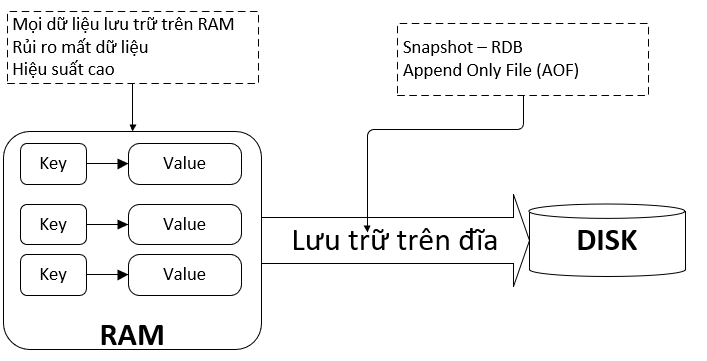
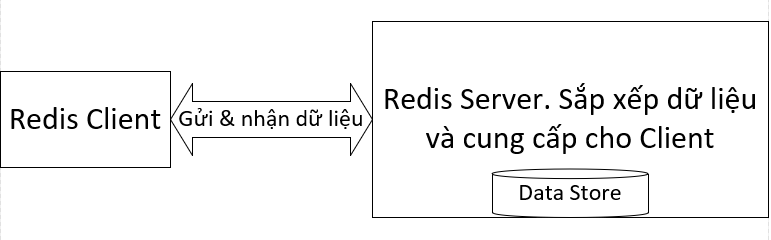
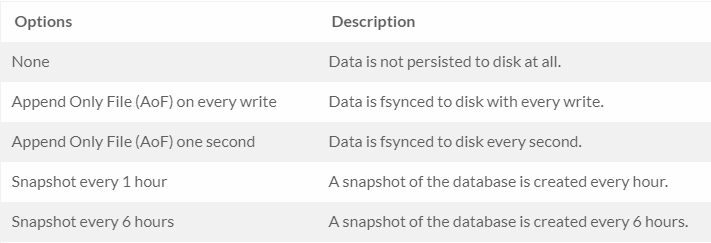
1. **Kiến trúc và cơ chế**
   1. **In – Memory Key – Value Store: Lưu trữ dưới cặp khóa và giá trị trên bộ nhớ RAM.**



* + - Key: phải là một chuỗi
    - Value: có thể là string, list, set, sorted set hoặc hash.
    - Ưu điểm: Redis lưu trữ dữ liệu trong bộ nhớ chính (RAM) nên tốc độ đọc ghi dữ liệu rất nhanh.
    - Nhược điểm: Vì bộ nhớ RAM hạn chế, nên Redis không thể lưu được các dữ liệu lớn mà chỉ lưu những văn bản nhỏ mà cần truy cập, sữa đổi và chèn vào tới một tốc độ rất nhanh.
  1. **Kiến trúc đơn sơ của Redis**



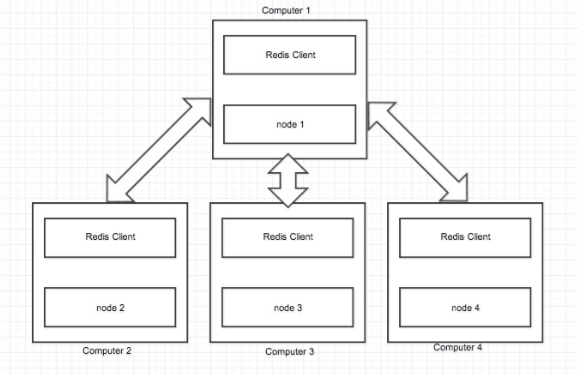
* + - Redis client: điều khiển truy vấn dữ liệu từ giao diện hoặc bất kỳ ngôn ngữ lập trình khác thông qua ngôn ngữ API của Redis để gửi yêu cầu đến server.
    - Redis server: chịu trách nhiệm cho việc sử lý và lưu trưc dữ liệu trong bộ nhớ. Nó xử lý tất cả các loại hình quản lý và tạo thành phần chính của kiến trúc.
    - Client và Server có thể nằm trên 1 máy hoặc 2 máy vật lý khác nhau.
  1. **Persistence (Duy trì, lưu trữ dữ liệu cứng)**
     + RDB: tạo 1 bản sao toàn bộ dữ liệu từ ram và cất sang bộ nhớ đĩa. Điều này được thực hiện định kỳ, do đó sẽ mất dữ liệu kể từ bản sao cuối. Phù hợp cho việc phục hồi từ thảm họa, dể dàng cất dữ trên trung tâm dữ liệu. Khởi động nhanh.
     + AOF: ghi log tất cả các hoạt động ghi mà được nhận bởi server, mọi thứ đều bền bỉ nhưng file có kích thước lớn hơn so với kiểu RDB.
     + SAVE Command: Người dùng có thể ra lệnh cho máy chủ thực hiện RDB bất cứ lúc nào.
     + Các tham số lựa chọn cấu hình



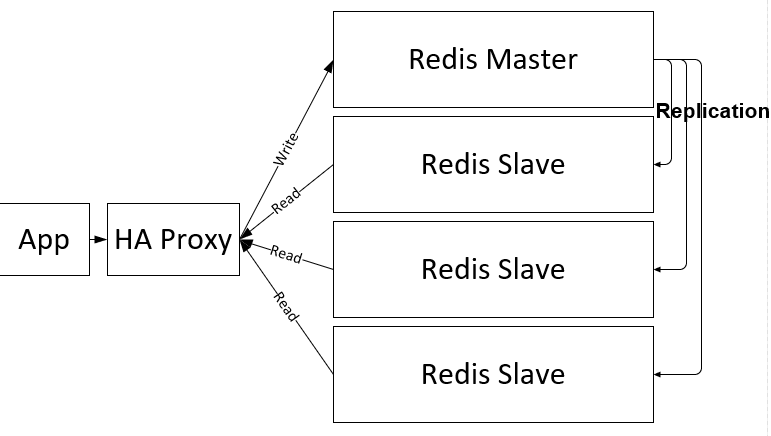
* + - So sánh RDB với AOF

|  |  |
| --- | --- |
| AOF | RDB |
| Hao bộ nhớ đĩa | Ít tốn bộ nhớ đĩa |
| Cung cấp độ bền tốt hơn (phục hồi điểm mới nhất trong thời gian) | Ít bền |
| Thời gian phục hồi lâu do kích thước file lớn | Phục hồi nhanh |
| Không gian đĩa cần mở rộng, vì File tăng kích cỡ theo thời gian | Không gian đĩa ít mở rộng, vì thực hiện một lần vài giờ |

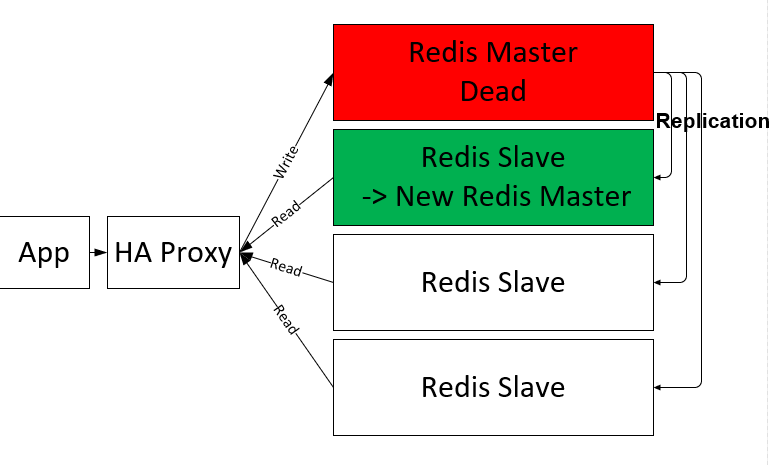
* + - Nguồn: <https://redis.io/topics/persistence>
  1. **Clustering (gom nhóm):**
     + Clustering là một kỹ thuật cho phép dữ liệu có thể được chia ra và nằm nhiều máy tính. Lợi thế chính là dùng nhiều dữ liệu hơn vì nó là sự kết hợp RAM của các máy tính.
     + Redis Cluster cũng cung cấp một số mức độ sẵn có trong quá trình phân vùng, nên có khả năng tiếp tục hoạt động khi một số nút hỏng hoặc không thể giao tiếp.
     + Tuy nhiên cụm ngừng hoạt động trong trường hợp có sự cố lớn hơn. Ví dụ như khi đa số master không có sẵn.
     + Giả sử chúng ta có một máy chủ redis với 64GB RAM, nghĩa là chúng ta chỉ có thể có 64GB dữ liệu. Bây giờ nếu chúng ta có 10 máy tính được nhóm lại với mỗi 64GB RAM thì chúng ta có thể lưu trữ 640GB dữ liệu.



* + - Trong hình trên, chúng ta có thể thấy rằng dữ liệu được chia thành bốn nút. Mỗi nút là một máy chủ redis được cấu hình như một nút cụm.
    - Nguồn: <https://redis.io/topics/cluster-tutorial>, <http://qnimate.com/overview-of-redis-architecture/>
  1. **Replication(Nhân bản)**
     + Là một kỹ thuật liên quan đến nhiều máy tính để cho phép chia sẻ khả năng truy cập dữ liệu. Trong một môi trường nhân rộng, nhiều máy tính chia sẻ cùng một dữ liệu với nhau để ngay cả khi một vài máy tính bị hỏng, tất cả dữ liệu sẽ có sẵn.
     + Tất cả các nô lệ có chứa chính xác cùng một dữ liệu như tổng thể. Có thể có nhiều như nô lệ cho mỗi máy chủ chính. Khi một nô lệ mới được đưa vào môi trường, thì nó sẽ tự động đồng bộ hóa tất cả dữ liệu.
     + Mọi truy vấn được đưa đến master, master sẽ thực hiện các hoạt động. Khi có thao tác ghi master sẽ phân phối dữ liệu đó đến Slave.



* + - Khi một master và một slave kết nối tốt, master giữ nô lệ cho việc cập nhật bằng cách gửi một luồng lệnh để tái tạo các thay đổi trên tập dữ liệu xảy ra trong bộ dữ liệu tổng thể. (lệnh ghi, key hết hạn, thu hồi key ….).
    - Khi mối kết nối giữa master và nô lệ bị đứt, do mạng hoặc timeout được cảm nhận trong master hoặc nô lệ, nô lệ kết nối lại và cố gắng tiến hành đồng bộ hóa, nó sẽ cố gắng để có được một phần của dòng lệnh từ server mà nó bị mất trong quá trình ngắt kết nối.
    - Khi không thể đồng bộ hoá một phần, nô lệ sẽ yêu cầu đồng bộ hóa lại toàn bộ. Điều này sẽ liên quan đến một quá trình phức tạp hơn, master cần tạo một ảnh chụp nhanh tất cả dữ liệu của nó, gửi nó đến nô lệ. Sau đó mọi tiến trình vẫn như củ.
    - Khi một Slave được đưa vào môi trường thì hệ thống sẽ tự đồng bộ hóa dữ liệu với Slave mới.
    - Master chết thì đưa Slave củ lên làm master, công việc được tiếp tục.



* + - Nguồn: <https://redis.io/topics/replication>