`: <len>]','[`key\_provider`: <provider>]'}'

: | caching '=' ' {'caching\_options'}'

: | `**options**`

clustering\_order: `column\_name` (**ASC** | **DESC**) (',' `column\_name` (**ASC** | **DESC**)) \*

* Ví dụ:

**CREATE** **TABLE** monkeySpecies (

species text **PRIMARY** **KEY**,

common\_name text,

population varint,

average\_size int

) **WITH** comment='Important biological records'

**AND** read\_repair\_chance = 1.0;

**CREATE** **TABLE** timeline (

userid uuid,

posted\_month int,

posted\_time uuid,

body text,

posted\_by text,

**PRIMARY** **KEY** (userid, posted\_month, posted\_time)

) **WITH** compaction = {'class': 'LeveledCompactionStrategy'};

**CREATE** **TABLE** loads (

machine inet,

cpu int,

mtime timeuuid,

load float,

**PRIMARY** **KEY** ((machine, cpu), mtime)

) **WITH** **CLUSTERING** **ORDER** **BY** (mtime **DESC**);

**CREATE** **TABLE** users\_picture (

userid uuid,

pictureid uuid,

body text,

posted\_by text,

**PRIMARY** **KEY** (userid, pictureid, posted\_by)

) **WITH** compression = {'sstable\_compression': 'LZ4Compressor'};

**CREATE** **TABLE** data\_atrest (

pk text **PRIMARY** **KEY**,

c0 int

) **WITH** scylla\_encryption\_options = {

'cipher\_algorithm': 'AES/ECB/PKCS5Padding',

'secret\_key\_strength': 128,

'key\_provider': 'LocalFileSystemKeyProviderFactory',

'secret\_key\_file': '/etc/scylla/data\_encryption\_keys/secret\_key'};

**CREATE** **TABLE** caching (

k int **PRIMARY** **KEY**,

v1 int,

v2 int,

) **WITH** caching = {'enabled': 'true'};

* Một số ví dụ về định nghĩa khóa chính là:

PRIMARY KEY (a): a là khóa phân vùng và không có các cột phân cụm.

PRIMARY KEY (a, b, c): a là khóa phân vùng, b và c là các cột phân nhóm.

PRIMARY KEY ((a, b), c): a và b soạn khóa phân vùng (cái này thường được gọi là khóa phân vùng tổng hợp), và c là cột phân cụm.

* ALTER TABLE: **ALTER** **TABLE** `table\_name` `alter\_table\_instruction`
* DROP TABLE: **DROP TABLE** [IF EXISTS] table\_name

### **5.3 CQL Shell (CQLSh)**

* cqlsh là dùng command line shell để tương tác với Cassandra thông qua CQL (Ngôn ngữ truy vấn Cassandra). Nó được vận chuyển cùng với mọi gói Cassandra và có thể được tìm thấy trong thư mục bin / cùng với tệp thực thi Cassandra. cqlsh sử dụng trình điều khiển giao thức gốc Python và kết nối với nút đơn được chỉ định trên command line. cqlsh tương thích với Python 2.7.
* Tuỳ chọn Dependencies:
* Pytz: Theo mặc định, cqlsh hiển thị tất cả các timestamps có múi giờ UTC. Để hỗ trợ hiển thị timestamps với múi giờ khác, thư viện pytz phải được cài đặt. Xem timezone tùy chọn trong cqlshrc để chỉ định múi giờ để sử dụng
* Cython: Hiệu suất của các hoạt động COPY của cqlsh có thể được cải thiện bằng cách cài đặt cython. Điều này sẽ biên dịch các mô-đun python trung tâm cho hiệu suất của COPY
* Cqlshrc: File cqlshrc chứa các tùy chọn cấu hình cho cqlsh. Theo mặc định, điều này nằm trong thư mục chính của người dùng: ~/.cassandra/cqlsh, nhưng một vị trí tùy chỉnh có thể được chỉ định bằng tùy chọn --cqlshrc. Bạn có thể tìm thấy tài liệu và giá trị cấu hình ví dụ trong tệp conf / cqlshrc.sample của cài đặt tarball.
* Tuỳ chọn Command Line:
* Cú pháp: cqlsh [options] [host [port]]
* Sử dụng SSL khi kết nối với Cassandra: -u --user
* Tên người dùng để xác thực với Cassandra bằng: -p --password
* Mật khẩu để xác thực với Cassandra phải được sử dụng cùng với –user: -k --keyspace
* Keyspace để xác thực phải được sử dụng cùng với –user: -f --file
* Thực thi các lệnh từ tệp đã cho, sau đó thoát: --debug
* In thông tin gỡ lỗi bổ sung: --encoding
* Chỉ định mã hóa không mặc định cho đầu ra (mặc định là UTF-8): --cqlshrc
* Chỉ định một vị trí không phải mặc định cho cqlshrctệp: -e --execute
* Thực hiện câu lệnh đã cho, sau đó thoát: --connect-timeout
* Chỉ định thời gian chờ kết nối tính bằng giây (mặc định là 2 giây): --request-timeout
* Chỉ định thời gian chờ yêu cầu tính bằng giây (mặc định là 10 giây): -t –tty

### **5.4 Thao tác dữ liệu**

* Các câu lệnh được CQL hỗ trợ để chèn, cập nhật, xóa và truy vấn dữ liệu tương tự như khi truy vấn bằng SQL
* BATCH: Nhiều câu lệnh INSERT, UPDATEvà DELETE có thể được thực hiện trong một lệnh đơn bằng cách nhóm chúng thông qua một lệnh BATCH

**batch\_statement** ::= BEGIN [ UNLOGGED | COUNTER ] BATCH

[ USING [update\_parameter](https://docs.scylladb.com/getting-started/dml/#grammar-token-update-parameter) ( AND [update\_parameter](https://docs.scylladb.com/getting-started/dml/#grammar-token-update-parameter) )\* ]

[modification\_statement](https://docs.scylladb.com/getting-started/dml/#grammar-token-modification-statement) (';' [modification\_statement](https://docs.scylladb.com/getting-started/dml/#grammar-token-modification-statement) )\*

APPLY BATCH

**modification\_statement** ::= [insert\_statement](https://docs.scylladb.com/getting-started/dml/#grammar-token-insert-statement) | [update\_statement](https://docs.scylladb.com/getting-started/dml/#grammar-token-update-statement) | [delete\_statement](https://docs.scylladb.com/getting-started/dml/#grammar-token-delete-statement)

Ví dụ:

**BEGIN** **BATCH**

**INSERT** **INTO** **users** (userid, **password**, name) **VALUES** ('user2', 'ch@ngem3b', 'second user');

**UPDATE** **users** **SET** **password** = 'ps22dhds' **WHERE** userid = 'user3';

**INSERT** **INTO** **users** (userid, **password**) **VALUES** ('user4', 'ch@ngem3c');

**DELETE** name **FROM** **users** **WHERE** userid = 'user1';

**APPLY** **BATCH**;

* Câu lệnh BATCH nhóm nhiều câu lệnh sửa đổi (insert, update, delete) thành một câu lệnh duy nhất. Nó phục vụ một số mục đích:
* Nó lưu các chuyến đi vòng quanh mạng giữa máy khách và máy chủ (và đôi khi giữa bộ điều phối máy chủ và các bản sao) khi thực hiện nhiều đợt cập nhật.
* Tất cả các cập nhật trong BATCH thuộc một khóa phân vùng nhất định được thực hiện nguyên tử.
* Theo mặc định, tất cả các hoạt động trong batch được thực hiện dưới dạng logged, để đảm bảo tất cả các đột biến cuối cùng sẽ hoàn thành (hoặc không có đột biến nào xảy ra)
* Lưu ý rằng:
* Các câu lệnh BATCH chỉ có thể chứa các câu lệnh UPDATE, INSERT và DELETE
* Các lô không phải là một tương tự đầy đủ cho các transaction SQL
* Nếu timestamp không được chỉ định cho mỗi thao tác, thì tất cả các thao tác sẽ được áp dụng với cùng một timestamp (một timestamp được tạo tự động hoặc timestamp được cung cấp ở batch level). Do thủ tục giải quyết xung đột của Scylla trong trường hợp ràng buộc timestamp, các hoạt động có thể được áp dụng theo thứ tự khác với thứ tự mà chúng được liệt kê trong câu lệnh BATCH. Để buộc một thứ tự thao tác cụ thể, bạn phải chỉ định timestamp cho mỗi thao tác.
* Một LOGGED batch vào một phân vùng duy nhất sẽ được chuyển đổi thành UNLOGGED batch như một tối ưu hoá

1. **Cơ chế phân tán**

Các định nghĩa lược đồ như keyspace và bảng được lưu trữ trong bảng hệ thống. Khi lược đồ được thay đổi, các nút trong một cụm được thông báo về những thay đổi thông qua cơ chế đẩy lược đồ. Nếu một nút bị lỗi trong quá trình thay đổi giản đồ, nó sẽ sử dụng cơ chế kéo giản đồ để đồng bộ hóa với phần còn lại của cụm khi khởi động.

### **6.1 Push**

Push xảy ra bất cứ khi nào một lược đồ được thay đổi (ví dụ CREATE KEYSPACE) thông qua trình quản lý di chuyển. Điều này làm cho các thay đổi lược đồ có thể nhìn thấy trên tất cả các nút có sẵn trong cụm. Nút cục bộ gửi các đột biến thay đổi giản đồ trong một DEFINITIONS\_UPDATE thông báo tới tất cả các nút thành viên đang hoạt động. Mỗi nút nhận được thông báo, áp dụng các đột biến thông qua hợp nhất giản đồ.

### **6.2 Pull**

Lược đồ pull được kích hoạt khi một người onAlive(), onChange(), onJoin() xử lý sự kiện trong StorageService được gọi.

Đây là chuỗi sự kiện trong Origin:

* Tại nút boostrap, MigrationManager.scheduleSchemaPull()được gọi.
* Các scheduleSchemaPull()chức năng nộp một MigrationTasklà sẽ gửi một MIGRATION\_REQUEST.
* Điểm cuối phản hồi bằng INTERNAL\_RESPONSE bao gồm danh sách các đột biến được trả về từ đó LegacySchemaTables.convertSchemaToMutations().
* Nút khởi động áp dụng các đột biến thông qua LegacySchemaTables.mergeSchema().

# PHẦN 2: HƯỚNG DẪN CÀI ĐẶT

1. **Cài đặt Docker**

* Link tải: [Tải Docker](https://desktop.docker.com/win/stable/Docker%20Desktop%20Installer.exe).
* Hướng dẫn cài đặt: [Cài đặt Docker](https://viblo.asia/p/cai-dat-docker-tren-windows-10-3Q75w6gelWb#_cai-docker-desktop-1).
* Cài đặt WLS 2: [Hướng dẫn](https://viblo.asia/p/cai-dat-docker-tren-windows-10-3Q75w6gelWb#_cai-dat-wsl-2-2)

1. **Cài đặt trên một máy**
   1. **Tải xuống Scylla**

Dùng lệnh ***docker pull scylladb/scylla*** để tải xuống Scylla phiên bản mới nhất:

Text

Description automatically generated

Đang tải xuống Scylla

Để kiểm tra đã tải xuống thành công hay chưa, ta mở Docker Desktop lên và vào mục Images:

Graphical user interface, text, application

Description automatically generated

Đã tải xuống thành công Scylla phiên bản mới nhất

* 1. **Cài đặt Scylla**

Dùng lệnh ***docker run --name scyllaU -d scylladb/scylla --overprovisioned 1 --smp 1*** để bắt đầu một node:

Thêm


Tạo 1 node có tên là ScyllaU

Sau khoảng 20 - 60 giây, ta kiểm tra xem node đã được tạo thành công hay chưa bằng lệnh ***docker exec -it scyllaU nodetool status***:

Text

Description automatically generated

Node ScyllaU đã được tạo thành công

* 1. **Nodetool status**

|  |  |
| --- | --- |
| Tham số | Mô tả |
| Datacenter | Trung tâm dữ liệu nắm giữ thông tin |
| Status | U: up, D: down |
| State | (N)ormal, (L)eaving , (J)oining or (M)oving |
| Address | Địa chỉ IP của node |
| Load | Kích thước trên đĩa mà dữ liệu Scylla có (cập nhật sau mỗi 60 giây) |
| Tokens | Số lượng mã thông báo trên mỗi nút |
| Owns | Hiển thị phần trăm dữ liệu do nút sở hữu (mỗi trung tâm dữ liệu) nhân với hệ số sao chép mà bạn đang sử dụng Ví dụ: nút có thể sở hữu 25% vòng lặp, nhưng hiển thị 100% nếu hệ số sao chép là 4 |
| Host ID | ID mạng máy chủ |
| Rack | Rack(giá đỡ) |

* 1. **Cấu hình Container**

Kiểm tra cấu hình Container, ta dùng lệnh ***docker stats scyllaU*** (scyllaU là tên container mà ta đã tạo ở phần 2.2) để giám sát theo thời gian thực một số đại lượng sử dụng bởi container:

Text

Description automatically generated

**Một số đại lượng sử dụng bởi container**

Trong đó:

|  |  |
| --- | --- |
| CONTAINER ID | ID của container |
| NAME | Tên của container |
| CPU % | Phần trăm CPU của máy chủ |
| MEM % | Bộ nhớ mà container đang sử dụng |
| MEM USAGE / LIMIT | Tổng bộ nhớ mà container đang sử dụng và tổng dung lượng bộ nhớ mà container được phép sử dụng |
| NET I/O | Lượng dữ liệu mà container đã gửi và nhận qua network interface của nó |
| BLOCK I/O | Lượng dữ liệu mà container đã đọc và ghi từ các block devices trên máy chủ |
| PIDs | Số lượng processes hoặc threads mà container đã tạo |

Chú giải các đại lượng

1. **Cài đặt trên cụm máy**

### **3.1 Cấu hình**

* Trong tệp **scylla.yaml**, chỉnh sửa các thông số được liệt kê bên dưới:
* **cluster\_name** - Đặt cluster\_name đã chọn
* **seeds** - Đặt nút đầu tiên và chỉ nút đầu tiên làm nút hạt giống. Chúng tôi sẽ thêm nhiều hạt giống sau trong quá trình này.
* **listen\_address** - Địa chỉ IP mà Scylla đã sử dụng để kết nối với các nút Scylla khác trong cụm
* **auto\_bootstrap** - Theo mặc định, tham số này được đặt thành true, nó cho phép các nút mới tự động di chuyển dữ liệu sang chính chúng.
* **endpoint\_snitch** - Đặt snitch đã chọn
* **rpc\_address** - Địa chỉ cho kết nối máy khách (Thrift, CQL)

Text

Description automatically generated

* Trong tệp **cassandra-rackdc.properties.dc1**, hãy chỉnh sửa các thông số được liệt kê bên dưới:
* **dc** - Đặt tên trung tâm dữ liệu
* **rack** - Đặt tên rack

Text

Description automatically generated

* Trong tệp **docker-compose.yml**, hãy chỉnh sửa các thông số được liệt kê bên dưới:

**image**: phiên bản scylla

* **--seeds SEEDS**: Các -seeds tùy chọn dòng lệnh cấu hình các node chính của Scylla. Nếu không có –seeds tùy chọn nào được chỉ định, Scylla sử dụng địa chỉ IP của chính nó làm seed.

Ví dụ, để cấu hình Scylla chạy với hai node seed 192.168.0.100 và 192.168.0.200: ***$ docker run --name some-scylla -d scylladb/scylla --seeds 192.168.0.100,192.168.0.200***

* **--listen-address ADDR**: Các --listen-address tùy chọn dòng lệnh cấu hình địa chỉ IP mà Scylla yêu cầu lắng nghe cho các kết nối từ nút Scylla khác.

Ví dụ, để cấu hình Scylla sử dụng địa chỉ lắng nghe 10.0.0.5:

***$ docker run --name some-scylla -d scylladb/scylla --listen-address 10.0.0.5***

* **--smp COUNT**: Các --smp tùy chọn dòng lệnh hạn chế Scylla để COUNTsố CPU. Tuy nhiên, tùy chọn này không yêu cầu một vị trí cụ thể của các CPU.

Ví dụ: để hạn chế Scylla ở 2 CPU: ***$ docker run --name some-scylla -d scylladb/scylla --smp 2***

* **--memory AMOUNT**: Các –memory tùy chọn dòng lệnh hạn chế Scylla để sử dụng lên đến AMOUNTbộ nhớ. Các AMOUNTgiá trị hỗ trợ cả Mđơn vị cho MB và Gđơn vị cho gigabyte.

Ví dụ: để hạn chế Scylla ở 4 GB bộ nhớ ta dùng câu lệnh: ***$ docker run --name some-scylla -d scylladb/scylla --memory 4G***

* **--overprovisioned ENABLE**: Các –overprovisioned tùy chọn dòng lệnh cho phép hoặc vô hiệu hóa tối ưu hóa để chạy Scylla trong một môi trường overprovisioned. Nếu không có –overprovisioned tùy chọn nào được chỉ định, Scylla sẽ mặc định chạy với tính năng tối ưu hóa được bật. Nếu –overprovisioned không được chỉ định và được để ở chế độ mặc định, chỉ định cpuset sẽ tự động vô hiệu hóa –overprovisioned.

Ví dụ: để kích hoạt tính năng tối ưu hóa để chạy trong môi trường được phân vùng tĩnh: ***$ docker run --name some-scylla -d scylladb/scylla --overprovisioned 0***

**--api\_address**: Địa chỉ cho các yêu cầu API REST.

Text

Description automatically generated

Mỗi services được cấu hình ban đầu sẽ là một node. Ví dụ:

Text

Description automatically generated

### **3.2 Cài đặt**

Đầu tiên chúng ta mở Terminal lên và cd vào tệp chứa các file cấu hình. Sau đó gõ dòng lệnh ***docker-compose up -d*** để tiến hành cài đặt:

Graphical user interface, text, application, chat or text message

Description automatically generated

Sau khoảng 20 – 60s, ta tiến hành kiểm tra các node đã join được vào datacenter hay chưa:

***docker exec -it vngroup\_congty01\_1 nodetool status***

Kết quả: Đã cài đặt thành công cụm máy (**NHOMCTY1**) Text

Description automatically generated

## **4. Cài đặt đa trung tâm dữ liệu**

### **4.1 Cấu hình**

Ta tạo thêm 1 file **docker-compose-dc2.yml** và tiến hành cấu hình với các node **congty03** và **congty04**:

Text

Description automatically generated

Tương tự ta tạo thêm file **cassandra-rackdc.properties.dc2** Text

Description automatically generated

### **4.2 Cài đặt**

Ta chạy câu lệnh ***docker-compose -f docker-compose-dc2.yml up -d*** để tiến hành cài đặt trung tâm dữ liệu thứ hai

Text

Description automatically generated

Sau khoảng 20 – 60s, ta tiến hành kiểm tra các node đã join được vào datacenter hay chưa:

***docker exec -it vngroup\_congty03\_1 nodetool status***

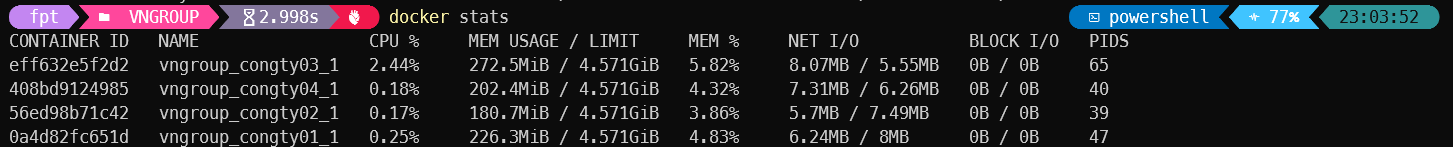
Kết quả: Đã cài đặt thành công cụm máy (**NHOMCTY2**)

Text

Description automatically generated

### **4.3 Cấu hình các node**

Chú thích về thông số: Xem phần 2.2 mục II



# PHẦN 3: THỰC NGHIỆM MÔ PHỎNG PHÂN TÁN

1. **Mô tả bài toán đặt ra với dữ liệu**

Một công ty muốn lưu thông tin và lương của từng nhân viên. Có hai hệ thống lưu trữ dữ liệu đang đặt tại 2 vị trí là NHOMCTY1 và NHOMCTY2. Tại NHOMCTY1 lưu thông tin chi tiết của từng nhân viên trong công ty và lưu thông tin lương của nhân viên. Và có 2 ứng dụng cần truy xuất thông tin như sau:

* Ứng dụng thứ nhất(tại các công ty con của NHOMCTY1) muốn xem trong công ty có bao nhiêu nhân viên nam/nữ, thêm, xoá và sửa thông tin của một số nhân viên trong công ty
* Ứng dụng thứ hai(tại các công ty con của NHOMCTY2) muốn xem những nhân viên nào có mức lương trên 10 triệu

1. **Mô tả cấu trúc dữ liệu sử dụng**

**2.1 Cụm**

Cụm không thực sự là một cấu trúc dữ liệu, một cụm là một tập hợp các nút Scylla hợp tác. Về mặt logic, một cụm chứa tất cả các dữ liệu. Về mặt vật lý, dữ liệu được tải rộng trên các mảnh vỡ. Có 2 thành phần cụm:

* Nút.
* Keyspaces: mỗi cụm quản lý một số không gian chính.

**2.2 Nút**

Một nút có tương ứng 1: 1 với máy chủ. Nó có một địa chỉ IP mà nó được biết đến. Nó là một đơn vị của sự thất bại - một nút có thể bị lỗi và phục hồi, được đưa vào ngoại tuyến và đưa trở lại ngoại tuyến. Một nút chứa các mảnh vỡ và bộ nhớ vật lý.

* Shards: mỗi lõi trong một nút được đại diện bởi một mảnh

**2.3 Mảnh vỡ**

Một mảnh vỡ có sự tương ứng 1: 1 với lõi xử lý. Mỗi mảnh chịu trách nhiệm cho một số vnodes. Các thành phần mảnh gồm:

* Máy chủ tiết kiệm: lắng nghe các kết nối trên giao thức tiết kiệm.
* Máy chủ gốc: nghe kết nối trên giao thức gốc.
* Siêu dữ liệu: bản sao của tất cả các siêu dữ liệu trong hệ thống về các không gian và bảng khóa (tên và cấu trúc của chúng).
* Memtables: đang chờ dữ liệu bẩn trong phần của không gian chính (tập hợp các vnodes) thuộc về mảnh vỡ.
* Vòng: mô tả về cấu trúc liên cụm: danh sách các nút, mối quan hệ vnode <-> node, mối quan hệ không gian vnode <-> key.
* Commitlog: tệp đĩa chứa nhật ký dữ liệu trong memtables.
* Sstables.

**2.4 Bảng ghi nhớ**

1. **Các bước thực nghiệm chi tiết**

### **3.1 Tạo dữ liệu cho máy**

Đầu tiên chúng ta sẽ chọn node congty01 để tạo dữ liệu ở đây. Trong scylla có một CLI là cqlsh: ***docker exec -it vngroup\_congty01\_1 cqlsh***

Text

Description automatically generated

Tạo 1 keyspace:



Truy cập vào keyspace:

Graphical user interface, text, application

Description automatically generated with medium confidence

Tạo bảng employee:

Text

Description automatically generated

Tiến hành đổ dữ liệu vào bảng:

Text

Description automatically generated

### **3.2 Truy vấn dữ liệu ở các máy trong 1 cụm**

#### 3.2.1 Truy vấn tại máy tạo dữ liệu (congty01)

Graphical user interface, text, application

Description automatically generated

#### 3.2.2 Truy vấn tại các máy khác (congty02)

Ta vào cqlsh của node congty02: ***docker exec -it vngroup\_congty02\_1 cqlsh***

Graphical user interface, text

Description automatically generated

Vào catalog và tiến hành truy vấn:

Graphical user interface, text, application, chat or text message

Description automatically generated

Truy vấn thành công.

### **3.3 Phân tán dữ liệu ra các datacenter khác**

Để chắc chắn không có sai sót, ta sẽ tạo lại một dữ liệu khác ở datacenter

NHOMCTY1:

***docker exec -it vngroup\_congty01\_1 cqlsh***

Tạo keyspace mới:

***CREATE KEYSPACE tracking WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy','NHOMCTY1' : 3};***

Vào tracking:

***use tracking;***

Tạo bảng:

Text

Description automatically generated

Insert dữ liệu:

Text

Description automatically generated

### **3.4 Configuring keyspaces cho Multi-DC**

Bây giờ cả hai cụm đều đang hoạt động, chúng tôi có thể bắt đầu chuyển đổi các keyspace hiện có để tồn tại trên cả hai trung tâm dữ liệu, vì vậy nếu một kẻ đột biến tấn công trung tâm dữ liệu đầu tiên, chúng tôi sẽ không bị mất dữ liệu. Các cqlsh tiện ích này sẽ cho phép chúng tôi thực hiện những thay đổi này với “**ALTER KEYSPACE**”. Bắt đầu bằng cách đăng nhập vào node đầu tiên (**congty01**) và chuyển đổi các keyspace hiện có có tên là **catalog** và **tracking**.

Trở lại cqlsh congty01: ***docker exec -it vngroup\_congty01\_1 cqlsh***

***ALTER KEYSPACE catalog WITH REPLICATION = {'class': 'NetworkTopologyStrategy', 'NHOMCTY1':3, 'NHOMCTY2':3};***

******

***ALTER KEYSPACE tracking WITH REPLICATION = {'class': 'NetworkTopologyStrategy', 'NHOMCTY1':3, 'NHOMCTY2':3};***

******

Ngoài ra, chúng ta cần chuyển đổi hệ thông keyspace:

***ALTER KEYSPACE system\_auth WITH replication = { 'class' : 'NetworkTopologyStrategy', 'NHOMCTY1' : 3, 'NHOMCTY2' : 3};***

******

***ALTER KEYSPACE system\_distributed WITH replication = { 'class' : 'NetworkTopologyStrategy', 'NHOMCTY1' : 3, 'NHOMCTY2' : 3};***

******

***ALTER KEYSPACE system\_traces WITH replication = { 'class' : 'NetworkTopologyStrategy', 'NHOMCTY1' : 3, 'NHOMCTY2' : 3};***

******

Các keyspace hiện đã sẵn sàng cho đa trung tâm dữ liệu, vì chúng ta cần đảm bảo dữ liệu nhất quán và đồng bộ trong mỗi trung tâm dữ liệu. Chúng ta có thể đạt được tính nhất quán của dữ liệu bằng cách chạy lệnh “ nodetool rebuild” trên mỗi nút trong trung tâm dữ liệu thứ hai như được hiển thị bên dưới. Lệnh này xây dựng lại dữ liệu của một nút bằng cách truyền dữ liệu từ các nút khác trong cụm.

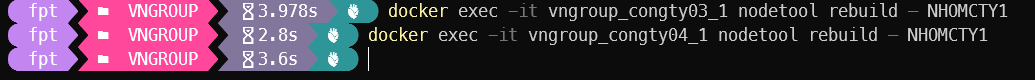
Thoát cqlsh của node congty01



Rebuild node congty03 và node congty04:

***docker exec -it vngroup\_congty03\_1 nodetool rebuild – NHOMCTY1***

***docker exec -it vngroup\_congty04\_1 nodetool rebuild – NHOMCTY1***

******

Sau khi lệnh đó được chạy, dữ liệu từ trung tâm dữ liệu đầu tiên sẽ được truyền sang trung tâm dữ liệu thứ hai và không có đầu ra nào được hiển thị từ thiết bị đầu cuối.

Để đảm bảo rằng các keyspace của chúng ta có thể truy cập được từ trung tâm dữ liệu thứ hai, chạy các lệnh sau:

***docker exec -it vngroup\_congty03\_1 cqlsh***

Text

Description automatically generated

***describe catalog;***

***Text

Description automatically generated***

***describe tracking;***

***Text

Description automatically generated***

### **3.5 Truy vấn dữ liệu ở các máy trong các datacenter**

Bây giờ chúng ta có thể truy cập dữ liệu ở bất kì node nào trong 2 datacenter NHOMCTY1 và NHOMCTY2.

Ví dụ ở node congty03:

Keyspace: catalog

Text

Description automatically generated

Keyspace: tracking

Graphical user interface, text, application

Description automatically generated

Tương tự, hãy thử truy cập dữ liệu ở node congty04:

Graphical user interface, text

Description automatically generated