Nhóm 7

- Vũ Minh Phương
- Võ Lê Ngọc Thịnh

Câu 1:

Khái niệm Load Balancing
Load Balancing là kỹ thuật phân phối đồng đều khối lượng công việc hoặc requests đến nhiều server trong cùng một hệ thống.

Mục tiêu của load balancing là:

- Tối ưu hóa hiệu suất hệ thống.
- Đảm bảo tính sẵn sàng (availability) và độ tin cậy (reliability).
- Tránh tình trạng một server bị quá tải trong khi server khác rảnh rỗi.

Load Balancer có thể hoạt động ở nhiều tầng trong mô hình mạng OSI:

- Layer 4 (Transport Layer): Cân bằng theo địa chỉ IP và cổng (TCP/UDP).
- Layer 7 (Application Layer): Cân bằng dựa trên nội dung HTTP (URL, cookie, header,...).

Cơ chế hoạt động của Load Balancer

1. Client gửi request đến Load Balancer (thay vì gửi trực tiếp đến server).
2. Load Balancer lựa chọn một server backend dựa theo thuật toán phân phối tải.
3. Request được chuyển tiếp đến server đó.
4. Server xử lý và gửi kết quả về cho Load Balancer.
5. Load Balancer trả kết quả lại cho client.

Cơ chế này giúp:

- Tăng khả năng scalability.
- Giảm downtime vì nếu một server hỏng, load balancer sẽ chuyển sang server khác.
- Cải thiện tốc độ phản hồi cho người dùng.

Round Robin Algorithm

Nguyên lý: Phân phối tuần tự các request lần lượt đến từng server theo thứ tự vòng tròn. Ví dụ: Có 3 server S1, S2, S3 thì request 1 $\rightarrow$ S1, request 2 $\rightarrow$ S2, request 3 $\rightarrow$ S3, request 4 $\rightarrow$ S1.

Ưu điểm:

- Dễ cài đặt, logic đơn giản
- Phù hợp khi các server có cấu hình tương đương nhau và thời gian xử lý trung bình tương đương.

Nhược điểm:

- Không xét đến tải thực tế của từng server
- Nếu một server xử lý chậm hơn, nó vẫn nhận lượng request như các server nhanh hơn \(\rightarrow\) dễ gây quá tải.

Nhận xét: **Round Robin** thích hợp cho hệ thống nhỏ, dễ triển khai, ít chênh lệch tài nguyên giữa các server.

Least Connection Algorithm

Nguyên lý: Chọn server đang có ít kết nối hoạt động nhất tại thời điểm đó để xử lý request mới. Ví dụ có 3 server, S1 đang xử lý 10 kết nối, S2 đang xử lý 5 và S3 là 7 \(\rightarrow\) Request tiếp theo sẽ được gửi đến S2 (ít kết nối nhất)

Ưu điểm:

- Cân bằng tải theo trạng thái thực tế của hệ thống.
- Hiệu quả hơn trong môi trường có sự khác biệt về hiệu năng giữa các server hoặc thời gian xử lý request không đồng đều.

Nhược điểm:

- Cần theo dõi trạng thái kết nối của từng server \(\rightarrow\) phức tạp hơn Round Robin.
- Có thể tốn thêm tài nguyên để thống kê và cập nhật trạng thái kết nối.

Nhận xét: **Least Connection** tối ưu hơn cho hệ thống lớn, nơi có sự khác biệt về hiệu suất server hoặc thời gian xử lý request không đồng đều.

Câu 2

Vấn đề mà Raft giải quyết trong hệ thống phân tán

Raft là một thuật toán đồng thuận (consensus algorithm) được thiết kế để dễ hiểu và triển khai hơn so với Paxos, trong khi vẫn cung cấp tính chịu lỗi và tính an toàn tương đương.

Vấn đề cơ bản mà Raft giải quyết trong một hệ thống phân tán là đảm bảo rằng tất cả các máy chủ (server) đồng ý về một chuỗi các thao tác (log entries) đã được cam kết, ngay cả khi một thiểu số máy chủ gặp sự cố (tính chịu lỗi - fault-tolerance).

Mục tiêu chính của Raft là xây dựng một máy trạng thái nhân rộng (replicated state machine), đảm bảo:

- Tính nhất quán (Consistency): Mọi máy chủ đều xử lý các lệnh theo cùng một thứ tự và kết thúc với cùng một trạng thái.
- Tính sẵn sàng (Availability): Hệ thống vẫn có thể tiếp tục hoạt động và xử lý yêu cầu ngay cả khi một số máy chủ bị lỗi.

Cách hoạt động của Raft

Raft quản lý việc nhân bản một log đồng thuận (replicated log). Nó chia vấn đề đồng thuận thành ba vấn đề phụ:

1. Bầu cử (Leader Election)

Các máy chủ trong Raft có thể ở một trong ba trạng thái: Follower, Candidate , hoặc Leader.

- Khi một Follower không nhận được tin nhắn Heartbeat từ Leader hiện tại trong khoảng thời gian chờ (election timeout), nó chuyển sang trạng thái Candidate.
- Candidate bắt đầu một nhiệm kỳ (term) mới, tự bỏ phiếu cho chính mình, và gửi yêu cầu RequestVote đến các máy chủ khác.
- Nếu Candidate nhận được phiếu bầu từ một đa số (quorum) máy chủ, nó trở thành Leader.
- Leader mới bắt đầu gửi các tin nhắn Heartbeat định kỳ đến tất cả Follower để duy trì vị trí và sự sống của mình.

2. Nhân rộng Log (Log Replication)

Leader là máy chủ duy nhất chịu trách nhiệm chấp nhận các lệnh từ client và nhân rộng chúng.

- Khi Leader nhận được một lệnh từ client, nó ghi lệnh đó vào log của riêng mình như một mục chưa được cam kết (uncommitted).
- Nó gửi yêu cầu AppendEntries chứa mục log này đến tất cả các Follower.
- Nếu một số đa số máy chủ (bao gồm cả Leader) đã ghi mục log này vào log của họ, Leader sẽ coi mục log đó là đã được cam kết.
- Mục log đã được cam kết sau đó được áp dụng vào máy trạng thái của Leader, và Leader phản hồi lại client.
- Leader sử dụng các yêu cầu AppendEntries tiếp theo (hoặc Heartbeat) để thông báo cho Follower biết mục log nào đã được cam kết, và Follower cũng áp dụng các mục log đó vào máy trạng thái của mình.

3. Tính an toàn (Safety)

Raft có các quy tắc nghiêm ngặt để đảm bảo rằng log không bị mâu thuẫn:

- Election Restriction: Một Candidate phải có log ít nhất mới bằng so với đa số các máy chủ khác để thắng cử. Điều này đảm bảo Leader mới chắc chắn chứa tất cả các mục log đã được cam kết.

- Log Matching: Leader sử dụng cơ chế kiểm tra và ghi đè để buộc các log của Follower phải khớp hoàn toàn với log của nó, nhằm giải quyết mọi sự khác biệt về log.

Khi nào nên sử dụng Raft?

Nên sử dụng Raft khi cần một hệ thống phân tán yêu cầu đồng thuận mạnh mẽ (strong consensus) và tính nhất quán cao (high consistency).

Raft là lựa chọn phù hợp cho:

- Hệ thống lưu trữ cấu hình: Ví dụ như etcd. Cần đảm bảo rằng tất cả các thành phần trong hệ thống có cùng một cấu hình chính xác và nhất quán.
- Các hệ thống lưu trữ có tính sẵn sàng cao: Nơi dữ liệu phải được nhân rộng một cách an toàn và nhất quán trên nhiều bản sao để đảm bảo không mất dữ liệu.
- Các thành phần quan trọng đòi hỏi trạng thái đồng bộ: Khi cần một "nguồn sự thật" duy nhất được chia sẻ và nhất quán giữa các dịch vụ.