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.

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

Redis (Remote Dictionary Server) là một hệ thống lưu trữ dữ liệu in-memory, open-source, được sử dụng để lưu trữ dữ liệu, cache, message broker và queue. Nó được viết tắt từ Remote Dictionary Server.

**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 SETINEL**

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.

1. **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.