*Phạm Nhân*

*There are only two hard things in Computer Science: cache invalidation and naming things*.-- Phil Karlton

# **I: Tổng quan về Cache**

Cache là một vùng đệm trong bộ nhớ máy tính để lưu trữ dữ liệu tạm thời, nhằm mục đích tăng tốc độ truy cập và giảm thời gian xử lý cho các yêu cầu sau này. Cache hoạt động như một lớp đệm giữa CPU và bộ nhớ chính, giúp giảm thiểu số lần truy cập vào bộ nhớ chậm hơn.

A diagram of a computer

Description automatically generated

*Mục tiêu chính của caching là để tăng tốc độ xử lý. Thay vì lặp đi lặp lại truy xuất dữ liệu từ nguồn gốc ban đầu (có thể chậm hoặc tốn tài nguyên), ứng dụng có thể lấy dữ liệu từ bộ nhớ cache gần đó. Việc truy xuất nhanh chóng này giúp tiết kiệm thời gian và tài nguyên, làm cho toàn bộ quá trình trở nên hiệu quả hơn.*

**Các đặc điểm của cache:**

* **Tốc độ truy cập nhanh:** Cache thường được lưu trữ trong RAM, cho phép truy cập dữ liệu nhanh chóng (sub-millisecond)
* **Dung lượng giới hạn:** Cache chỉ có thể lưu trữ một phần nhỏ của dữ liệu toàn cục, thường là dữ liệu thường xuyên truy cập
* **Thời gian tồn tại ngắn:** Dữ liệu trong cache thuần thường có thời gian sống ngắn, cần được cập nhật liên tục
* **Không bền vững**: Dữ liệu trong cache thuần sẽ bị mất khi máy tính tắt hoặc khởi động lại

**Phân loại:**

* Cache trong bộ nhớ (RAM cache): Đây là loại cache phổ biến nhất, sử dụng bộ nhớ RAM để lưu trữ dữ liệu.
* Cache đệm (Buffer cache): Lưu trữ dữ liệu tạm thời trước khi được viết vào đĩa.
* Cache đệm mạng (Network buffer cache): Lưu trữ dữ liệu tạm thời trong quá trình truyền tải dữ liệu qua mạng.
* Cache GPU: Sử dụng bộ nhớ video card để tăng tốc độ xử lý đồ họa.
* Cache đệm trang (Page cache): Lưu trữ các trang web đã truy cập gần đây để giảm thời gian tải lại.

# **II: Memory Cache – RAM Cache**

## **1: Khái niệm**

A yellow background with arrows and a rectangular sign

Description automatically generated

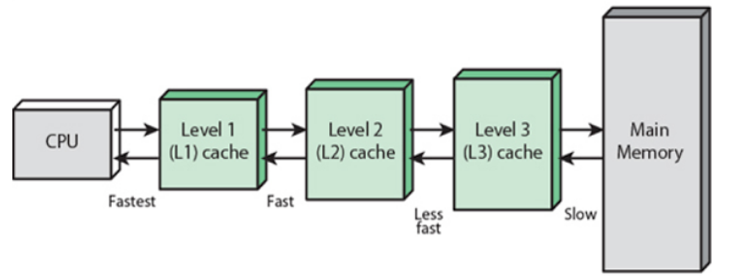
Memory Cache chỉ đơn giản là việc tổ chức và lấy dữ liệu trên bộ nhớ chính (RAM) vì tốc độ của bộ nhớ này rất nhanh, dữ liệu không có trên bộ nhớ sẽ được lấy từ cơ sở dữ liệu và dữ liệu trên bộ nhớ khi thay đổi cũng sẽ được đồng bộ lại với cơ sở dữ liệu.

Cách thức tổ chức dữ liệu của memory cache: Dữ liệu được lưu trên cache thường ở dạng key-value, cả key và value thường sẽ ở dạng byte array .

Tại sao không dưới dạng đối tượng mà lại lưu ở dạng byte array?

* Lưu dữ liệu ở dạng đối tượng sẽ tốn bộ nhớ: một con trỏ trong java cũng tốn 8 bytes rồi vậy nếu chúng ta chuyển từ byte array sang đối tượng, có thể chúng ta sẽ phải sử dụng rất nhiều con trỏ. Thêm vào nữa khi lưu dữ liệu ở dạng byte array chúng ta sẽ có thể tận được sức mạnh của các giao thức như Protobuf hay MsgPack để giảm được kích thước của dữ liệu, từ đó tiết kiệm được rất nhiều khi lưu ở bộ nhớ
* Lưu đối tượng sẽ làm giảm hiệu năng của cache server: Muốn lưu được dạng đối tượng, chúng ta sẽ phải trải qua các bước serialize và deserialize điều này sẽ rất tốn thời gian xử lý cho server, tốt nhất việc đó hãy để client lo.

## **2: Phân loại bộ nhớ đệm**



* **Level 1:** Mức đầu tiền hay còn gọi là primary cache. Với loại cache này sẽ được lưu vào chính bản thân cpu nên có tốc độ tưởng đương tốc độ cpu. Ví dụ như cpu có 4 core thì mỗi core sẽ có 1 bộ đệm riêng. Kích thước bộ nhớ nằm khoảng 2KB đến 64KB
* **Level 2:** mức thứ hai hay còn gọi là Secondary Cache. Bộ nhớ cấp 2 này có thể nằm bên trong cpu hoặc ngoài cpu. Tất cả các core của cpu có thể có bộ nhớ đệm cấp 2 riêng biệt hoặc chia sẻ bộ nhớ đệm cấp 2 với nhau. Trong trường hợp nằm ngoài cpu thì nó được kết nối với cpu qua 1 bus có tốc độ rất cao. Kích thước bộ nhớ đệm này khoảng 256KB đến 512KB. Về tốc độ chúng chậm hơn bộ nhớ đệm L1
* **Level 3:** main cache. Không phải tất cả bộ xử lý đều có bộ nhớ đệm này. Một số high processor có thể có loại này để năng cao hiệu suất bộ nhớ đệm cấp 1 và cấp 2. Nằm ngoài cpu và chia sẻ giữa các core của cpu. Dung lượng từ 1MB tới 8MB. Mặc dù chậm hơn L1 và L2 nhưng nó nhanh hơn bộ nhớ RAM

## **3: Cách CPU tìm dữ liệu trong cache**

A diagram of a computer system

Description automatically generated

**B1:** Khi CPU cần dữ liệu, trước hết, nó tìm bên trong bộ đệm L1. Nếu nó không tìm thấy gì trong L1, nó sẽ tìm bên trong bộ đệm L2.

**B2:** Nếu một lần nữa, nó không tìm thấy dữ liệu trong bộ đệm L2, nó sẽ tìm trong bộ đệm L3.

**B3:** Nếu dữ liệu được tìm thấy trong bộ nhớ đệm, thì nó được gọi là lần truy cập bộ nhớ cache.

**B4:** Ngược lại, nếu dữ liệu không được tìm thấy bên trong bộ nhớ cache, nó được gọi là lỗi bộ nhớ cache. Nếu dữ liệu không có sẵn trong bất kỳ bộ nhớ đệm nào, nó sẽ nằm trong Bộ nhớ truy cập ngẫu nhiên (RAM).

**B5:** Nếu RAM cũng không có dữ liệu, thì nó sẽ lấy dữ liệu đó từ Ổ đĩa cứng.

## **4: Memory cache trong Java**

### **4.1 Đọc, ghi cache với java core**

***Java Core*** *yêu cầu tự triển khai và quản lý cache, bao gồm cấu hình, lưu trữ, và truy xuất.*

**a, Đọc dữ liệu**

A diagram of a computer

Description automatically generated

**Bước 1.** Client sẽ serialize key thành byte array

**Bước 2.** Client sẽ gọi đến cache với key byte array này để lấy giá trị (value)

**Bước 3.** Cache sẽ sẽ kiểm tra trong map trên bộ có nhớ có giá trị không, nếu không có, nó sẽ lục tìm ở trong database và ném dữ liệu vào bộ nhớ, lần kế tiếp nó sẽ không cần phải truy cập vào db nữa.

**Bước 4.** Cache sẽ trả lại dữ liệu cho client, tất cả ở dạng byte array.

**Bước 5.** Client sẽ deserialize byte array thành đối tượng tương ứng cho client.

* Note : Quá trình đọc dữ liệu sẽ sử dụng chung một luồng để tránh tình trạng không đồng bộ

**b, Ghi dữ liệu**

A diagram of a computer

Description automatically generated with medium confidence

**Bước 1.** Client sẽ serialize cả key và value thành dạng byte array và gửi đến cache

**Bước 2.** Cache sẽ lưu chính xác các byte array này vào bộ nhớ, sau đó nó sẽ ném các byte array này vào một queue để cho luồng khác thực hiện việc ghi vào database, điều này cho phép cache nhanh chóng được giải phóng để trả về kết quả cho client

**Bước 3**. Cache sẽ trả lại kết quả cho client.

**Bước 4.** Client kết thúc việc gọi cache và làm việc khác

**Bước 5.** Song song với bước 3 và 4 tầng persistent cũng sẽ gom lại các câu lệnh update thành một bó và lưu xuống database để làm tăng hiệu năng.

### **4.2 Đọc ghi dữ liệu với Java Framework ( Spring & Spring Boot)**

Spring sử dụng **Spring Cache** để quản lý cache. Đây là một cơ chế quản lý cache được tích hợp sẵn và dễ sử dụng nhờ các annotation và khả năng hỗ trợ nhiều thư việc cache như Redis, Jcache, Caffeine, …

**Một số anotation chính như :**

* EnableCaching: Bật tính năng caching cho ứng dụng.
* @Cacheable: Lưu kết quả của phương thức vào cache trước khi thực thi.
* @CachePut: Cập nhật cache trước khi thực thi phương thức .
* @CacheEvict: Xóa dữ liệu từ cache.
* @Caching: Cho phép sử dụng nhiều annotation caching cùng lúc trên một phương thức.
* @CacheConfig: Định nghĩa các thuộc tính chung cho các phương thức được đánh dấu với annotations caching

**Spring cache hoạt động như thế nào :**

1. Khi một phương thức được đánh dấu @Cacheable được gọi:

* Spring đầu tiên kiểm tra cache trước khi thực thi phương thức.
* Nếu có kết quả trong cache, nó trả về ngay lập tức mà không gọi phương thức.
* Nếu không có trong cache, nó thực thi phương thức và lưu kết quả vào cache.

1. Các annotation khác như @CachePut và @CacheEvict cũng hoạt động tương tự, nhưng với các hành động khác nhau (cập nhật hoặc xóa cache).
2. Spring Boot tự động cấu hình CacheManager và cache sto re dựa trên thư viện cache được chọn.

# **III: Redis Cache**

## **1: Tổng quan về Redis Cache**

### **1.1 Tại sao Redis ra đời ?**

Như đã tìm hiểu ở trên, Memory Cache thường được lưu trữ trên RAM của server. Tuy nhiên, khi ứng dụng ngày càng lớn và cơ sở dữ liệu tăng trưởng nhanh, kích thước của bộ nhớ RAM sẽ trở thành một giới hạn lớn. Điều này khiến việc sử dụng cache trở nên khó khăn, vì RAM không đủ dung lượng để lưu trữ toàn bộ dữ liệu. Để giải quyết vấn đề này, chúng ta cần bổ sung thêm server để mở rộng bộ nhớ. Đây chính là lý do Redis ra đời, nhằm cung cấp một giải pháp quản lý bộ nhớ cache hiệu quả, có khả năng mở rộng dễ dàng trên nhiều server.

Có một câu hỏi như sau: “ *Dùng thêm server để quản lý cache nhưng tại sao không tăng dung lượng RAM lên ?”*

Có thể trả lời câu hỏi trên như sau :

**1.** Bộ nhớ RAM có giới hạn vật lý

**2.** Chi phí nâng cấp không hiệu quả

**3.** Đảm bảo tính khả dụng (High Availability)

**4.** Khi chỉ sử dụng 1 server với dung lượng RAM lớn, nếu server gặp sự cố khiến cho toàn bộ hệ thống sẽ ngừng hoạt động. Điều này gây rủi ro rất lớn cho các ứng dụng quan trọng.

* Với nhiều server (distributed system), Redis có thể sử dụng **Redis Cluster** để phân chia dữ liệu trên các server khác nhau.
* Nếu một server bị lỗi, dữ liệu vẫn có thể được truy cập từ các server còn lại mà không làm gián đoạn hoạt động.

**5.** Tăng khả năng mở rộng theo chiều ngang (Scalability) – Khi nâng cấp RAM đến một mức nào đó sẽ không thể tiếp tục nâng cấp thêm được nữa.

* **Thêm server** (scale-out) giúp bạn mở rộng hệ thống gần như vô hạn. Mỗi server bổ sung sẽ tăng dung lượng bộ nhớ và năng lực xử lý của toàn hệ thống. Redis hỗ trợ điều này thông qua **sharding** hoặc **Redis Cluster**, giúp phân phối dữ liệu trên nhiều server một cách tự động.

**6.** Phân phối tải (Load Balancing)

* Khi ứng dụng có nhiều người dùng hoặc truy cập lớn, một server duy nhất dù có nhiều RAM cũng sẽ gặp giới hạn về hiệu suất. Bằng cách sử dụng nhiều server, Redis có thể phân phối tải giữa các server, đảm bảo hệ thống hoạt động mượt mà hơn.

**7.** ….. hehee

### **1.2 Redis quản lý cache như thế nào ?**

**Cấu trúc dữ liệu:**

* Mỗi giá trị trong Redis được lưu trữ dưới dạng key-value pair.
* Keys là duy nhất và được sử dụng để truy cập giá trị tương ứng.
* Values có thể là chuỗi, số nguyên, danh sách, set, map hoặc các cấu trúc dữ liệu phức tạp hơn.

**Caching cơ bản:**

1. Khi một ứng dụng muốn lưu trữ dữ liệu trong Redis:

* Ứng dụng gửi một request đến Redis với key và value cần lưu.
* Redis lưu trữ cặp key-value vào bộ nhớ.

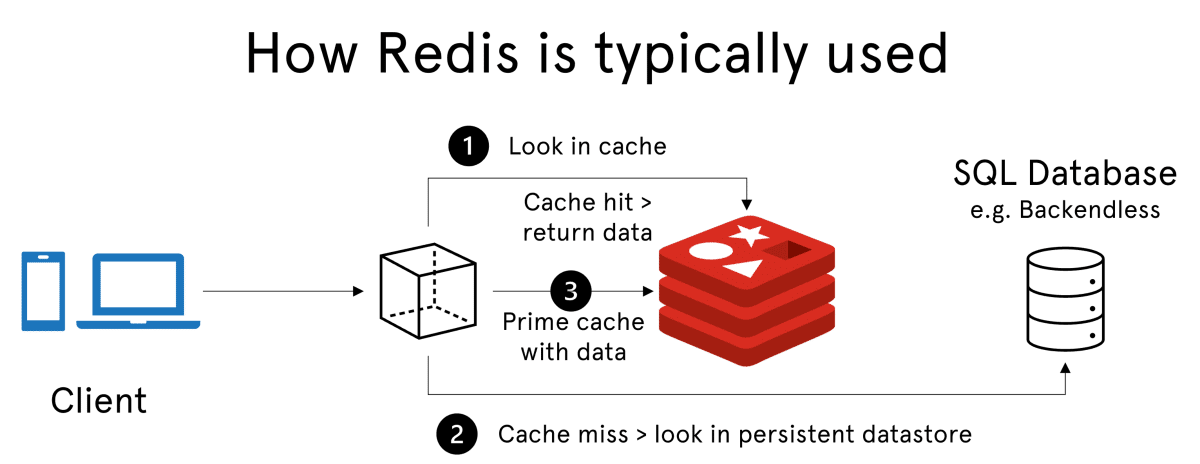
1. Khi ứng dụng muốn lấy lại dữ liệu từ Redis:

* Ứng dụng gửi một request với key đã biết.
* Redis trả về value tương ứng với key đó.

**Các tính năng quan trọng**

Redis cung cấp nhiều tính năng giúp quản lý cache hiệu quả:

* TTL (Time To Live): Redis có thể tự động xóa các key sau một khoảng thời gian nhất định.
* Expiration: Các giá trị trong Redis có thể tự động hết hạn và bị xóa.
* Persistence: Dữ liệu trong Redis có thể được lưu trữ vĩnh viễn trên đĩa để phục hồi sau khi restart.



Đây là một mô hình quản lý cache đơn giản của redis:

**B1:** Client sẽ gọi tới thông qua giao thức TCP tới redis bằng key (Cache look)

**B2:** Nếu dữ liệu đã tồn tại trên Redis cache (cache hit) -> Redis trả kết quả về cho hệ thống một cách nhanh chóng mà không cần truy cập vào cơ sở dữ liệu chính.Nếu không thực hiện **B3.**

**B3:** Truy cập cơ sở dữ liệu chính (nếu không có trong cache – Cache miss) sau đó cập nhật dữ liệu vừa truy nhập ở cơ sở dữ liệu vào Redis Cache

### **1.3 Luồng thực hiện của Redis cache**

Mục này sẽ tìm hiểu thêm rồi bổ sung sau

## **2: Các mô hình Redis** **Replication/ Cluster / HA**

### **1. Mô hình đơn:**

*Mô hình đơn của Redis là phiên bản cơ bản nhất của Redis, hoạt động như một hệ thống đơn lẻ trên một máy chủ. Đây là mô hình cơ bản và phổ biến nhất của Redis.* A diagram of a api

Description automatically generated

Dưới đây là chi tiết về cách thức hoạt động của mô hình đơn Redis

Cách thức hoạt động

1. Server:
   * Redis server chạy như một tiến trình độc lập trên m ột máy chủ.
   * Nó lắng nghe trên cổng mặc định là 6379 cho giao tiếp TCP và 6379 (đã được mở rộng) cho giao tiếp Unix socket.
2. Client:
   * Các client kết nối với Redis server thông qua TCP hoặc Unix socket.
   * Client gửi các lệnh (commands) đến server để đọc, viết hoặc xóa dữ liệu.
3. Cấu trúc dữ liệu:
   * Redis sử dụng cấu trúc dữ liệu trong bộ nhớ RAM.
   * Dữ liệu được tổ chức thành các key-value pairs.
4. Persistence:
   * Mô hình đơn có thể được cấu hình để persist dữ liệu lên disk.
   * Có hai cách persistence chính: AOF (Append Only File) và RDB (Redis Database).
5. Sự đồng bộ:
   * Mô hình đơn hoạt động như một hệ thống độc lập.
   * Không có sự đồng bộ giữa các instance khác.

Ưu điểm

1. Dễ dàng triển khai:
   * Chỉ cần cài đặt và chạy một instance Redis trên một máy chủ.
2. Hiệu suất cao:
   * Tốc độ truy cập cực nhanh vì dữ liệu được lưu trong RAM.
3. Tương thích:
   * Phổ biến và tương thích với nhiều ngôn ngữ lập trình.
4. Phức tạp thấp:
   * Dễ quản lý và duy trì so với mô hình phức tạp hơn.

Nhược điểm

1. Hạn chế khả năng mở rộng:
   * Không thể phân phối tải hoặc tăng dung lượng bằng cách thêm node mới.
2. Độ tin cậy thấp:
   * Nếu máy chủ gặp lỗi, toàn bộ dữ liệu sẽ bị mất.
3. Tính sẵn sàng:
   * Chỉ có sẵn sàng trong trường hợp không có lỗi nào xảy ra với máy chủ.

Cấu hình và quản lý

1. Cấu hình:
   * Có thể được thực hiện thông qua file redis.conf hoặc các lệnh command-line.
2. Persistence:
   * Có thể được cấu hình để tự động persist dữ liệu lên disk tại các thời điểm nhất định hoặc khi Redis bị tắt.
3. Backup:
   * Cần thực hiện backup thủ công của file RDB hoặc AOF để bảo vệ dữ liệu.

Sử dụng trong môi trường đơn lẻ

Mô hình đơn Redis phù hợp cho nhiều ứng dụng nhỏ đến trung bình, đặc biệt là trong môi trường phát triển hoặc thử nghiệm. Nó cung cấp một giải pháp đơn giản và hiệu quả để lưu trữ và truy cập dữ liệu nhanh chóng.

Tuy nhiên, khi ứng dụng lớn hơn hoặc cần khả năng mở rộng cao hơn, người dùng thường chuyển sang mô hình cluster của Redis để phân phối tải và tăng độ tin cậy.

*Phân biệt:*

*1) Replication: 1master-1slave hoặc 1master-nhiều-Slave (mỗi node sẽ chứa đủ 100% dữ liệu).*

*2) Sharding cluster: Partition data (data được chia lẻ và lưu trên nhiều node khác nhau, tổng dữ liệu riêng rẽ các node = 100%).*

*>> Khi nào ta nên chọn giữa replication / cluster sharding?:*

*- Về cơ bản, mô hình (1Master)- (1Slave) là đủ hoàn toàn để redis chạy hết hiệu năng cho cả 1 hệ thống lớn hàng trăm nghìn truy vấn trên 1s (bank/viễn thông...).*

*- Vấn đề nằm ở chỗ khi lượng Ram cần dùng để lưu trữ Data của redis > vượt quá Ram của Server đang chạy. Khi đó ta sẽ bắt đầu chia nhỏ dữ liệu ra để lưu ở "nhiều server" khác nhau. Khi client gọi vào, redis-cluster sẽ hướng dẫn truy xuất vào chính xác node nào có dữ liệu.*

### **2. Mô hình Master-Slave hoặc 1 Master và nhiều SLAVE**

A diagram of a diagram of a red cube with white text

Description automatically generated

*Ưu điểm:*

*- Mô hình 1Master-1Slave hoặc 1Master-(N)Slave sẽ đảm bảo dữ liệu luôn luôn được dự phòng.*

*- Khi xảy ra sự cố với node Master, ta sẽ manual Slave node thay làm Master.*

*- Có thể cho node M làm write, và các node S làm read, tăng khả năng chia tải.*

*Nhược điểm:*

*- Khi Master chết, phải cấu hình thủ công Slave lên làm Master. Và phải tự động chuyển luồng cho client gọi vào M mới.*

*- Tối ưu hơn, ta có thể có mô hình Setinel tự động detect Master down và đẩy Slave node khác lên làm Master ở mục C.*

*- Vẫn sẽ có độ trễ về đồng bộ thông tin từ M > S. Ví dụ ta HMSET hàng triệu key có độ dài lớn vào M.*

Mô hình Master-Slave trong Redis là một cấu trúc dữ liệu sao chép, nơi có một máy chủ chính (Master) và nhiều máy chủ phụ (Slave).

* Master xử lý các yêu cầu đọc/ghi từ phía khách hàng
* Slave sao chép dữ liệu từ Master để tăng khả năng sẵn sàng và hiệu suất

Cách hoạt động

1. Master duy trì bản sao của dữ liệu và xử lý các yêu cầu từ khách hàng.
2. Khi Master nhận được một thay đổi dữ liệu, nó sẽ sao chép thay đổi đó đến một hoặc nhiều Slave.
3. Slave liên tục kết nối với Master và cập nhật dữ liệu từ đó.
4. Nếu Master bị lỗi, một trong các Slave sẽ được nâng lên thành Master mới.

Ưu điểm

* Tăng khả năng sẵn sàng và hiệu suất bằng cách phân phối dữ liệu
* Cải thiện khả năng phục hồi khi có lỗi
* Cho phép thực hiện các tác vụ đọc trên Slave mà không ảnh hưởng đến Master

Lưu ý

* Mặc định Redis sử dụng replication không đồng bộ (asynchronous)
* Có thể cấu hình replication đồng bộ bằng cách sử dụng lệnh WAIT của Redis
* Cần cấu hình persistence trên cả Master và Slave để đảm bảo tính toàn vẹn dữ liệu

Kết luận

Mô hình Master-Slave là một cách hiệu quả để tăng khả năng sẵn sàng và hiệu suất của Redis, đặc biệt phù hợp cho các ứng dụng cần độ khả dụng cao và khả năng mở rộng. Việc cấu hình và quản lý mô hình này đòi hỏi sự cân nhắc kỹ lưỡng về hiệu suất và tính toàn vẹn dữ liệu.

### **3. Mô hình Sentinel**

A diagram of a diagram of a diagram

Description automatically generated  
  
  
*Ưu điểm:*

*- Mô hình Setinel đã tối ưu ở việc "bầu chọn" đâu sẽ là Master node khi có node bị chết*

*- Nhưng app không biết đâu là Master mới để gọi vào khi Master bị thay đổi.*

*Để khắc phục vấn đề này, các siêu nhân khác đã đưa ra phương án dùng HA-Proxy để phát hiện và lái luồng TCP về redis master.*

*Hoặc dùng thư viện client sẵn có (như java jedis) có thể tự detect được đâu là M trong khối Setinel (không cần cài HAProxy)*

*Nhược điểm:*

*- Cần nhiều tài nguyên (cần ít nhất là 3 node để tránh bị tình trạng bầu chọn không đồng đều, các slave-setinel tự nhận mình là master - hiện tượng Split-Brain)*

SETINEL là một hệ thống giám sát tự động do Redis cung cấp để thực hiện failover tự động giữa các instance Redis master-slave. Nó bao gồm:

* Một nhóm các instance Sentinel làm việc cùng nhau
* Các instance Redis master và slave

Cách hoạt động

1. Các Sentinel giám sát tình trạng của các instance Redis đang theo dõi.
2. Khi master gặp lỗi, các Sentinel sẽ tiến hành bầu cử để chọn một slave mới làm master.
3. Quá trình bầu cử dựa trên các tiêu chí:
   * Reachability của slave
   * Thời gian downtime của slave
   * Priority của slave (nếu được thiết lập)
   * Offset của slave (tương đương với dữ liệu đã cập nhật)
   * ID của quá trình
4. Sau khi bầu cử xong, Sentinel sẽ cập nhật cấu hình và thông báo cho các client.
5. Các client có thể tự động kết nối đến instance mới được bầu làm master.

Ưu điểm

* Tự động failover mà không cần can thiệp thủ công
* Cung cấp khả năng phục hồi cao
* Tự động phát hiện và điều hướng client đến instance master mới

Cấu hình

Để thiết lập SETINEL:

1. Thiết lập các instance Redis master-slave ban đầu.
2. Thiết lập các instance Sentinel với cấu hình:

* port <port>
* sentinel monitor <group-name> <ip> <port> <quorum>
* sentinel down-after-milliseconds <group-name> <milliseconds>
* sentinel parallel-syncs <group-name> <count>
* sentinel failover-timeout <group-name> <milliseconds>

1. Khởi động các instance Sentinel.
2. SETINEL sẽ tự động phát hiện và theo dõi các instance Redis.

Kết luận

SETINEL là một giải pháp tự động hóa failover hiệu quả cho Redis, giúp tăng độ khả dụng và giảm thiểu thời gian dừng dịch vụ khi xảy ra lỗi. Tuy nhiên, nó cũng đòi hỏi sự hiểu biết về cấu hình và quản lý để sử dụng hiệu quả.

**4. Mô hình Sharding Cluster**

A diagram of a diagram of a api

Description automatically generated with medium confidence

*Ưu điểm:*

*- Khắc phục được yếu điểm các mô hình trên là băm nhỏ dữ liệu sang các node.*

*- Đảm bảo downtime gần như zero. Dữ liệu luôn được đảm bảo 100% (gồm 3 node chính và 3 node phụ)*

*- Dễ dàng mở rộng, chỉ cần gõ lệnh chia dữ liệu sang các node mới*

*Nhược Điểm:*

*- Cấu hình phức tạp, cần code client phải hỗ trợ cluster.*

*- Chỉ chạy trên db0 (không hỗ trợ multi db)*

Sharding Cluster là một phương pháp phân tán dữ liệu trong Redis, nơi dữ liệu được chia thành các mảnh (shards) và phân phối đều trên nhiều node Redis.

Cách hoạt động

1. Datenospace được chia thành các slot (khoảng 16384 slot).
2. Mỗi key được map vào một slot dựa trên hash của key.
3. Các slot được phân phối đều trên các node trong cluster.
4. Các node trong cluster tự động cân bằng dữ liệu khi thêm hoặc loại bỏ node.

Ưu điểm:

1. High performance: Cung cấp hiệu suất tương tự như Redis standalone.
2. High availability: Hỗ trợ cấu hình master-replica để đảm bảo tính sẵn sàng cao.
3. Horizontal scalability: Có thể dễ dàng thêm hoặc loại bỏ node mà không gây downtime.
4. Native solution: Không cần sử dụng proxy hay công cụ bên ngoài.
5. Compatibility: Almost completely compatible with standalone Redis.

Nhược điểm:

1. Requires client support: Cần cập nhật client để hỗ trợ cluster.
2. Limited multi-key operations: Chỉ hỗ trợ các toán tử nhiều key thuộc cùng một slot.
3. Only one database: Chỉ hỗ trợ database 0, không hỗ trợ SELECT command.

Kết luận

Sharding Cluster là một giải pháp phân tán dữ liệu hiệu quả cho Redis, cung cấp khả năng mở rộng, độ sẵn sàng cao và hiệu suất tốt. Tuy nhiên, nó cũng đòi hỏi sự đầu tư vào việc cập nhật client và giới hạn về khả năng sử dụng multiple databases. Việc lựa chọn mô hình này phụ thuộc vào nhu cầu cụ thể của dự án về khả năng mở rộng và độ sẵn sàng.