https://medium.com/@ayushsaxena823/what-is-redis-and-how-does-it-work-cfe2853eb9a9  
Redis đơn luồng   
Bạn có thể thắc mắc: Tại sao Redis lại chọn thiết kế đơn luồng? Chẳng phải sử dụng nhiều luồng và tận dụng toàn bộ lõi CPU để xử lý song song sẽ hiệu quả hơn sao?

Câu trả lời nằm ở bản chất của khối lượng công việc mà Redis đảm nhiệm. Redis giống như một đầu bếp trong gian bếp máy tính, xử lý từng tác vụ một để đảm bảo sự hoàn hảo. Mặc dù có nhiều trợ lý có vẻ là ý tưởng hay, nhưng đối với Redis, nó ưu tiên một luồng duy nhất để đảm bảo độ chính xác, tránh sự hỗn loạn và thực hiện từng nhiệm vụ một cách hoàn hảo.

Dù nhiều ứng dụng hưởng lợi từ xử lý song song, Redis tập trung chủ yếu vào truy cập dữ liệu nhanh và độ trễ tối thiểu. Thiết kế đơn luồng đơn giản hóa việc thực thi, đảm bảo các lệnh được thực hiện tuần tự mà không phải đối mặt với các vấn đề phức tạp liên quan đến quản lý nhiều luồng hoặc đồng bộ hóa.

Trong Redis, mỗi lệnh được thực thi một cách nguyên tử, đảm bảo tính nhất quán. Thiết kế đơn giản này góp phần tăng tốc độ thực thi, vì luồng đơn có thể tận dụng toàn bộ bộ nhớ đệm của CPU, giảm thiểu các lần truy cập thất bại (cache miss) và tối ưu hóa hiệu suất.

Trong môi trường đa luồng, các luồng chia sẻ một số tài nguyên, bao gồm cả bộ nhớ đệm CPU. Khi một luồng cập nhật dữ liệu, nó có thể khiến bộ nhớ đệm bị cập nhật hoặc mất hiệu lực. Nếu một luồng khác đang sử dụng cùng dữ liệu hoặc dữ liệu liên quan, nó có thể gặp phải cái gọi là “cache miss” vì dữ liệu cần thiết không còn trong bộ nhớ đệm hoặc đã bị thay đổi.

Tình huống này tạo ra hiện tượng tranh chấp bộ nhớ đệm hoặc xung đột bộ nhớ đệm (cache contention/cache thrashing), nơi các luồng cạnh tranh truy cập vào cùng một không gian bộ nhớ đệm hạn chế. Thay vì dòng dữ liệu chảy trơn tru, bạn sẽ gặp phải các gián đoạn và sự kém hiệu quả do các luồng tranh giành quyền truy cập, dẫn đến tắc nghẽn hiệu suất tiềm ẩn.

Vậy là chúng ta đã hiểu tại sao Redis chọn cách tiếp cận đơn luồng. Nhưng đây là câu hỏi thú vị: Làm thế nào mà một luồng duy nhất có thể xử lý hàng ngàn yêu cầu đến và phản hồi đi một cách đồng thời? Chẳng phải luồng này sẽ bị chặn lại khi xử lý từng yêu cầu sao?

Quản lý hàng ngàn kết nối một cách hài hòa  
Đây là lúc I/O multiplexing xuất hiện — đây chính là "bí kíp" giúp Redis đạt được tính đồng thời rõ ràng. Nghe có vẻ là một thuật ngữ phức tạp, đúng không? Hãy để tôi giải thích chi tiết.

Các bước thực hiện:

1. Redis Server Đang Hoạt Động Tại Một Địa Chỉ: Redis, giống như một nhân viên siêng năng, luôn sẵn sàng hoạt động, sẵn sàng xử lý các yêu cầu và chia sẻ dữ liệu của nó.
2. Redis Chấp Nhận Nhiều Kết Nối TCP Qua Các Client: Khi nhiều client (có thể là các ứng dụng hoặc dịch vụ) muốn tương tác với Redis, chúng thiết lập các kết nối TCP. Lúc này, một socket mạng được tạo ra giữa Redis và client. Socket mạng là một kênh giao tiếp ảo hai chiều, nơi dữ liệu có thể chảy theo cả hai hướng.
3. Nếu Không Có I/O Multiplexing: Để đọc dữ liệu từ một socket mạng, Redis thực hiện một lệnh gọi hệ thống (system call) read(). Lệnh này, vốn là một thao tác I/O, có tính chất blocking (chặn luồng), nghĩa là quy trình đơn luồng của Redis sẽ phải chờ trên kết nối TCP liên quan cho đến khi có dữ liệu sẵn sàng để đọc. Dữ liệu này thường là các yêu cầu và dữ liệu liên quan mà các client gửi tới Redis. Hành vi blocking này ngụ ý rằng Redis, với bản chất đơn luồng của nó, sẽ bị giới hạn trong việc xử lý chỉ một kết nối TCP tại một thời điểm. Luồng của máy chủ Redis sẽ phải chờ trên kết nối của một client cụ thể cho đến khi dữ liệu sẵn sàng được đọc, do đó không thể xử lý nhiều kết nối cùng lúc.

Liệu có giải pháp nào cho vấn đề này không?: Chúng ta có thể làm gì đó để chỉ gửi yêu cầu đọc khi biết rằng dữ liệu đã sẵn sàng để đọc, từ đó tránh được tính chất blocking? Chúng ta có thể nhận được một dạng thông báo nào đó kiểu như “Dữ liệu của bạn đã sẵn sàng, hãy đọc và xử lý nó”?

Bạn đúng rồi, đây chính là điều mà I/O multiplexing làm được.

**I/O Multiplexing (Đa luồng I/O)**

I/O Multiplexing cho phép Redis giám sát nhiều kết nối cùng lúc mà không cần chặn luồng chính của nó. Thay vì phải chờ dữ liệu trên một kết nối duy nhất, Redis có thể theo dõi đồng thời nhiều kết nối.

Redis sử dụng lệnh gọi hệ thống select() hoặc poll() để đăng ký quan tâm đến nhiều socket (kết nối) cùng lúc. Các lệnh gọi này cho phép Redis chỉ định một tập hợp các socket mà nó muốn theo dõi cho các sự kiện cụ thể, chẳng hạn như khi sẵn sàng để đọc dữ liệu.

Các lệnh gọi hệ thống như select() hoặc poll() thuộc nhóm lệnh gọi hệ thống giám sát I/O. Redis, với luồng đơn của mình, sẽ gọi lệnh select() hoặc poll() và chuyển sang trạng thái chờ các sự kiện. Trong thời gian này, Redis không xử lý bất kỳ kết nối cụ thể nào; thay vào đó, nó chờ thông báo về các sự kiện xảy ra trên các socket đã đăng ký và xử lý từng yêu cầu từ các socket sẵn sàng.

**Vậy làm thế nào đạt được hiệu quả khi chỉ xử lý từng yêu cầu một?**

Redis tận dụng một cách hiệu quả thực tế rằng hoạt động I/O mạng (network I/O) tốn nhiều thời gian hơn rất nhiều so với các thao tác trên bộ nhớ của Redis (vốn là nguyên tử). Nhờ đó, Redis có thể cung cấp thông lượng cao, độ trễ thấp, và một dạng đồng thời rõ ràng nhưng cực kỳ hiệu quả.

Khi một sự kiện xảy ra trên bất kỳ socket nào đã đăng ký (ví dụ: có dữ liệu sẵn sàng để đọc), lệnh select() hoặc poll() trả về. Giá trị trả về cho biết socket nào có sự kiện, giúp Redis xác định nơi dữ liệu sẵn sàng để xử lý.

Redis sau đó xử lý các sự kiện cụ thể trên các socket được hệ thống thông báo. Ví dụ: nếu có dữ liệu sẵn sàng để đọc trên một socket nhất định, Redis có thể bắt đầu thao tác đọc ngay lập tức mà không cần chờ đợi, giải quyết được vấn đề chặn luồng trong I/O truyền thống.

Cách tiếp cận theo sự kiện này là không đồng bộ (asynchronous); Redis không chủ động kiểm tra từng socket mà thay vào đó phản hồi các sự kiện khi chúng xảy ra. Điều này cho phép Redis quản lý hiệu quả một số lượng lớn kết nối mà không lãng phí tài nguyên cho việc kiểm tra liên tục.

Bằng cách chờ các sự kiện thay vì chặn trên từng socket riêng lẻ, Redis tối đa hóa việc sử dụng luồng đơn và tài nguyên hệ thống. Điều này đảm bảo rằng máy chủ luôn phản hồi nhanh chóng với các sự kiện trên nhiều kết nối mà không bị trì hoãn không cần thiết.

**Lệnh Gọi Hệ Thống Giám Sát I/O**

Lệnh gọi hệ thống giám sát I/O là các hàm được cung cấp bởi hệ điều hành, cho phép chương trình của bạn theo dõi nhiều nguồn nhập/xuất, như file descriptor hoặc socket mạng, cùng một lúc.

Hãy tưởng tượng việc giám sát I/O giống như việc bạn thuê một người gác cổng (watchman) cho chương trình của mình. Bạn chỉ cho người gác cổng biết bạn quan tâm đến những sự kiện nào (sẵn sàng để đọc, ghi, v.v.), và sau đó chương trình của bạn tạm nghỉ. Người gác cổng sẽ theo dõi mọi thứ. Khi có sự kiện xảy ra, người gác cổng sẽ thông báo: “Này, có thứ gì đó đã sẵn sàng rồi!” và lúc đó chương trình của bạn sẽ quay lại để xử lý sự kiện cụ thể đó.

Bài khác <https://medium.com/nerd-for-tech/understanding-redis-in-system-design-7a3aa8abc26a>

**Redis hoạt động như thế nào?**

Toàn bộ dữ liệu trong Redis được lưu trữ trong bộ nhớ chính (RAM) của máy chủ, khác với các cơ sở dữ liệu truyền thống như PostgreSQL, SQL Server và các hệ thống tương tự, thường lưu trữ dữ liệu chủ yếu trên đĩa. Trong khi các cơ sở dữ liệu dựa trên đĩa phải thực hiện nhiều lần truy xuất dữ liệu từ đĩa, thì các hệ thống lưu trữ dữ liệu trong bộ nhớ như Redis không phải chịu những hạn chế đó. Kết quả là Redis đạt được hiệu suất cực nhanh, với thời gian đọc hoặc ghi trung bình chưa đến 1 mili giây và khả năng hỗ trợ hàng triệu thao tác mỗi giây.

**Redis có phải là nguồn dữ liệu chính (First Source of Truth)?**

Câu trả lời là **Không**. Redis không thể được coi là nguồn dữ liệu chính mà thường chỉ được sử dụng như một giải pháp bổ sung để cải thiện hiệu suất hệ thống. Theo định lý CAP (Consistency, Availability, Partition Tolerance), Redis không hoàn toàn nhất quán (consistent) hoặc khả dụng cao (highly available).

Lý do chủ yếu nằm ở cách Redis đồng bộ dữ liệu từ bộ nhớ vào đĩa — yếu tố quyết định sự nhất quán của dữ liệu.

Redis lưu trữ dữ liệu trong bộ nhớ (RAM), giúp tăng tốc độ đọc/ghi dữ liệu, nhưng trong trường hợp máy chủ gặp sự cố, toàn bộ dữ liệu trong bộ nhớ sẽ bị mất. Đối với một số ứng dụng, việc mất dữ liệu trong trường hợp sự cố là chấp nhận được. Nhưng đối với các ứng dụng khác, việc có thể khôi phục dữ liệu sau khi khởi động lại máy chủ là rất quan trọng.

Redis cung cấp một số tùy chọn lưu trữ (persistence):

1. **RDB (Redis Database):** Lưu trữ dưới dạng snapshot dữ liệu tại các thời điểm cụ thể.
2. **AOF (Append Only File):** Ghi lại mọi thao tác ghi dữ liệu để có thể tái dựng dữ liệu gốc khi khởi động lại.
3. **No Persistence:** Không lưu trữ dữ liệu, chỉ giữ dữ liệu trong RAM khi máy chủ đang chạy.
4. **RDB + AOF:** Kết hợp cả hai phương pháp lưu trữ để tăng cường khả năng phục hồi dữ liệu.

**Ưu và nhược điểm của các phương pháp lưu trữ trong Redis**

**RDB (Redis Database)**

**Ưu điểm:**

* Là một tệp duy nhất, nhỏ gọn, phù hợp để sao lưu và phục hồi khi gặp thảm họa.
* Cho phép khởi động lại nhanh chóng với các bộ dữ liệu lớn.
* Dễ dàng lưu trữ và di chuyển đến các trung tâm dữ liệu xa.

**Nhược điểm:**

* Không phù hợp nếu cần giảm thiểu nguy cơ mất dữ liệu. Ví dụ, nếu tạo snapshot mỗi 5 phút, bạn có thể mất dữ liệu từ vài phút cuối nếu Redis gặp sự cố.

**AOF (Append Only File)**

**Ưu điểm:**

* Độ bền cao hơn với các chính sách fsync khác nhau (mỗi giây, mỗi thao tác, hoặc không fsync).
* Dữ liệu được lưu theo kiểu ghi nối tiếp, dễ đọc và dễ khôi phục.
* Redis tự động viết lại tệp AOF khi tệp quá lớn, đảm bảo an toàn trong quá trình này.

**Nhược điểm:**

* Tệp AOF thường lớn hơn RDB cho cùng một bộ dữ liệu.
* Tùy thuộc vào chính sách fsync, hiệu suất có thể thấp hơn RDB.

**Nên sử dụng phương pháp nào?**

* **RDB:** Nếu bạn chấp nhận được việc mất vài phút dữ liệu và ưu tiên tốc độ khởi động lại.
* **AOF:** Nếu bạn cần độ bền dữ liệu cao hơn và có thể chấp nhận tốn nhiều tài nguyên hơn.
* **Kết hợp cả RDB và AOF:** Nếu bạn muốn đảm bảo mức độ an toàn dữ liệu tương đương với PostgreSQL.

**Mô hình lưu trữ quan trọng trong Redis**

**Snapshotting**

Redis mặc định lưu snapshot dữ liệu vào đĩa dưới dạng tệp nhị phân dump.rdb. Bạn có thể cấu hình Redis để lưu snapshot sau mỗi N giây nếu có ít nhất M thay đổi trong dữ liệu, hoặc thực hiện thủ công thông qua lệnh SAVE hoặc BGSAVE.

Snapshotting rất phù hợp cho các trường hợp sao lưu định kỳ hoặc khi bạn cần một bản snapshot gọn nhẹ để phục hồi dữ liệu nhanh chóng.