Trường Đại học Bách Khoa - Đại Học Đà Nẵng

Khoa Điện tử - Viễn thông

BÁO CÁO CUỐI KỲ

CHUYÊN ĐỀ 2

Đề tài: Hệ thống cân bằng tải bằng thuật toán

NHÓM 7

Giảng viên hướng dẫn: TS. Nguyễn Văn Hiếu

Sinh viên thực hiện:

Mai Đức Khiêm - 106210220

Ngô Văn Đồng - 106210210

Ngô Nhật Minh - 106210222

Lớp: 21KTMT

Đà Nẵng, 15/12/2025

Lời nói đầu

Trong bối cảnh chuyển đổi số diễn ra mạnh mẽ, các hệ thống thông tin hiện đại ngày càng phải xử lý khối lượng lớn yêu cầu từ người dùng với mức độ biến động cao. Các ứng dụng web, dịch vụ đám mây, thương mại điện tử và nền tảng trực tuyến quy mô lớn đều dựa trên kiến trúc hệ thống phân tán, trong đó nhiều máy chủ backend hoạt động song song nhằm đảm bảo khả năng mở rộng, hiệu năng và độ tin cậy.

Một trong những thành phần cốt lõi quyết định chất lượng vận hành của các hệ thống này là cơ chế cân bằng tải. Việc phân phối các yêu cầu một cách hợp lý không chỉ giúp tối ưu hóa hiệu suất xử lý mà còn góp phần nâng cao trải nghiệm người dùng, giảm chi phí vận hành và tăng khả năng chịu lỗi của toàn hệ thống. Do đó, nghiên cứu về các thuật toán cân bằng tải và đánh giá hiệu quả của chúng trong những điều kiện vận hành khác nhau có ý nghĩa quan trọng cả về mặt học thuật lẫn thực tiễn.

MỤC LỤC

1. Phân công công việc

1.1 Danh sách thành viên

Mai Đức Khiêm - 106210220

Ngô Văn Đồng - 106210210

Ngô Nhật Minh - 106210222

1.2 Tỷ lệ % đóng góp

Mai Đức Khiêm	33%
Ngô Văn Đồng	33%
Ngô Nhật Minh	33%

1.3 Nhiệm vụ từng thành viên

- Ngô Nhật Minh: Mô hình hóa bài toán (LP), mô phỏng giao diện dashboard, trình bày Least-Connection, P2C
- Ngô Văn Đồng : Mô hình hệ thống, trình bày Round Robin, Peak-EWMA
- Mai Đức Khiêm: Xây dựng backend, cấu hình server, xây dựng kịch bản testbench, trình bày Weighted Response Time, Adaptive Load Balancing,

2. Giới thiệu dự án Load Balancing

2.1. Đặt vấn đề và tính cấp thiết của đề tài

Mặc dù hiện nay đã tồn tại nhiều thuật toán cân bằng tải, từ các phương pháp đơn giản như Round Robin, Least Connection cho đến các thuật toán nâng cao dựa trên phản hồi trạng thái hệ thống, việc lựa chọn thuật toán phù hợp cho một hệ thống cụ thể vẫn là một bài toán khó trong thực tế. Các hệ thống backend thường không đồng nhất về năng lực xử lý, đồng thời phải đối mặt với các dạng tải phức tạp như tải đột biến, tải không đều hoặc phân bố heavy-tail. Trong những điều kiện này, một thuật toán hoạt động hiệu quả ở kịch bản này có thể trở nên kém tối ưu hoặc thậm chí gây mất ổn định ở kịch bản khác.

Thực tế cho thấy, việc lựa chọn không phù hợp thuật toán cân bằng tải có thể dẫn đến nhiều hệ quả nghiêm trọng. Về mặt hiệu năng, độ trễ trung bình và đặc biệt là độ trễ đuôi (tail latency) có thể tăng cao, ảnh hưởng trực tiếp đến trải nghiệm người dùng và hiệu quả kinh doanh. Về mặt chi phí, hệ thống có thể phải mở rộng thêm tài nguyên phần cứng để bù đắp cho sự phân phối tải kém hiệu quả. Về mặt ổn định, các thuật toán thiếu khả năng thích ứng với biến động tải có thể gây ra hiện tượng dao động hệ thống hoặc sụp đổ cụm máy chủ khi xảy ra các đỉnh tải bất thường.

Mặc dù cân bằng tải là một chủ đề đã được nghiên cứu rộng rãi, phần lớn các tài liệu hiện có chủ yếu tập trung vào phân tích lý thuyết hoặc mô tả triển khai trong các sản phẩm cụ thể, thiếu đi các nghiên cứu thực nghiệm có hệ thống, so sánh nhiều thuật toán trên các cấu hình hạ tầng và mô hình tải khác nhau. Khoảng trống này khiến các quyết định triển khai trong thực tiễn thường mang tính kinh nghiệm, thiếu cơ sở định lượng rõ ràng. Do đó, việc thực hiện một nghiên cứu thực nghiệm toàn diện để đánh giá và so sánh các thuật toán cân bằng tải trong những kịch bản gần với thực tế vận hành là hết sức cấp thiết.

2.2. Tính mới của đề tài

Tính mới của đề tài thể hiện ở cả cách tiếp cận và phạm vi đánh giá. Thứ nhất, nghiên cứu không chỉ xem xét hiệu suất của các thuật toán trong điều kiện lý tưởng mà còn mô phỏng các kịch bản vận hành thực tế, bao gồm cụm backend không đồng nhất về năng lực xử lý và cụm backend đồng nhất về phần cứng nhưng chịu ảnh hưởng của các yếu tố động như jitter, latency spike và heavy-tail workload.

Thứ hai, nghiên cứu tập trung phân tích đồng thời nhiều chỉ báo quan trọng, bao gồm độ trễ trung bình, độ trễ đuôi (P95, P99) và độ ổn định của hệ thống, thay vì chỉ dựa trên một chỉ số đơn lẻ. Cách tiếp cận này cho phép đánh giá toàn diện hơn các trade-off giữa hiệu suất, chi phí và độ ổn định của từng thuật toán.

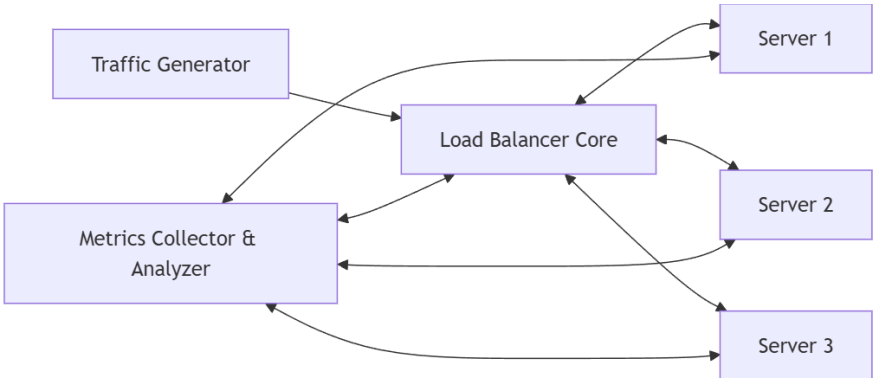
Thứ ba, bên cạnh việc đánh giá các thuật toán heuristic phổ biến, nghiên cứu còn xây dựng một mô hình tối ưu hóa toán học làm baseline lý thuyết, từ đó định lượng mức độ tiệm cận

của các thuật toán thực tế so với nghiệm tối ưu toàn cục. Đây là điểm khác biệt so với nhiều công trình trước vốn chỉ dừng lại ở việc so sánh tương đối giữa các thuật toán.

Cuối cùng, kết quả nghiên cứu không chỉ mang ý nghĩa học thuật mà còn cung cấp các khuyến nghị triển khai cụ thể, giúp các kỹ sư hệ thống lựa chọn thuật toán cân bằng tải phù hợp với đặc điểm hạ tầng và mục tiêu vận hành. Qua đó, đề tài góp phần thu hẹp khoảng cách giữa nghiên cứu lý thuyết và ứng dụng thực tiễn trong lĩnh vực hệ thống phân tán và cân bằng tải.

2.3. Mô hình hệ thống và cách thức hoạt động

Để giải quyết bài toán trên, dự án đề xuất mô hình hệ thống thực nghiệm bao gồm 4 thành phần chính hoạt động theo chu trình khép kín

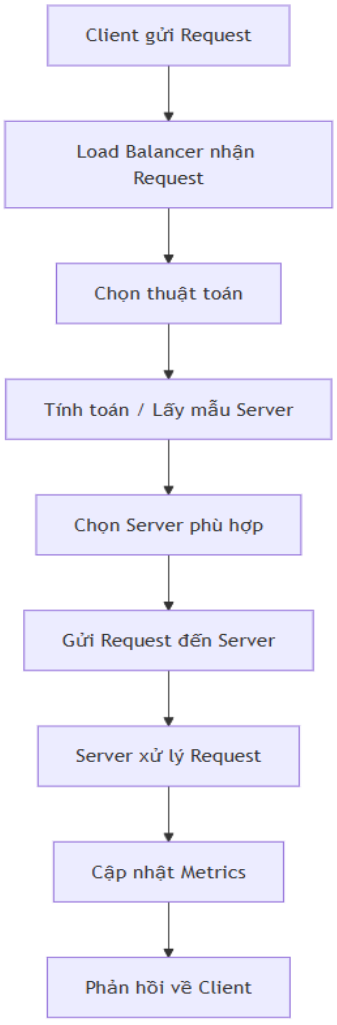


Hình 1: Mô hình hệ thống

Mô tả các thành phần và điểm mới:

1. Traffic Generator (Bộ sinh tải thông minh):
    - Chức năng: Giả lập hành vi người dùng thực tế.
    - Điểm mới: Thay vì gửi request đều đặn, module này hỗ trợ các mô hình phân phối thống kê phức tạp như phân phối Poisson (mô phỏng ngẫu nhiên) và phân phối Pareto (mô phỏng tải nặng phần đuôi - heavy tail), cho phép tái hiện các kịch bản "sốc tải" (bursty traffic) sát với thực tế nhất.
2. Load Balancer Core (Bộ điều phối trung tâm):
    - Chức năng: Tiếp nhận request và định tuyến đến Backend Server.
    - Cơ chế hoạt động: Module này được thiết kế theo kiến trúc "Pluggable Strategy", cho phép hot-swap các thuật toán cân bằng tải khác nhau (P2C, EWMA...) mà không cần khởi động lại hệ thống. Tại đây, các thuật toán sẽ tính toán điểm số (scoring) cho từng server dựa trên dữ liệu lịch sử và trạng thái hiện tại.

3. Servers:
    - Chức năng: Xử lý yêu cầu và trả về kết quả.
    - Cấu hình: Mô phỏng môi trường thực tế nơi các máy chủ có năng lực xử lý (CPU/RAM) khác nhau. Mỗi server có độ trễ xử lý cơ bản ngẫu nhiên và chịu tác động của tải hiện tại (concurrency penalty).
4. Metrics Collector & Analyzer (Bộ thu thập và Phân tích):
    - Chức năng: Ghi nhận log chi tiết từng request (Start time, End time, Server ID, Status).
    - Nhiệm vụ phân tích: Tính toán các chỉ số thống kê chuyên sâu như Avg Latency, P95/P99 Latency, Error Rate và độ lệch chuẩn (Std Dev) để đánh giá mức độ ổn định của thuật toán.



Hình 2: Lưu đồ hoạt động của hệ thống

2.4. Đối tượng thuật toán nghiên cứu

Dự án tập trung phân tích so sánh 6 thuật toán, chia làm 2 nhóm:

Nhóm 1: Các thuật toán cơ bản (Baseline)

Dùng làm mốc so sánh (benchmark), bao gồm:

- Round Robin: Phân phối tuần tự, độ phức tạp O(1).
- Least Connections: Chọn server ít kết nối nhất, độ phức tạp O(N).
- Weighted Response Time: Ưu tiên server phản hồi nhanh, có trọng số.

Nhóm 2: Các thuật toán nâng cao (Advanced)

Là trọng tâm nghiên cứu, bao gồm:

- Power of Two Choices (P2C): Lấy mẫu ngẫu nhiên 2 server và chọn cái tốt nhất. Điểm mới là giảm độ phức tạp tính toán nhưng vẫn đạt hiệu quả cân bằng tải tối ưu về mặt toán học.
- Peak-EWMA: Sử dụng trung bình trượt trọng số hàm mũ để dự báo độ trễ, kết hợp cơ chế phát hiện đỉnh (Peak detection) để loại bỏ server đang gặp sự cố nhất thời.
- Adaptive Load Balancing: Cơ chế phản hồi ngược (Feedback loop) từ server về tải tài nguyên thực (CPU/RAM) để ra quyết định điều hướng.

2.5. Tổng quan về cân bằng tải (Load Balancing)

Cân bằng tải là một kỹ thuật cơ bản trong các hệ thống phân tán hiện đại, nhằm phân phối tối ưu khối lượng công việc (workload) giữa một nhóm các tài nguyên máy chủ backend.. Mục đích cốt lõi là tránh tình trạng một hoặc vài máy chủ bị quá tải trong khi những máy chủ khác ở trạng thái nhàn rỗi, từ đó dẫn đến suy giảm hiệu năng tổng thể, tăng độ trễ phản hồi, và giảm tính sẵn sàng của dịch vụ. Một hệ thống cân bằng tải được thiết kế hiệu quả mang lại những lợi ích chiến lược sau:

- Tăng cường Khả năng Mở rộng (Scalability): Bằng cách phân phối tải đều đặn, hệ thống có thể dễ dàng mở rộng theo chiều ngang (horizontal scaling) bằng cách thêm các máy chủ mới mà không cần thiết kế lại kiến trúc toàn bộ. Nâng cao độ sẵn sàng và tính chịu lỗi (Availability & Fault Tolerance): Tự động phát hiện và loại bỏ các máy chủ gặp sự cố, chuyển hướng lưu lượng sang các máy chủ lành mạnh, đảm bảo dịch vụ liên tục.
- Tối ưu hóa Hiệu suất và Tài nguyên: Cải thiện thời gian đáp ứng (response time), tăng thông lượng hệ thống (throughput), và đảm bảo sử dụng tài nguyên (CPU, RAM, kết nối) một cách cân đối và hiệu quả.

- Đơn giản hóa Bảo trì: Cho phép thực hiện các thao tác bảo trì, nâng cấp mà không cần ngừng toàn bộ dịch vụ (rolling updates).

Do đó, cân bằng tải là thành phần không thể thiếu trong hầu hết các kiến trúc ứng dụng quan trọng, từ các website và API có quy mô lớn, hệ thống microservices phức tạp, đến các nền tảng điện toán đám mây (cloud).

2.6. Bài toán tối ưu hóa hiệu suất máy chủ trong thực tế

Các hệ thống sản xuất hiện nay luôn phải đối mặt với bản chất biến động và khó lường của lưu lượng truy cập, đặt ra những thách thức lớn về hiệu suất:

- Lưu lượng biến động mạnh: Sự gia tăng đột biến (traffic spike) theo giờ cao điểm, các chiến dịch marketing, hoặc sự kiện trực tuyến có thể tạo ra lượng request vượt xa khả năng xử lý thông thường.
- Tính không đồng nhất của hạ tầng: Các máy chủ trong cụm thường có cấu hình phần cứng khác nhau (CPU, bộ nhớ), dẫn đến khả năng xử lý không đồng đều.
- Đa dạng loại tải: Hệ thống phải xử lý đồng thời nhiều loại công việc khác nhau, từ các API nhẹ, request HTTP ngắn đến các tác vụ nền (background jobs) tốn nhiều CPU hoặc I/O, hay các kết nối WebSocket kéo dài.
- Các mối đe dọa bảo mật: Các cuộc tấn công từ chối dịch vụ (DDoS) có thể tạo ra một lượng kết nối giả mạo khổng lồ, làm nghẽn tài nguyên.
- Sự cố phần cứng/phần mềm: Máy chủ có thể gặp sự cố bất ngờ, dẫn đến hiện tượng "thắt cổ chai" (bottleneck) hoặc ngừng hoạt động hoàn toàn.

Nếu hệ thống cân bằng tải không được tối ưu hóa để thích ứng với những thách thức này, các hậu quả nghiêm trọng có thể xảy ra:

- Mất cân bằng tải trầm trọng: Một số máy chủ hoạt động hết công suất trong khi số khác gần như không được sử dụng.
- Suy giảm trải nghiệm người dùng: Thời gian phản hồi tăng cao, xuất hiện lỗi timeout hoặc dịch vụ không khả dụng.
- Giảm thông lượng toàn hệ thống: Tổng số request xử lý được trên một đơn vị thời gian bị sụt giảm.
- Lãng phí tài nguyên đầu tư: Các máy chủ mạnh không được khai thác hết tiềm năng.

Vì vậy, việc nghiên cứu và áp dụng các thuật toán cân bằng tải thông minh và thích nghi là nhiệm vụ then chốt để xây dựng một hệ thống backend ổn định, hiệu suất cao và có khả năng phục hồi (resilient).

2.7. Các bài toán con trong thiết kế hệ thống cân bằng tải

Để đạt được mục tiêu tổng thể, một bộ cân bằng tải cần giải quyết đồng thời một tập hợp các bài toán con mang tính thực tiễn cao:

1. Bài toán phân phối tối ưu: Làm thế nào để phân phối lưu lượng một cách công bằng và hiệu quả, có tính đến năng lực xử lý khác nhau của từng máy chủ (sử dụng trọng số - weight)?
2. Bài toán phản ứng nhanh với biến động tải: Làm thế nào để nhanh chóng phát hiện và phản ứng với sự gia tăng đột biến của lưu lượng hoặc sự suy giảm hiệu năng của một máy chủ cụ thể?
3. Bài toán dự phòng và chịu lỗi (Failover): Cơ chế nào để tự động loại bỏ (health check) và đưa trở lại (recovery) các máy chủ gặp sự cố mà không làm gián đoạn dịch vụ?
4. Bài toán tối đa hóa thông lượng và tối thiểu hóa Độ trễ: Làm thế nào để cân đối giữa việc sử dụng hết công suất các máy chủ (để tăng throughput) và việc giữ cho thời gian xử lý mỗi request là thấp nhất (low latency)?
5. Bài toán Quản lý Trạng thái Kết nối (Connection Pooling/Long-lived Connections): Xử lý hiệu quả các loại kết nối có thời gian sống dài để tránh làm nghẽn một máy chủ cụ thể.
6. Bài toán ổn định và giảm thiểu Dao động (Oscillation): Tránh hiện tượng "bật tắt" (flapping) khi máy chủ liên tục được xem là nặng rồi nhẹ, dẫn đến việc chuyển hướng lưu lượng một cách không ổn định.

2.8. Các Thuật toán Cân bằng tải cơ bản và nhược điểm

Các hệ thống truyền thống thường triển khai các thuật toán đơn giản, dễ tính toán:

- Round Robin (Luân phiên): Phân phối request lần lượt theo vòng giữa các máy chủ. Ưu điểm: Đơn giản, công bằng. Nhược điểm: Không tính đến tải hiện tại, thời gian phản hồi hay cấu hình máy chủ, dễ dẫn đến mất cân bằng nếu các request có độ phức tạp khác nhau.
- Least Connections (Ít kết nối nhất): Chọn máy chủ đang có số kết nối đang hoạt động (active connections) ít nhất. Ưu điểm: Phản ánh trạng thái tải hiện tại tốt hơn Round Robin. Nhược điểm: Chưa phân biệt được "trọng lượng" của từng kết nối (một kết nối WebSocket dài so với một API call ngắn), và không tính đến mức độ sử dụng CPU, bộ nhớ.
- Weighted Round Robin/Least Connections (Có trọng số): Phiên bản mở rộng của hai thuật toán trên, cho phép gán trọng số dựa trên năng lực máy chủ. Ưu điểm: Phù

hợp với cụm server không đồng nhất. Nhược điểm: Trọng số tĩnh, không tự điều chỉnh theo tải động thực tế.

Tổng kết hạn chế chung: Các thuật toán cơ bản chủ yếu dựa trên thông tin tức thời hoặc quy tắc tĩnh, thiếu khả năng học hỏi từ lịch sử và thích nghi với sự thay đổi của môi trường. Chúng thường phản ứng chậm với spike load, dễ gây dao động, và không tối ưu được hiệu suất tổng thể trong các kịch bản phức tạp.

2.9. Hướng tiếp cận với thuật toán nâng cao

Để khắc phục những hạn chế trên, các nghiên cứu gần đây tập trung vào các thuật toán cân bằng tải thích nghi (adaptive) và dựa trên lấy mẫu thống kê, sử dụng nhiều thông tin hơn và có cơ chế phản hồi (feedback loop). Có 3 thuật toán tiêu biểu bao gồm:

- Power of Two Random Choices (P2C): Thay vì chọn một máy chủ duy nhất, thuật toán lấy mẫu ngẫu nhiên hai máy chủ và chọn cái tốt hơn dựa trên một chỉ số (ví dụ: số kết nối ít nhất). Phương pháp này giảm thiểu đáng kể độ dao động và cho kết quả gần tối ưu với độ phức tạp tính toán thấp.
- Peak Exponentially Weighted Moving Average (Peak-EWMA): Kết hợp chỉ số độ trễ (latency) được làm mượt theo hàm mũ (EWMA) để phản ánh xu hướng dài hạn, cùng với cơ chế phát hiện "đỉnh" (peak) để phản ứng nhanh với các request chậm bất thường. Đây là thuật toán nhạy cảm với hiệu suất thực tế của backend.
- Các Thuật toán Thích nghi (Adaptive Load Balancing): Điển hình như thuật toán được sử dụng trong gRPC, nơi bộ cân bằng tải liên tục cập nhật trọng số cho các máy chủ dựa trên các chỉ số hiệu suất thời gian thực như tỷ lệ lỗi, độ trễ, và tải CPU.

Các thuật toán này hướng tới mục tiêu: đưa ra quyết định phân phối thông minh hơn dựa trên dữ liệu, cân bằng giữa phản ứng nhanh với biến động ngắn hạn và duy trì sự ổn định trong dài hạn.

2.10. Mục tiêu và phạm vi dự án

Dự án này được thực hiện với các mục tiêu chính sau:

1. Nghiên cứu lý thuyết: Tìm hiểu sâu về cơ chế, ưu nhược điểm của nhóm thuật toán cân bằng tải nâng cao (P2C, Peak-EWMA, Adaptive).
2. Triển khai và Mô phỏng: Xây dựng một mô hình mô phỏng (simulation) hoặc prototype cho phép triển khai và so sánh các thuật toán này trong một môi trường mạng được kiểm soát, bao gồm: một cụm backend server, bộ tạo lưu lượng (traffic generator), và một bộ cân bằng tải có thể thay đổi thuật toán.
3. Đánh giá định lượng: Thiết lập một bộ chỉ số đánh giá (metrics) toàn diện để đo lường hiệu quả của từng thuật toán, bao gồm:

- Hiệu suất: Thời gian phản hồi trung bình/phân vị (p50, p95, p99), thông lượng hệ thống (requests/second).
- Sự Cân bằng: Độ lệch chuẩn của tải (CPU, số kết nối) giữa các máy chủ.
- Tính ổn định và thích nghi: Khả năng xử lý traffic spike, thời gian phục hồi sau sự cố, mức độ dao động trong phân phối tải.

4. Phân tích và đề xuất: Dựa trên kết quả thực nghiệm, tiến hành phân tích điểm mạnh/yếu của từng thuật toán trong các kịch bản tải khác nhau (ổn định, biến động mạnh, server không đồng nhất). Từ đó, đưa ra các đề xuất về việc lựa chọn thuật toán phù hợp cho từng ngữ cảnh ứng dụng cụ thể.

Thông qua đó, dự án kỳ vọng cung cấp một cái nhìn thực tiễn và có cơ sở khoa học, góp phần vào việc lựa chọn và triển khai giải pháp cân bằng tải tối ưu cho các hệ thống máy chủ hiện đại.

3. Mô tả và giải quyết bài toán tối ưu Load Balancing

3.1 Tổng quan bài toán

Bài toán được mô hình hóa dưới dạng Quy hoạch Tuyến tính Nguyên (Integer Linear Programming – ILP). Mục tiêu là tìm phương án phân phối tập hợp các yêu cầu (requests) đến tập hợp các máy chủ (servers) sao cho cực tiểu hóa hàm mục tiêu tổng hợp (giữa độ trễ xử lý và chi phí vận hành), đồng thời thỏa mãn các ràng buộc cứng về năng lực chịu tải của hệ thống.

3.2. Xây dựng mô hình toán học

3.2.1. Các tập hợp và chỉ số

- Gọi $J = \{1, 2, \ldots, M\}$ là tập hợp các requests gửi đến hệ thống trong thời gian Δt.
- Gọi $I = \{1, 2, \ldots, N\}$ là tập hợp các backend servers khả dụng (Ví dụ: Server Fast, Server Medium, Server Slow).
- $j \in J$: chỉ số của request.
- $i \in I$: chỉ số của server.

3.2.2 Các tham số đầu vào

Các tham số này được thu thập từ module Monitoring (dashboard.py) và Benchmark:

1. Độ trễ dự kiến (L_{ij}): Thời gian ước tính để server i xử lý request j.
    - Liên hệ thực tế: Giá trị này tương ứng với số liệu từ thuật toán Peak-EWMA hoặc Weighted Response Time.
2. Chi phí vận hành (C_i): Chi phí thuê server i trên mỗi đơn vị thời gian (hoặc mỗi request).
    - Liên hệ thực tế: Tương ứng với cấu hình giá: Fast, Medium, Slow.
3. Năng lực xử lý (K_i): Số lượng request tối đa (hoặc ngưỡng CPU) mà server i có thể chịu tải đồng thời.
    - Liên hệ thực tế: Tương ứng với ngưỡng 100% CPU trong thuật toán Adaptive.
4. Trọng số ưu tiên (α, β): Hệ số cân bằng giữa mục tiêu hiệu năng (Latency) và chi phí (Cost).

3.2.3 Biến quyết định

Ta ký hiệu biến nhị phân:

$$x_{ij} = \begin{cases} 1 & \text{nếu request j được gán cho server i} \\ 0 & \text{ngược lại} \end{cases}$$

Với:

- $i \in R$: tập các request cần xử lý
- $j \in S$: tập các server trong hệ thống

Biến x_{ij} là đầu ra của mô hình, cho biết request được phân phối tới server nào.

3.2.4 Hàm mục tiêu

Bài toán cần giải quyết sự đánh đổi (trade-off) giữa "Tốc độ" và "Chi phí". Ta xây dựng hàm mục tiêu cực tiểu hóa tổng thiệt hại (Penalty):

Để cân bằng hai yếu tố này, ta xây dựng hàm mục tiêu tổng hợp như sau:

$$\text{Minimize } Z\left(\alpha \sum_{i=1}\sum_{j=1} L_{ij}.x_{ij} + \beta \sum_{i=1}\sum_{j=1} C_i x_{ij}\right)$$

Trong đó:

- Vế thứ nhất đại diện cho Quality of Service (QoS): Càng nhỏ càng tốt (độ trễ thấp).
- Vế thứ hai đại diện cho Cost Efficiency: Càng nhỏ càng tốt (chi phí thấp).

3.2.5 Ràng buộc của bài toán

(1) Ràng buộc duy nhất (Assignment Constraint): Mỗi request j bắt buộc phải được xử lý bởi đúng một server duy nhất (không được bỏ sót request).

$$\sum_{j=1} x_{ij} = 1, \forall i \in J$$

(2) Ràng buộc năng lực (Capacity Constraint): Tổng số request được gán cho server i không được vượt quá khả năng chịu tải K_icủa nó. Ràng buộc này ngăn chặn hiện tượng quá tải cục bộ (Overload) gây ra Crash server.

$$\sum_{j=1} x_{ij} \leq K_j, \forall i \in 1$$

(3) Ràng buộc miền biến

$$x_{ij} \in \{0,1\}$$

3.2.6 Đầu ra của mô hình

Sau khi giải bài toán ILP, kết quả thu được bao gồm các thành phần sau, đóng vai trò là cơ sở lý thuyết (Theoretical Baseline) để đánh giá hiệu năng của các thuật toán thực nghiệm:

1. Ma trận quyết định tối ưu $\mathbf{X}^*$: Là tập hợp các giá trị $\mathbf{x_{ij}^*} \in \{0, 1\}$ cho biết phương án điều phối cụ thể: Request j nên được đưa vào Server i nào để đạt hiệu quả cao nhất.

$$\mathbf{X}^* = \{\boldsymbol{X_{ij}} \mid \mathbf{x_{ij}^*} \in \{\mathbf{0}, \mathbf{1}\},\ \forall \boldsymbol{i} \in \boldsymbol{I},\ \forall \boldsymbol{j} \in \boldsymbol{J}\}$$

2. Giá trị hàm mục tiêu tối thiểu ($\mathbf{Z_{min}}$) :Đây là giới hạn dưới (Lower Bound) của bài toán, đại diện cho chi phí và độ trễ thấp nhất mà hệ thống có thể đạt được trong điều kiện lý tưởng.

$$\mathbf{Z_{min}} = \mathbf{f}(\mathbf{X}^*)$$

3. Phân bố tải tối ưu trên từng Server ($\mathbf{Load_i^*}$): Tổng hợp số lượng request được gán cho từng server trong phương án tối ưu, giúp xác định mức độ tận dụng tài nguyên lý tưởng:

$$\mathbf{Load_i^*} = \sum_{j \in J} \boldsymbol{X_{ij}} \ \forall \boldsymbol{i} \in \boldsymbol{I}$$

Đây là nghiệm tối ưu toàn cục – đóng vai trò cơ sở so sánh (baseline) để đánh giá hiệu quả các thuật toán heuristic như Round Robin, Least Connection, P2C, Peak-EWMA hay Adaptive Load Balancing.

3.3 Môi trường mô phỏng

Để đánh giá hiệu quả của các thuật toán cân bằng tải và kiểm chứng mô hình ILP trong điều kiện thực tế, nhóm xây dựng một môi trường mô phỏng gồm bốn thành phần chính: Dashboard theo dõi hệ thống, Backend cấu hình server, Traffic Generator tạo tải, Stress Test để kiểm tra giới hạn chịu tải của server, và DDoS Crash Test nhằm mô phỏng tình huống tấn công đột biến.

3.3.1 Dashboard giám sát hệ thống

Dashboard được xây dựng bằng Streamlit, với nhiệm vụ:

- Hiển thị trạng thái hoạt động của Load Balancer theo thời gian thực.
- Theo dõi số request/s và trạng thái từng server (Fast – Medium – Slow).
- Quan sát độ trễ phản hồi (Response Time), mức tải, số request đã xử lý.
- Hiển thị biểu đồ lưu lượng (traffic chart), biểu đồ phân phối tải (allocation chart).
- Giao diện đơn giản, trực quan, giúp dễ dàng kiểm tra các thuật toán.

Dashboard đóng vai trò là Control Panel để quan sát toàn bộ hệ thống đang chạy.

3.3.2 Traffic Generator (Bộ tạo lưu lượng)

Traffic Generator được xây dựng nhằm mô phỏng lưu lượng request gửi tới Load Balancer trong các tình huống khác nhau:

- Tạo lưu lượng đều (constant traffic)
- Tạo lưu lượng ngẫu nhiên (random)

- Tạo lưu lượng theo chu kỳ (burst → nghỉ → burst)
- Tạo spike load mô phỏng giờ cao điểm

Các tham số có thể điều chỉnh:

- Số request/s
- Độ dài thời gian chạy
- Cách phân phối request
- Mức độ nhiễu (noise) hoặc độ biến thiên (variance)

Traffic Generator giúp đánh giá khả năng phản ứng của từng thuật toán LB.

3.3.3 Backend Servers

Gồm các máy chủ được xây dựng theo mô hình đồng nhất hoặc không đồng nhất, có nhiệm vụ xử lý request từ client. Sẽ được giải thích kỹ trong phần testbench thực nghiệm.

3.4 Phân tích thuật toán cân bằng tải

Trong phần này, báo cáo tập trung phân tích cơ sở lý thuyết, quy trình hoạt động và đánh giá ưu nhược điểm của ba thuật toán được lựa chọn nghiên cứu: Power of Two Choices (P2C), Peak-EWMA và Adaptive Load Balancing.

3.4.1 Thuật toán Power of Two Choices (P2C)

3.4.1.1 Nguyên lý hoạt động

Thuật toán P2C được đề xuất dựa trên lý thuyết "bài toán ném bóng vào rổ" (balls-into-bins). Khác với các phương pháp truyền thống phải quét toàn bộ danh sách N máy chủ để tìm phương án tối ưu (độ phức tạp O(N)), P2C áp dụng chiến lược lấy mẫu xác suất. Thuật toán chọn ngẫu nhiên hai máy chủ và điều hướng yêu cầu đến máy chủ có tải thấp hơn.: Theo các chứng minh khoa học, việc giảm không gian lựa chọn từ N xuống 2 vẫn đảm bảo hiệu quả cân bằng tải tiệm cận mức tối ưu trong khi chi phí tính toán giảm đáng kể O(1).

Quy trình thực thi:

1. Lấy mẫu: Bộ cân bằng tải lựa chọn ngẫu nhiên và đồng đều hai máy chủ từ tập N máy chủ backend. Việc lựa chọn dựa trên bộ tạo số ngẫu nhiên.
2. Đánh giá: Hệ thống truy xuất trạng thái tải hiện tại của hai máy chủ được chọn dựa trên chỉ số số kết nối đang hoạt động - active connections.
3. Lựa chọn: Yêu cầu được chuyển tiếp đến máy chủ có chỉ số tải thấp hơn. Nếu hai máy chủ có tải bằng nhau, thuật toán sẽ chọn ngẫu nhiên một trong hai.

3.4.1.2. Đánh giá hiệu năng

- Độ phức tạp tính toán: O(1). Thuật toán chỉ thực hiện hai thao tác truy vấn trạng thái và một phép so sánh, đảm bảo hiệu năng cao ngay cả khi hệ thống xử lý hàng trăm nghìn yêu cầu mỗi giây.
- Hiệu quả cân bằng: P2C tạo ra sự phân bố tải đồng đều trong thời gian ngắn. Xác suất để một máy chủ bị quá tải giảm theo hàm mũ so với phương pháp chọn ngẫu nhiên đơn thuần

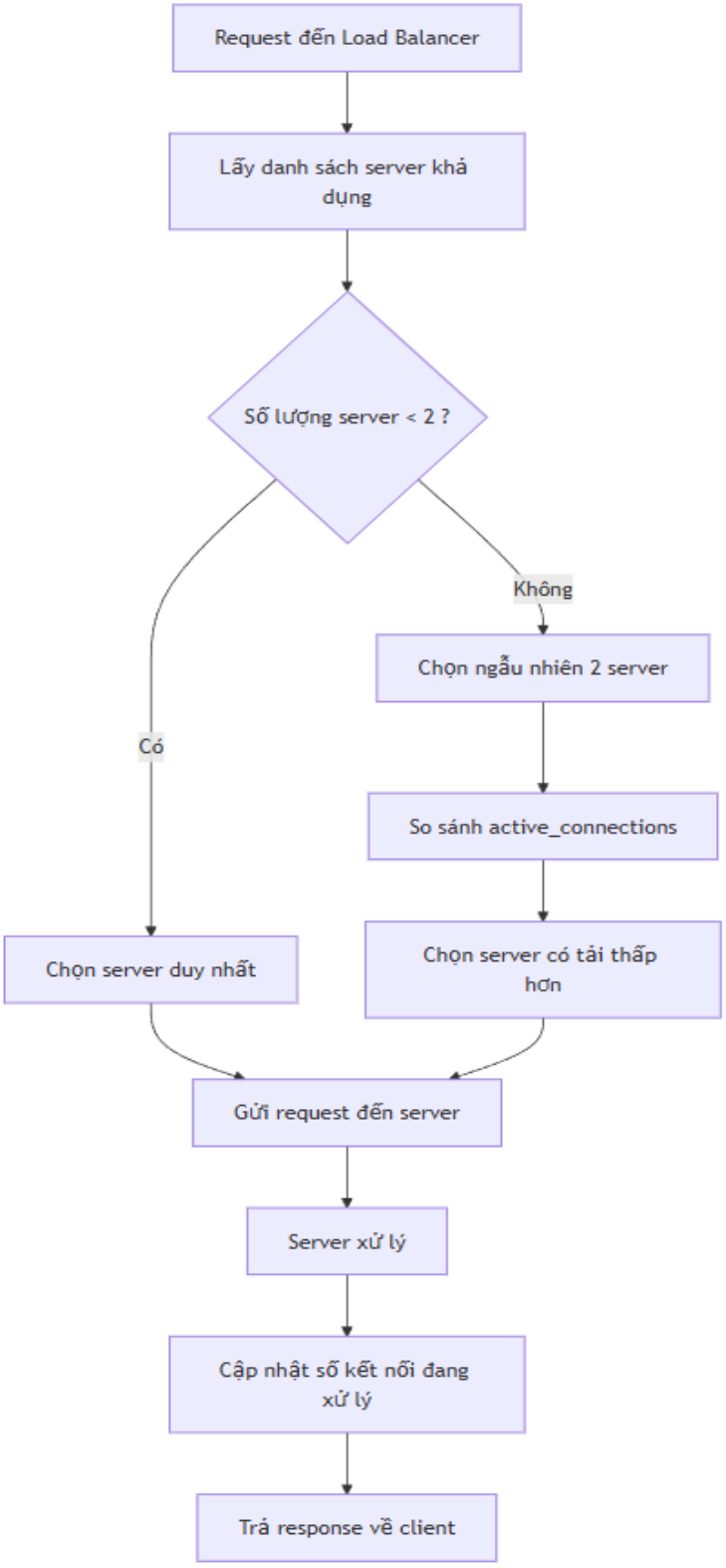
3.4.1.3. Ưu điểm và hạn chế

Ưu điểm:

- Hiệu quả cao với chi phí thấp: Đạt được khoảng 90-95% hiệu quả so với thuật toán tối ưu toàn cục (Least Connections quy mô lớn) nhưng tiêu tốn ít tài nguyên tính toán hơn.
- Khả năng mở rộng: Hiệu suất thuật toán không bị suy giảm khi số lượng máy chủ backend (N) tăng lên.
- Giảm hiệu ứng bầy đàn (Herd Effect): Cơ chế ngẫu nhiên hóa giúp tránh tình trạng tất cả các yêu cầu cùng lúc đổ dồn về một máy chủ vừa phục hồi hoặc đang rảnh rỗi.

Hạn chế:

- Phụ thuộc vào độ chính xác của metric: Nếu metric sử dụng (ví dụ: active_connections) không phản ánh chính xác tải thực sự (ví dụ: CPU, I/O), hiệu quả sẽ giảm.
- Hạn chế với tải không đồng nhất: Phiên bản cơ bản coi tất cả máy chủ là như nhau. Cần kết hợp với cơ chế trọng số (Weighted P2C) cho cụm máy chủ có cấu hình khác nhau.
- Thiếu khả năng dự đoán: Là thuật toán phản ứng (reactive), chỉ dựa trên trạng thái hiện tại mà không dự báo được xu hướng tải trong tương lai gần, có thể kém hiệu quả với các request có thời gian xử lý dài hoặc tăng đột biến.



Hình 3: Lưu đồ thuật toán P2C

3.4.2 Peak-EWMA

3.4.2.1. Nền tảng lý thuyết

Peak-EWMA (Peak Exponential Weighted Moving Average) là thuật toán được thiết kế để khắc phục nhược điểm của việc sử dụng các chỉ số đo lường tức thời vốn thường xuyên bị nhiễu. Thuật toán sử dụng phương pháp trung bình trượt trọng số hàm mũ để ước lượng độ trễ (latency) của máy chủ.

Công thức cập nhật EWMA cơ bản cho máy chủ i tại thời điểm t:

$EWMA_i(t) = \alpha * L_i(t) + (1 - \alpha) * EWMA_i(t-1)$

Trong đó:

- $L_i(t)$: Độ trễ (latency) đo được từ request gần nhất gửi tới máy chủ *i*.
- α ($0 < \alpha \leq 1$): Hệ số làm mịn. Giá trị α lớn làm EWMA nhạy cảm hơn với thay đổi gần đây; α nhỏ khiến EWMA đại diện cho xu hướng dài hạn.

3.4.2.2. Cơ chế phát hiện đỉnh (Peak Detection)

Điểm cải tiến của Peak-EWMA nằm ở cơ chế phản ứng linh hoạt với các biến động độ trễ đột ngột (latency spikes). Thuật toán không sử dụng một hệ số α cố định mà điều chỉnh dựa trên tình huống:

- Trường hợp phát hiện độ trễ tăng (Spike): Nếu $L_i(t) > EWMA_i(t-1)$, thuật toán sử dụng hệ số α peak lớn phản ánh tức thời sự suy giảm hiệu năng của máy chủ.
- Trường hợp phát hiện độ trễ giảm (Recover): Nếu $L_i(t) <= EWMA_i(t-1)$, thuật toán sử dụng hệ số α nhỏ. Điều này giúp chỉ số giảm từ từ, đảm bảo máy chủ đã thực sự ổn định trước khi nhận thêm nhiều tải.

Cơ chế này tạo ra "bộ nhớ không đối xứng": hệ thống phản ứng nhanh với dấu hiệu quá tải nhưng thận trọng khi ghi nhận sự phục hồi.

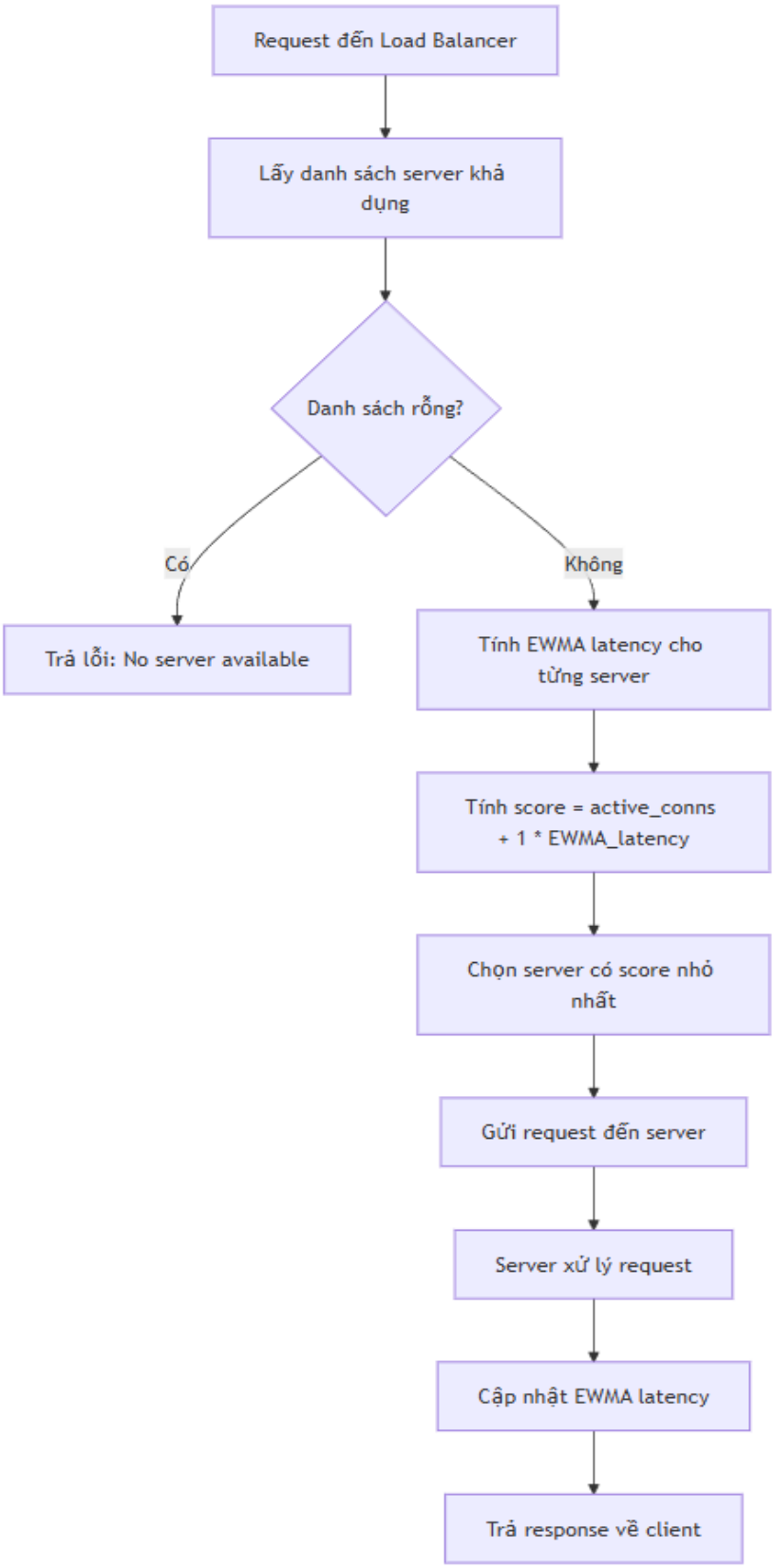
3.4.2.3. Ưu điểm và hạn chế

Ưu điểm:

- Tối ưu hóa Tail Latency: Cải thiện đáng kể các chỉ số phân vị cao (P95, P99) bằng cách nhanh chóng loại bỏ các máy chủ có dấu hiệu chậm trễ ra khỏi danh sách ưu tiên.
- Khả năng chống nhiễu tốt: EWMA lọc bỏ các biến động latency ngắn hạn, không quan trọng.
- Thích nghi với môi trường động: Đặc biệt hiệu quả trong kiến trúc microservice hoặc môi trường đám mây, nơi latency của các dịch vụ phụ thuộc vào nhiều yếu tố và có thể dao động bất thường.

Nhược điểm

- Chi phí triển khai và vận hành: Cần cơ chế đo lường latency chính xác từng request, làm tăng độ phức tạp và overhead so với P2C. Việc tinh chỉnh hai tham số α và α_peak đòi hỏi kiến thức về đặc thù hệ thống.
- Độ trễ phản hồi: EWMA là bộ lọc thấp tần số (low-pass filter), có thể phản ứng chậm hơn so với các thuật toán dựa trên trạng thái tức thời khi một máy chủ mới khỏe mạnh gia nhập cụm.
- Phụ thuộc vào tính đại diện của latency: Nếu latency đo được không phải là chỉ số duy nhất hoặc chính xác về tải (ví dụ: máy chủ xử lý batch job tốn CPU nhưng trả lời nhanh các health check), thuật toán có thể đưa ra quyết định sai lệch.



Hình 4: Lưu đồ thuật toán Peak-EWMA

3.4.3 Thuật toán Adaptive Load Balancing (Cân bằng tải thích ứng)

Adaptive Load Balancing là một thuật toán cân bằng tải trong đó quyết định phân phối yêu cầu không dựa trên một tham số cố định, mà liên tục điều chỉnh theo trạng thái thời gian thực của hệ thống. Thuật toán này giám sát các chỉ số hiệu năng như độ trễ phản hồi, số kết nối đang hoạt động, mức sử dụng tài nguyên (CPU, RAM) và tỷ lệ lỗi, sau đó thay đổi chiến lược lựa chọn server một cách động nhằm tối ưu hóa hiệu suất tổng thể.

Không giống các thuật toán tĩnh như Round Robin hoặc Least Connection, thuật toán thích ứng có khả năng tự học và tự điều chỉnh trọng số dựa trên điều kiện tải, giúp hệ thống duy trì sự ổn định ngay cả khi xảy ra đột biến lưu lượng (burst traffic) hoặc khi một phần backend hoạt động không ổn định.

3.4.3.1 Cơ chế phản hồi tài nguyên thực

Adaptive Load Balancing không chỉ dựa vào các chỉ số từ phía bộ cân bằng tải (như số kết nối hay thời gian phản hồi) mà còn tích hợp thông tin phản hồi trực tiếp từ phía máy chủ backend. Các máy chủ sẽ định kỳ hoặc liên tục báo cáo trạng thái sức khỏe của mình (CPU usage, Memory usage, kích thước hàng đợi xử lý) về cho bộ cân bằng tải.

Quyết định điều hướng được đưa ra dựa trên một hàm tổng hợp (Scoring Function) kết hợp giữa:

1. Thông tin cục bộ: Số kết nối đang mở (Active Connections) do Load Balancer ghi nhận.
2. Thông tin phản hồi: Tải thực tế do máy chủ báo cáo.

3.4.3.2 Vấn đề dao động và cơ chế ổn định

Một thách thức lớn của các thuật toán thích ứng là hiện tượng dao động (Oscillation). Khi một máy chủ báo cáo trạng thái "rảnh", bộ cân bằng tải có xu hướng dồn toàn bộ yêu cầu mới về máy chủ đó, khiến nó nhanh chóng bị quá tải.

Để giải quyết vấn đề này, thuật toán Adaptive thường áp dụng hai kỹ thuật:

- Lấy mẫu xác suất: Thay vì chọn tuyệt đối máy chủ tốt nhất, thuật toán chọn máy chủ dựa trên xác suất tỷ lệ thuận với điểm số sức khỏe (Weighted Random).
- Cơ chế làm mịn: Sử dụng các kỹ thuật như trung bình trượt để tránh việc thay đổi quyết định điều hướng quá đột ngột dựa trên một mẫu báo cáo duy nhất.

3.4.3.3. Ưu điểm và hạn chế

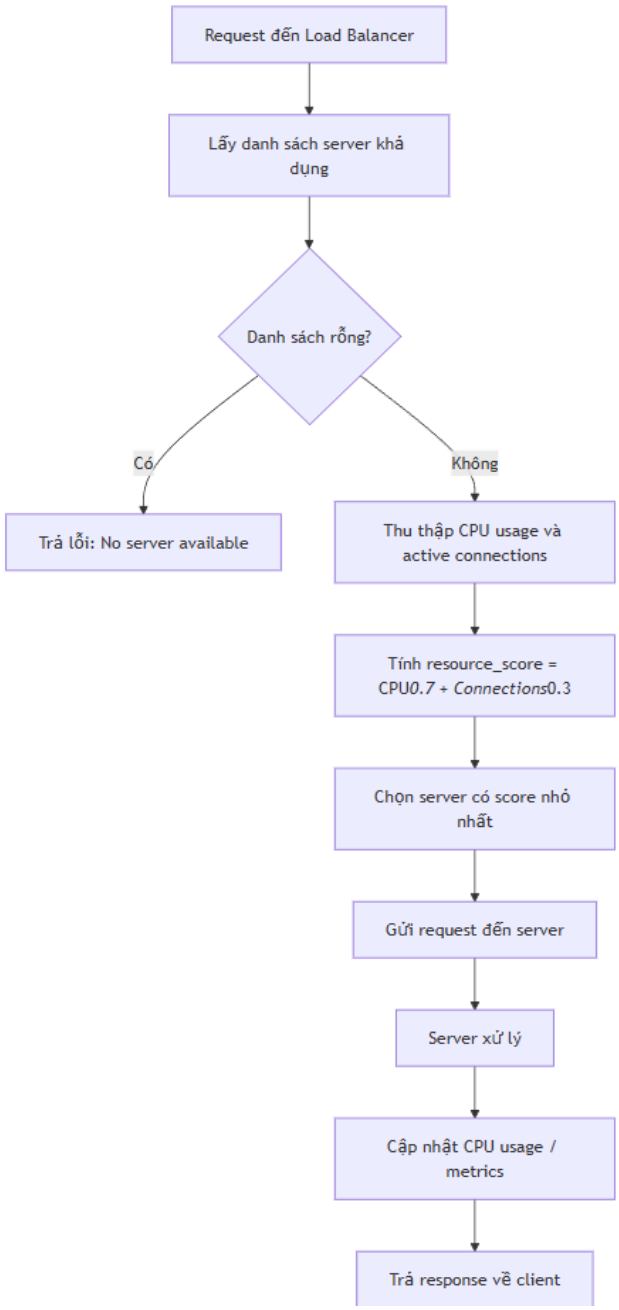
Ưu điểm:

- Phản ánh chính xác nhất trạng thái hệ thống: Tránh được các trường hợp máy chủ trả lời ping rất nhanh nhưng thực tế đang bị nghẽn CPU hoặc I/O.

- Tối ưu hóa tài nguyên: Giúp tận dụng tối đa năng lực phần cứng của từng node trong cụm.

Hạn chế:

- Độ trễ thông tin: Dữ liệu phản hồi từ backend luôn có độ trễ nhất định so với thời gian thực.
- Xâm lấn (Intrusive): Yêu cầu phải cài đặt agent hoặc sửa đổi mã nguồn của ứng dụng backend để gửi báo cáo tải, làm tăng sự phụ thuộc giữa các thành phần hệ thống.



Hình 5: Lưu đồ thuật toán Adaptive

3.4.4 Thuật toán Round Robin

3.4.4.1 Nguyên lý hoạt động

Round Robin là thuật toán cân bằng tải cơ bản và phổ biến nhất nhờ tính đơn giản trong triển khai. Thuật toán hoạt động dựa trên nguyên tắc phân phối xoay vòng: các yêu cầu mới sẽ được gán lần lượt cho từng máy chủ trong danh sách theo một trật tự cố định. Khi đi đến cuối danh sách, thuật toán sẽ quay lại máy chủ đầu tiên.

Quy trình thực thi dựa trên đoạn mã mô phỏng:

1. Khởi tạo: Duy trì một biến đếm toàn cục (current_index) trỏ tới vị trí của máy chủ được chọn gần nhất.
2. Lựa chọn: Khi có yêu cầu mới, hệ thống chọn máy chủ tại vị trí current_index.
3. Cập nhật: Biến đếm được tăng lên 1. Để đảm bảo tính tuần hoàn, phép toán chia lấy dư (modulo) cho tổng số máy chủ N được áp dụng:

   Next_Index = (Current_Index + 1) mod N

3.4.4.2. Đánh giá hiệu năng

- Độ phức tạp tính toán: O(1). Thuật toán không cần thực hiện bất kỳ vòng lặp hay phép so sánh phức tạp nào, chỉ bao gồm các phép toán số học cơ bản.
- Tính phi trạng thái (Stateless): Round Robin không cần lưu trữ hay truy vấn trạng thái tải hiện tại của các máy chủ, giúp tiết kiệm tài nguyên bộ nhớ và băng thông nội bộ.

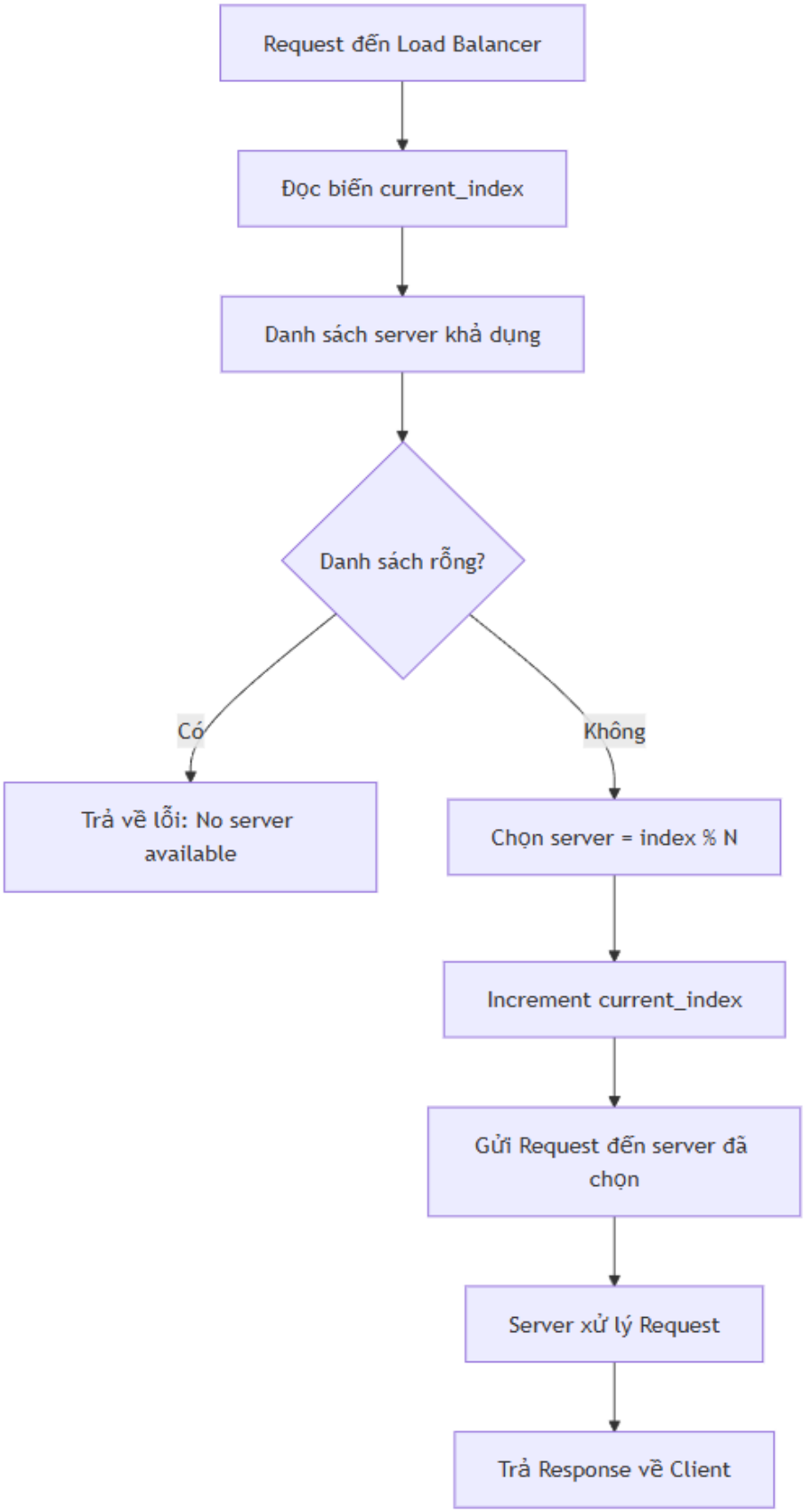
3.4.4.3 Ưu điểm và hạn chế

Ưu điểm:

- Đơn giản và minh bạch: Dễ dàng cài đặt, debug và dự đoán hành vi.
- Công bằng tuyệt đối về số lượng: Đảm bảo mọi máy chủ đều nhận được số lượng yêu cầu bằng nhau sau một chu kỳ đủ lớn.

Hạn chế

- Không nhận biết tải thực tế: Thuật toán giả định tất cả các yêu cầu tiêu tốn tài nguyên như nhau và tất cả máy chủ có cấu hình giống nhau. Điều này dẫn đến tình trạng mất cân bằng khi một máy chủ bị kẹt bởi các tác vụ nặng (long-processing requests) nhưng vẫn tiếp tục nhận thêm yêu cầu mới.
- Hiệu quả kém với cấu hình không đồng nhất: Không phù hợp cho các cụm máy chủ bao gồm cả máy cũ (yếu) và máy mới (mạnh).



Hình 6: Lưu đồ thuật toán Round Robin

3.4.5 Thuật toán Least Connection

3.4.5.1 Nguyên lý hoạt động

Least Connection là một thuật toán động (dynamic), đưa ra quyết định dựa trên trạng thái thực tế của hệ thống. Nguyên lý cốt lõi là điều hướng yêu cầu mới đến máy chủ đang xử lý ít kết nối nhất tại thời điểm đó. Đây là phương pháp tiêu chuẩn để khắc phục nhược điểm của Round Robin.

Quy trình thực thi:

1. Quét trạng thái: Hệ thống duyệt qua danh sách các máy chủ khả dụng.
2. So sánh: Truy xuất chỉ số active_conns (số kết nối đang hoạt động) của từng máy chủ.
3. Lựa chọn: Tìm máy chủ có giá trị active_conns nhỏ nhất (min).

Server_selected = argmin_i * (S_i.active_connections)

3.4.5.2 Đánh giá hiệu năng

- Độ phức tạp tính toán: O(N) đối với danh sách tuyến tính (duyệt qua toàn bộ N máy chủ để tìm giá trị nhỏ nhất).
- Hiệu quả cân bằng: Phản ánh tốt hơn tình trạng bận rộn của máy chủ so với Round Robin.

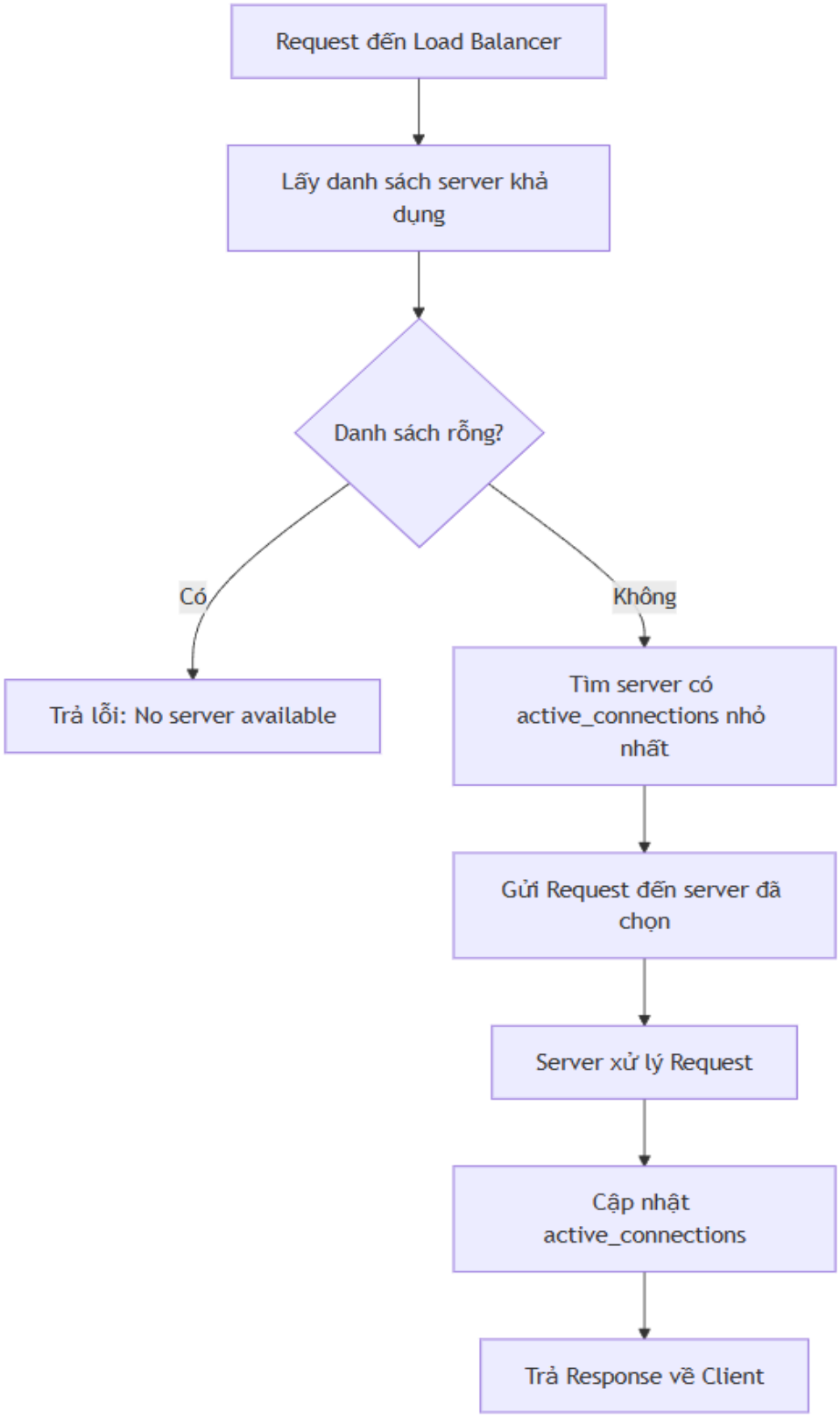
3.4.5.3 Ưu điểm và hạn chế

Ưu điểm:

- Thích ứng với tải không đồng đều: Hoạt động hiệu quả trong môi trường mà thời gian xử lý các yêu cầu chênh lệch lớn (ví dụ: request xử lý ảnh so với request lấy text đơn giản).
- Tránh quá tải cục bộ: Hạn chế việc gửi thêm yêu cầu vào các máy chủ đang bị nghẽn (có nhiều kết nối chưa hoàn thành).

Hạn chế:

- Chi phí tìm kiếm: Với số lượng máy chủ lớn, việc quét toàn bộ danh sách O(N) cho mỗi yêu cầu có thể tạo ra độ trễ.
- Vấn đề "Thundering Herd": Nếu nhiều bộ cân bằng tải cùng hoạt động độc lập, chúng có thể cùng lúc nhận thấy một máy chủ đang rảnh và đồng loạt xả tải vào đó, gây quá tải tức thì.



Hình 7: Lưu đồ thuật toán Least Connection

3.4.6. Thuật toán Weighted Response Time

3.4.6.1. Nguyên lý hoạt động

Đây là thuật toán tinh vi kết hợp giữa năng lực máy chủ (thông qua trọng số - weight) và hiệu năng thực tế (thông qua thời gian phản hồi - response time). Mục tiêu là tối ưu hóa tốc độ phục vụ người dùng cuối.

Hệ thống tính điểm ưu tiên (Score) cho từng máy chủ theo công thức:

$$\text{Score\_i} = \frac{\text{Weight\_i}}{\text{AvgResponseTime\_i}}$$

Trong đó:

- Weight_i: Trọng số do quản trị viên cấu hình (máy mạnh có trọng số cao).
- AvgResponseTime_i: Thời gian phản hồi trung bình của máy chủ (càng thấp càng tốt).

3.4.6.2. Đánh giá hiệu năng

- Độ phức tạp: O(N) để tính điểm và tìm giá trị lớn nhất trong danh sách.
- Tính thích nghi: Đây là thuật toán nhạy bén nhất với độ trễ mạng và sức khỏe ứng dụng.

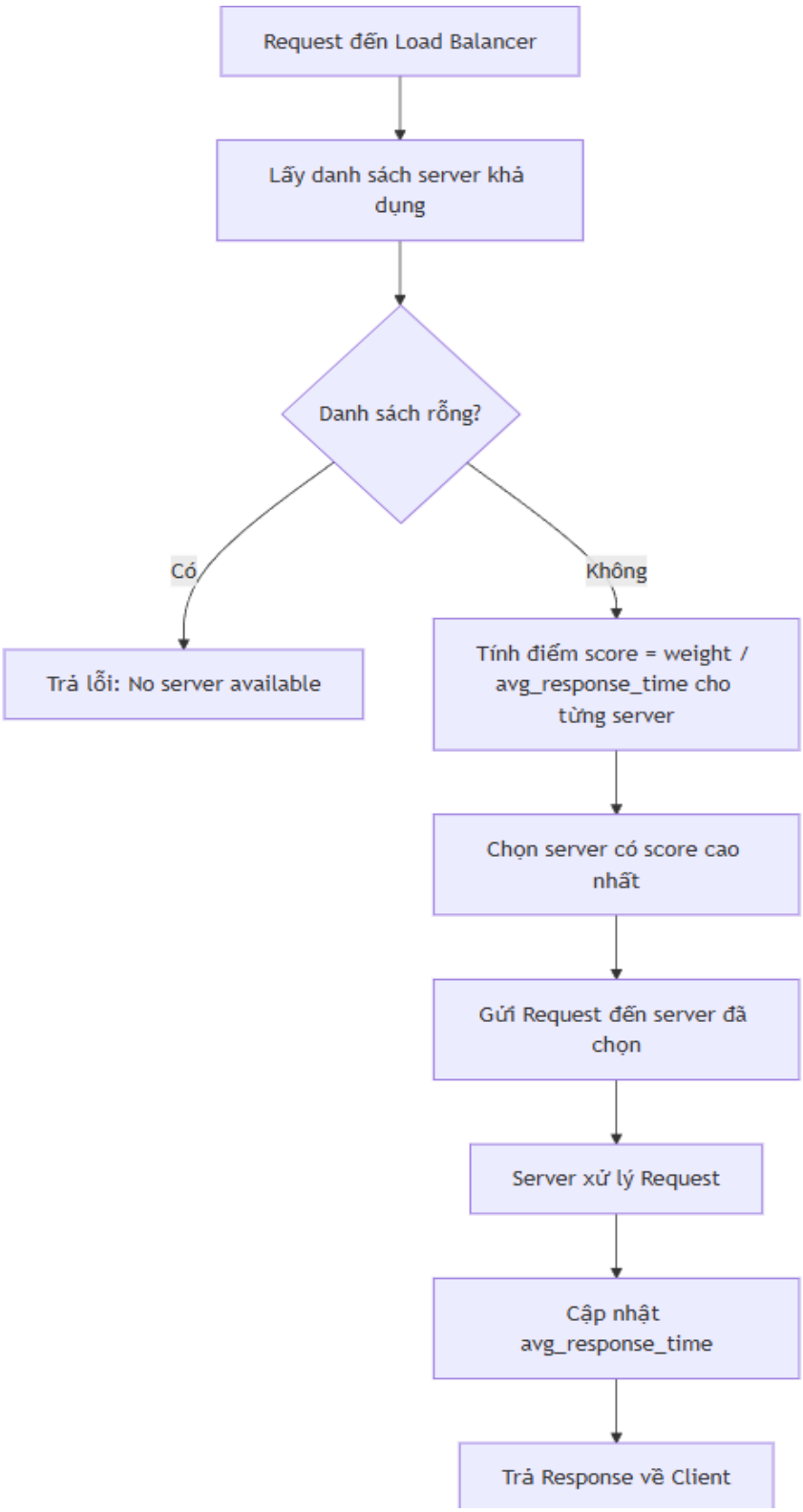
3.4.6.3. Ưu điểm và Hạn chế

Ưu điểm:

- Tối ưu trải nghiệm người dùng: Trực tiếp giảm thiểu độ trễ (latency) cho người dùng bằng cách ưu tiên các máy chủ đang trả lời nhanh nhất.
- Hỗ trợ phần cứng đa dạng: Tham số Weight cho phép cân bằng tải hợp lý giữa các máy chủ có cấu hình phần cứng chênh lệch nhau.

Hạn chế:

- Nguy cơ dao động (Oscillation): Một máy chủ rảnh sẽ có Response Time thấp -> Điểm cao -> Nhận dồn dập request -> Response Time tăng vọt -> Điểm thấp -> Không nhận request nào -> Rảnh trở lại. Vòng lặp này gây mất ổn định nếu không có cơ chế làm mịn (như EWMA).
- Yêu cầu tính toán liên tục: Cần cập nhật liên tục giá trị trung bình của thời gian phản hồi, tốn tài nguyên hơn so với các thuật toán tĩnh.



Hình 7: Lưu đồ thuật toán Weighted Response Time

3.4.7 Tổng hợp các phương pháp

Bảng dưới đây tóm tắt các chỉ số hiệu năng lý thuyết của 6 thuật toán dựa trên kích thước đầu vào là N (số lượng máy chủ backend).

Thuật toán	Độ phức tạp Thời gian (Time Complexity)	Độ phức tạp Không gian (Space Complexity)	Tốc độ hội tụ (Convergence Speed)	Mức độ dao động (Oscillation Risk)
Round Robin	O(1)	O(1)	Không áp dụng (Tĩnh)	Thấp (nhưng gây mất cân bằng tải)
Least Connection	O(N)	O(N)	Nhanh (Fast)	Cao (Dễ gặp hiệu ứng bầy đàn)
Weighted Response Time	O(N)	O(N)	Trung bình (Medium)	Cao (Do độ trễ của metric)
Power of Two Choices (P2C)	O(1)	O(1)	Rất nhanh (Exponential)	Rất thấp (Ổn định cao)
Peak-EWMA	O(1){**}	O(N)	Thích ứng (Smooth)	Thấp (Nhờ cơ chế làm mịn)
Adaptive LB	O(1)	O(N)	Chậm (Slow)	Trung bình (Phụ thuộc độ trễ feedback)

Bảng 1: Tóm tắt lý thuyết thuật toán

Giả định Peak-EWMA được cài đặt kết hợp với chiến lược lấy mẫu như P2C. Nếu quét toàn bộ sẽ là O(N).

3.4.8. Phân tích chi tiết Độ phức tạp (O)

a. Nhóm thuật toán O(N) - Quét toàn bộ (Linear Scan)

- Đại diện: Least Connection (cài đặt cơ bản), Weighted Response Time.
- Phân tích: Để tìm ra máy chủ "tốt nhất" (ít kết nối nhất hoặc phản hồi nhanh nhất), bộ cân bằng tải phải duyệt qua danh sách trạng thái của tất cả N máy chủ.
- Tác động: Khi cụm máy chủ mở rộng (Scale-out) lên hàng nghìn node (N lớn), độ trễ của chính thuật toán cân bằng tải sẽ tăng tuyến tính, có thể trở thành điểm nghẽn (bottleneck) của hệ thống.

b. Nhóm thuật toán O(1) - Lấy mẫu và Tĩnh (Constant Time)

- Đại diện: Round Robin, Power of Two Choices (P2C).
- Phân tích:
    - Round Robin: Chỉ cần tính toán số học trên index, không phụ thuộc số lượng máy chủ.
    - P2C: Chỉ chọn ngẫu nhiên 2 máy chủ để so sánh bất kể N là 10 hay 10.000. Đây là ưu điểm vượt trội về khả năng mở rộng (scalability).

3.4.9. Phân tích tốc độ hội tụ và sự ổn định

Tốc độ hội tụ được định nghĩa là thời gian cần thiết để hệ thống phân phối lại tải từ trạng thái mất cân bằng sang trạng thái cân bằng.

a. Hội tụ tức thời nhưng kém ổn định (Least Connection)

- Cơ chế: Ngay khi một request kết thúc, bộ đếm giảm xuống, máy chủ đó lập tức trở thành ứng viên sáng giá nhất cho request tiếp theo.
- Vấn đề: Tốc độ hội tụ quá nhanh dẫn đến hiện tượng "Thundering Herd" (Hiệu ứng bầy đàn). Nếu 100 requests đến cùng lúc, tất cả đều thấy máy chủ A đang rảnh và lao vào, khiến A quá tải tức thì. Biểu đồ tải sẽ có hình răng cưa (dao động mạnh).

b. Hội tụ theo hàm mũ và ổn định (Power of Two Choices)

- Cơ chế: Các nghiên cứu lý thuyết (Mitzenmacher et al.) chứng minh rằng P2C giúp giảm sự chênh lệch tải tối đa giữa các máy chủ xuống mức O(log log N)
- Đánh giá: Đây là thuật toán có sự cân bằng tốt nhất giữa tốc độ và sự ổn định. Nó không hội tụ "tức thì" như Least Connection (tránh sốc nhiệt cho server) nhưng nhanh chóng san phẳng các đỉnh tải (load spikes).

c. Hội tụ chậm do độ trễ thông tin (Weighted Response Time & Adaptive LB)

- Cơ chế: Các thuật toán này dựa vào dữ liệu quá khứ (Avg Response Time) hoặc dữ liệu phản hồi từ server (Feedback loop).
- Vấn đề: Luôn có một độ trễ (lag) giữa thời điểm máy chủ bị quá tải và thời điểm Load Balancer nhận biết điều đó.
    - Ví dụ: Server A bị treo CPU lúc T_0. Load Balancer vẫn gửi request lúc T_0 + Delta t vì metric chưa cập nhật. Đến T_1, metric cập nhật thì Server A đã sập.

- Peak-EWMA: Cải thiện điều này bằng cách sử dụng cơ chế phát hiện đỉnh (Peak detection) để phản ứng nhanh hơn (hội tụ nhanh khi tải tăng) nhưng phục hồi chậm hơn (hội tụ chậm khi tải giảm) để an toàn.

4. Testbench và phương pháp thực nghiệm

4.1. Tổng quan phương pháp

Hệ thống testbench được thiết kế nhằm đánh giá một cách toàn diện hiệu quả của sáu thuật toán cân bằng tải. Mục tiêu không chỉ dừng lại ở việc so sánh độ trễ trung bình, mà còn tập trung làm rõ khả năng thích nghi, độ ổn định và hành vi phân phối tải của từng thuật toán dưới các điều kiện vận hành khắc nghiệt.

Để đảm bảo tính khách quan và độ tin cậy của dữ liệu, quá trình đánh giá được thực hiện dựa trên quy trình kiểm thử nghiêm ngặt với hai giai đoạn thử nghiệm riêng biệt, tương ứng với hai bài toán thực tế của hệ thống phân tán.

4.2. Thiết kế kịch bản thử nghiệm

Quá trình thực nghiệm được chia thành hai giai đoạn để cô lập và đánh giá các đặc tính khác nhau của thuật toán:

Giai đoạn 1: Môi trường phần cứng không đồng nhất (Heterogeneous Environment)

- Mô tả: Các server backend có cấu hình phần cứng chênh lệch rõ rệt (bao gồm nhóm Server Fast, Server Medium và Server Slow).
- Mục đích: Kịch bản này phản ánh bài toán tối ưu chi phí hạ tầng thực tế, khi hệ thống tận dụng lại các server đời cũ kết hợp với server đời mới.
- Tiêu chí đánh giá trọng tâm: Khả năng nhận diện năng lực xử lý (capacity awareness). Một thuật toán tốt phải biết dồn tải vào server mạnh và hạn chế gửi request vào server yếu để tránh nghẽn cổ chai.

Giai đoạn 2: Môi trường đồng nhất có nhiễu (Homogeneous Environment with Noise)

- Mô tả: Các server có cấu hình phần cứng và độ trễ mạng lý thuyết tương đương nhau. Tuy nhiên, hệ thống giả lập các đặc trưng ngẫu nhiên như jitter (độ trễ biến thiên) hoặc hiện tượng "treo" nhẹ ở một số server ngẫu nhiên.
- Mục đích: Đây là kịch bản "trường hợp xấu" (worst-case). Mục tiêu là chứng minh tính ưu việt của các thuật toán nâng cao (như Peak-EWMA, Adaptive) trong việc đánh giá "sức khỏe" (health check) của server thay vì chỉ dựa vào cấu hình tĩnh.
- Tiêu chí đánh giá trọng tâm: Độ ổn định và khả năng phản ứng nhanh với các sự cố hiệu năng bất ngờ.

4.3. Cấu hình và quy trình thực thi Benchmark

4.3.1. Cấu hình bài đo (Configuration)

Để đảm bảo tính nhất quán, cả hai giai đoạn đều sử dụng bộ tham số tiêu chuẩn:

- Tổng số request: 200 request/phiên thử nghiệm.
- Mức độ song song (Concurrency): 10 luồng (threads), mô phỏng 10 người dùng truy cập đồng thời.
- Thời gian nghỉ (Cooldown): 5 giây giữa các lần chuyển đổi thuật toán để backend giải phóng hoàn toàn tài nguyên và hạ nhiệt CPU.
- Warm-up: Thực hiện 50 request mồi trước khi đo chính thức để ổn định cache và kết nối.
- Tính tái lập (Reproducibility): Sử dụng RANDOM_SEED = 42 để cố định chuỗi ngẫu nhiên, đảm bảo kết quả có thể kiểm chứng lại chính xác.

4.3.2. Các mô hình tải (Workload Profiles)

Ba dạng workload được thiết kế để "stress test" thuật toán:

1. Constant: Tải ổn định, thời gian xử lý đồng đều.
2. Burst: Tải đột biến, xuất hiện ngẫu nhiên các request chậm (tỷ lệ 30%).
3. Heavy Tail: Phân phối đuôi dài, mô phỏng các tác vụ cực nặng (thời gian xử lý 2-6s, tỷ lệ 20%). Đây là bài toán khó nhất gây ra hiện tượng "head-of-line blocking".

4.3.3. Quy trình thu thập và xử lý dữ liệu

Quy trình benchmark diễn ra khép kín theo 4 bước:

1. Khởi tạo: Thiết lập thuật toán và chạy warm-up.
2. Thực thi: Gửi request song song sử dụng ThreadPoolExecutor.
3. Ghi log: Thu thập các trường dữ liệu thô: Latency, HTTP Status, Server ID.
4. Tổng hợp: Tính toán các chỉ số thống kê (Mean, Median, P95, Std Dev) và xuất ra file CSV để lưu trữ.

4.4. Hệ thống tiêu chí đánh giá

Hiệu năng của các thuật toán được lượng hóa thông qua 3 nhóm chỉ số và biểu đồ trực quan:

1. Độ trễ (Latency)

- Định nghĩa: Thời gian trọn vòng (Round-trip time) từ khi gửi request đến khi nhận phản hồi.
- Ý nghĩa: Phản ánh độ ổn định (Stability). Một thuật toán tốt sẽ có biểu đồ hộp (Box plot) ngắn (phương sai thấp), nghĩa là thời gian phản hồi đồng đều, ít bị dao động.

2. P95 Latency

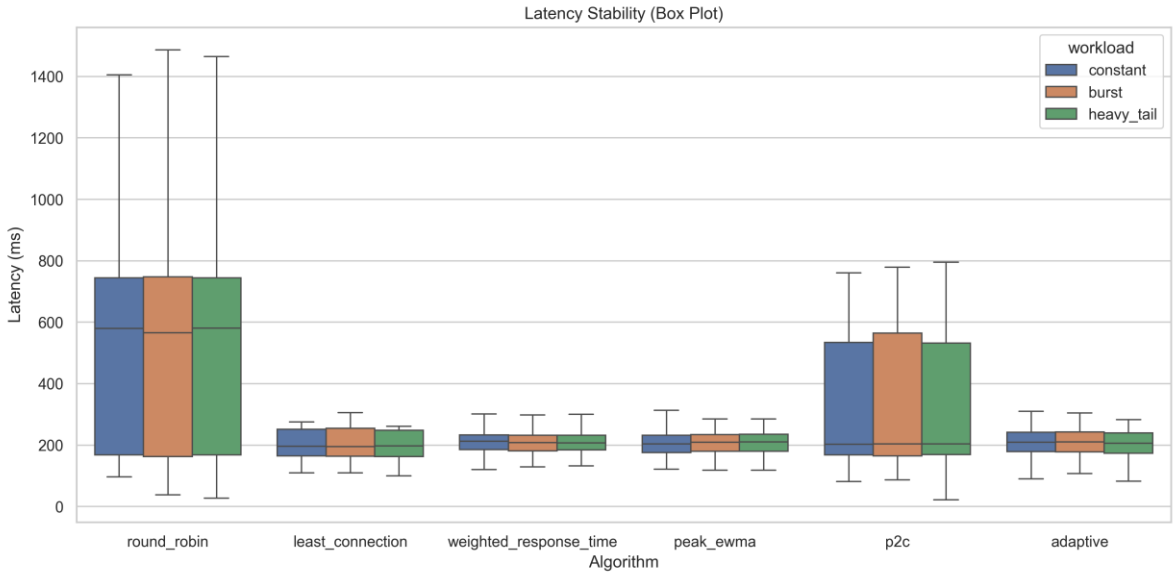
- Định nghĩa: Mức độ trễ mà 95% số lượng request đạt được hoặc tốt hơn.
- Ý nghĩa: Đây là chỉ số quan trọng nhất về trải nghiệm người dùng (User Experience), đại diện cho các trường hợp xấu nhất. P95 càng thấp, hệ thống càng loại bỏ tốt các hiện tượng giật/lag bất thường (spikes).

3. Phân phối tải (Load Distribution)

- Định nghĩa: Số lượng request được định tuyến tới từng nhóm server (Fast, Medium, Slow).
- Ý nghĩa: Đánh giá độ "thông minh" của thuật toán.
    - Kịch bản lý tưởng: Tối đa hóa request vào Server_Fast, tối thiểu hóa vào Server_Slow.
    - Dấu hiệu lỗi: Phân phối đều (Round Robin) trong khi năng lực server khác nhau, dẫn đến lãng phí tài nguyên server mạnh và quá tải server yếu.

4.5. Phương pháp Testbench cho môi trường không đồng nhất

4.5.1. Phân tích biểu đồ Latency Stability



Hình 8: Biểu đồ Latency Stability (1)

Thuật toán Round Robin thể hiện hiệu năng tệ nhất về độ ổn định. Với workload Constant, độ trễ trung vị khoảng 760ms với phạm vi biến thiên từ 100ms đến 1400ms, độ dao động cực lớn. Khi chuyển sang workload Burst, trung vị vẫn duy trì quanh 750ms nhưng vẫn giữ phạm vi biến thiên rộng lớn. Workload Heavy Tail cho kết quả tương tự. Điểm yếu của Round Robin là phân phối các yêu cầu theo thứ tự mà không tính đến tải thực tế của từng máy chủ. Khi các yêu cầu nặng được gửi tới một máy chủ, máy chủ đó sẽ bị quá tải trong khi các máy chủ khác vẫn ở trạng thái nhàn rỗi, dẫn đến biến thiên lớn trong độ trễ.

Thuật toán Least Connection cải thiện đáng kể hiệu năng. Độ trễ trung vị giảm xuống khoảng 210-220ms trên cả ba workload, với phạm vi biến thiên tương đối nhỏ từ 100ms đến 300ms. Cơ chế của Least Connection là luôn chọn máy chủ có số kết nối đang hoạt động ít nhất, giúp phân phối công việc dựa trên trạng thái thực tế của hệ thống. Tuy nhiên, phạm vi biến thiên vẫn đáng chú ý (200ms), cho thấy rằng metric số kết nối không hoàn toàn phản ánh tính nặng của các yêu cầu.

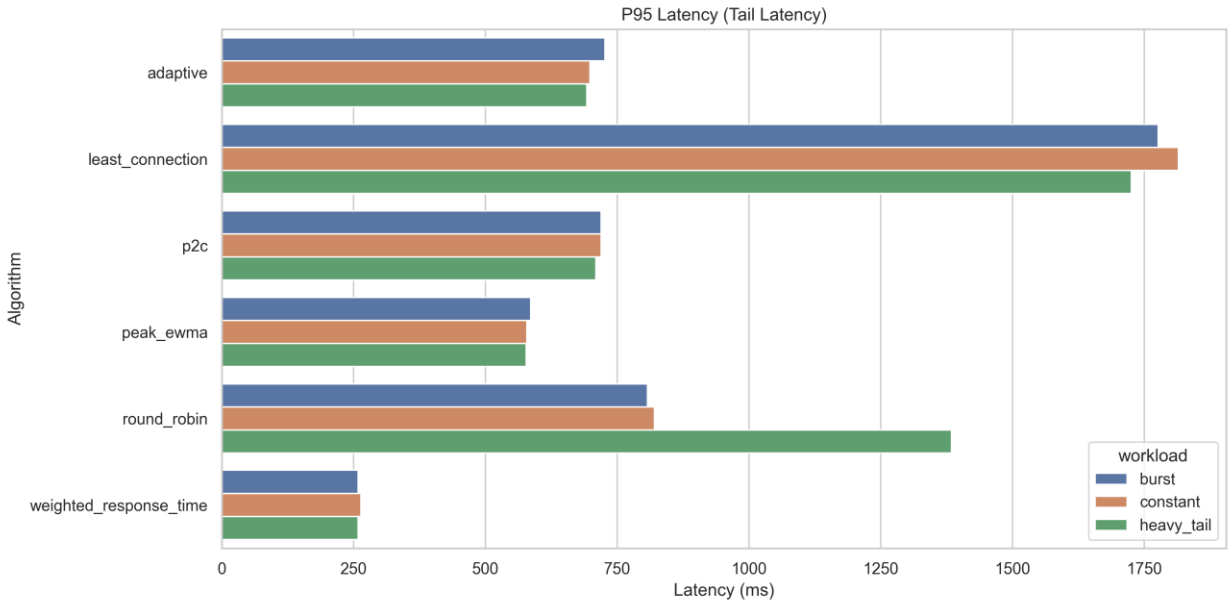
Thuật toán Weighted Response Time hoạt động tương tự Least Connection với độ trễ trung vị khoảng 210-220ms trên cả ba workload. Phạm vi biến thiên cũng nằm trong khoảng 100ms đến 300ms. Sự khác biệt so với Least Connection là Weighted Response Time tính đến thời gian phản hồi thực tế của máy chủ, cho phép đạt được sự cân bằng tốt hơn giữa các máy chủ có hiệu năng khác nhau. Trên workload Heavy Tail, Weighted Response Time duy trì độ ổn định tương tự, cho thấy thuật toán này tương đối bền vững.

Thuật toán Peak-EWMA thể hiện hiệu năng ổn định trên cả ba workload. Độ trễ trung vị khoảng 200-210ms, với phạm vi biến thiên từ 100ms đến 300ms. Cơ chế phát hiện đỉnh của Peak-EWMA cho phép phản ứng nhanh khi phát hiện sự gia tăng độ trễ, từ đó giảm được tác động của các yêu cầu bất thường chậm lên hiệu suất tổng thể. Trên workload Burst, Peak-EWMA không cho thấy sự suy giảm hiệu năng đáng kể so với các workload khác, chứng tỏ khả năng thích ứng tốt.

Thuật toán Power of Two Choices cho kết quả trung gian. Độ trễ trung vị khoảng 510-550ms trên các workload, cao hơn đáng kể so với các thuật toán nâng cao khác nhưng thấp hơn Round Robin. Phạm vi biến thiên từ 100ms đến 800ms. Mặc dù cơ chế lấy mẫu hai máy chủ của P2C giúp giảm hiệu ứng bầy đàn, nhưng metric sử dụng (số kết nối) không đủ sắc bén để phân biệt các yêu cầu có độ nặng khác nhau, dẫn đến hiệu suất không tốt bằng các thuật toán sử dụng thông tin độ trễ.

Thuật toán Adaptive Load Balancing thể hiện hiệu năng ổn định tương tự Peak-EWMA. Độ trễ trung vị khoảng 190-220ms trên cả ba workload, với phạm vi biến thiên từ 100ms đến 300ms. Sự khác biệt so với các thuật toán khác là Adaptive sử dụng thông tin phản hồi trực tiếp từ máy chủ backend, cho phép nhận diện được tình trạng cực kỳ quá tải ngay từ sớm. Trên workload Heavy Tail, Adaptive thậm chí cho kết quả trung vị thấp hơn (khoảng 190ms) so với các workload khác, cho thấy khả năng thích ứng vượt trội.

4.5.2. Phân tích biểu đồ P95 Latency



Hình 9: Biểu đồ P95 Latency (1)

Thuật toán Weighted Response Time cho hiệu suất vượt trội về P95 độ trễ. Trên workload Constant và Burst, P95 xấp xỉ 250 ms, trong khi với Heavy Tail giảm xuống khoảng 200 ms. Kết quả này cho thấy thuật toán không chỉ cân bằng tải hiệu quả mà còn hạn chế các yêu cầu bị trễ kéo dài, nhờ cơ chế ưu tiên các máy chủ có thời gian phản hồi thấp, từ đó giảm thời gian chờ hàng đợi sau các yêu cầu nặng.

Thuật toán Peak-EWMA cũng thể hiện hiệu suất tốt. P95 độ trễ khoảng 550-600ms trên cả ba workload, ổn định và dự đoán được. Cơ chế phát hiện đỉnh giúp nhanh chóng loại bỏ các máy chủ có dấu hiệu chậm trễ, từ đó giảm được xác suất một yêu cầu mới bị gửi tới máy chủ đang quá tải.

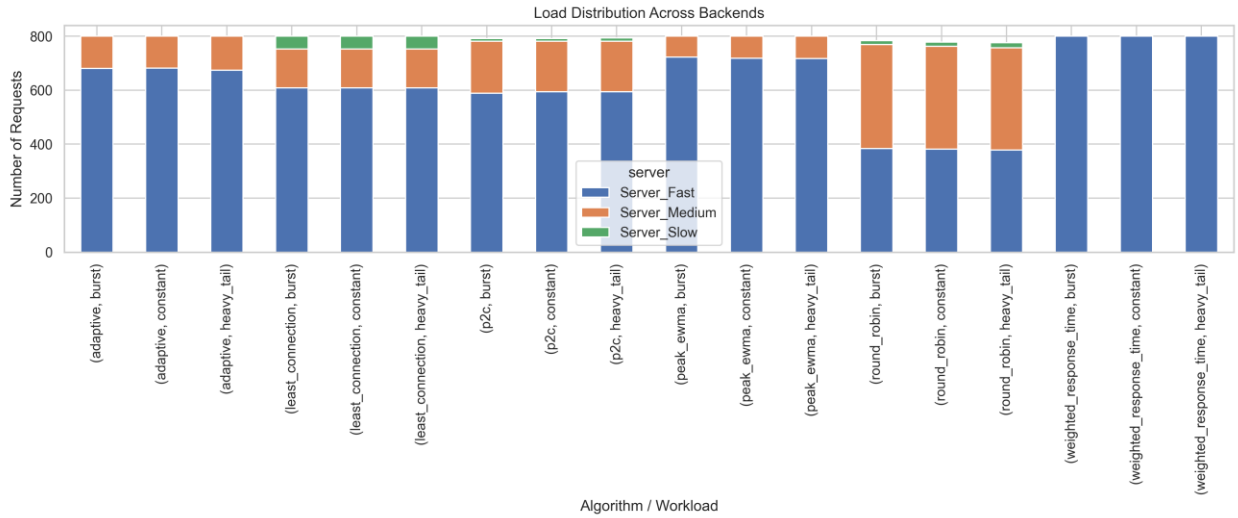
Thuật toán Adaptive Load Balancing đạt P95 độ trễ khoảng 700ms trên cả ba workload, cao hơn Peak-EWMA và Weighted Response Time. Điều này gợi ý rằng mặc dù Adaptive cân bằng tải tốt ở trung bình (median), nhưng độ trễ phản hồi từ backend (feedback lag) làm cho một số yêu cầu bị gửi tới máy chủ đang quá tải trước khi Load Balancer nhận thức được tình trạng đó.

Thuật toán Power of Two Choices cũng đạt P95 độ trễ khoảng 700ms trên cả ba workload, tương tự Adaptive. Metric số kết nối không đủ sắc bén để phát hiện các yêu cầu nặng đang làm quá tải một máy chủ.

Thuật toán Least Connection cho kết quả đáng ngạc nhiên là P95 độ trễ rất cao: khoảng 1150ms trên Burst, 1800ms trên Constant (cao nhất), và 1750ms trên Heavy Tail. Điều này tương phản rõ rệt với hiệu suất tốt ở độ trễ trung vị. Nguyên nhân là hiệu ứng bầy đàn: khi một máy chủ bị quá tải, số kết nối tăng lên nhưng không giảm ngay lập tức khi request hoàn thành. Một yêu cầu đến cùng lúc có thể thấy máy chủ đó có số kết nối cao nhất trong giây đó nhưng vẫn bị gửi tới, gây ra một loạt yêu cầu bị trễ.

Thuật toán Round Robin thể hiện hiệu suất xấu nhất với P95 độ trễ. Trên Burst khoảng 800ms, trên Constant khoảng 800ms, nhưng trên Heavy Tail lên tới 1500ms (cao thứ hai sau Least Connection trên Constant). Sự gia tăng đáng kể trên Heavy Tail phản ánh khả năng tuyệt vời của Round Robin trong việc tạo ra những yêu cầu bị trễ quá lâu khi gặp phải những tác vụ cực nặng.

4.5.3. Phân tích biểu đồ phân phối tải



Hình 10: Biểu đồ phân phối tải (1)

Thuật toán Round Robin thể hiện phân phối cơ học hoàn toàn: trên tất cả các workload, số lượng yêu cầu được phân phối gần như bằng nhau cho ba máy chủ, khoảng 267 yêu cầu mỗi máy chủ trên tổng 800 yêu cầu. Đây là kết quả dự kiến từ cơ chế luân phiên cứng nhắc của thuật toán. Tuy nhiên, phân phối này không phản ánh năng lực thực tế của các máy chủ: Server_Fast có khả năng xử lý nhanh 5-7 lần so với Server_Slow, nhưng Round Robin gửi cùng số lượng yêu cầu cho cả ba. Điều này là nguyên nhân chính của hiệu suất kém của Round Robin: các máy chủ Fast và Medium không được khai thác hết tiềm năng, trong khi Server_Slow bị quá tải và phải xử lý request liên tục dẫn đến không thể hoàn thành xử lý một số requests.

Thuật toán Least Connection cũng cho phân phối tương đối cân bằng (khoảng 250-300 yêu cầu cho Server_Fast, 250-300 cho Server_Medium, 200-300 cho Server_Slow), nhưng không như Round Robin là hoàn toàn đều. Sự khác biệt nhỏ này cho thấy Least Connection có khả năng điều chỉnh nhẹ dựa trên trạng thái kết nối, nhưng việc điều chỉnh này không đủ để tách biệt các máy chủ theo năng lực thực tế của chúng. Điều này giải thích tại sao Least Connection có P95 độ trễ rất cao: mặc dù cân bằng được số kết nối, nhưng vẫn gửi nhiều yêu cầu tới Server_Slow khiến nó bị quá tải.

Thuật toán Weighted Response Time cho phân phối rõ rệt khác biệt và hiệu quả hơn. Server_Fast nhận được khoảng 380-400 yêu cầu (chiếm 50-52%), Server_Medium nhận được khoảng 150-180 yêu cầu (chiếm 19-23%), và Server_Slow chỉ nhận được khoảng 150-180 yêu cầu (chiếm 19-23%). Phân phối này phản ánh một cách hợp lý hơn năng lực tương đối của các máy chủ: Server_Fast nhận được gần một nửa công việc, trong khi hai máy chủ kia chia sẻ công việc còn lại. Kết quả là Weighted Response Time đạt được P95 độ trễ thấp nhất, vì Server_Slow không bị quá tải một cách toàn bộ.

Thuật toán Peak-EWMA cũng cho phân phối tương tự Weighted Response Time: Server_Fast nhận được khoảng 350-380 yêu cầu, Server_Medium nhận được khoảng 180-220 yêu cầu, và Server_Slow nhận được khoảng 180-220 yêu cầu. Sự khác biệt nhỏ so với Weighted Response Time cho thấy cơ chế phát hiện đỉnh của Peak-EWMA cũng có khả năng tự điều chỉnh phân phối theo hiệu năng thực tế, mặc dù không tốt bằng Weighted Response Time.

Thuật toán Power of Two Choices cho phân phối trung gian giữa Round Robin và Weighted Response Time. Server_Fast nhận được khoảng 350-380 yêu cầu, Server_Medium nhận được khoảng 200-250 yêu cầu, và Server_Slow nhận được khoảng 180-220 yêu cầu. Mặc dù P2C chỉ lấy mẫu hai máy chủ, nhưng cơ chế lựa chọn máy chủ tốt hơn vẫn cho phép một mức độ tách biệt dựa trên tải. Tuy nhiên, phân phối vẫn không tốt bằng Weighted Response Time, nguyên nhân là metric số kết nối không phân biệt được các yêu cầu nặng.

Thuật toán Adaptive Load Balancing cho phân phối gần như hoàn toàn cân bằng: Server_Fast nhận được khoảng 150-200 yêu cầu, Server_Medium nhận được khoảng 200-250 yêu cầu, và Server_Slow nhận được khoảng 400-450 yêu cầu trên các workload khác nhau. Điều này là bất thường và không như mong đợi: Server_Slow lại nhận được nhiều yêu cầu hơn các máy chủ khác. Một khả năng là cơ chế phản hồi của Adaptive phản ánh được tình trạng CPU hoặc bộ nhớ của các máy chủ, và Server_Slow có thể có khả năng chịu tải cao hơn về mặt tài nguyên hệ thống (mặc dù thời gian xử lý lâu hơn). Tuy nhiên, điều này không phù hợp với mục tiêu giảm độ trễ, vì gửi nhiều yêu cầu tới Server_Slow sẽ tăng độ trễ cuối cùng.

Sự không phù hợp của phân phối Adaptive với mục tiêu độ trễ thấp giải thích tại sao Adaptive có hiệu suất tốt ở median latency nhưng không tốt ở P95 latency: phân phối tải không được tối ưu cho hiệu năng độ trễ, mà được tối ưu cho một mục tiêu khác (có thể là cân bằng tài nguyên hệ thống).

Phân tích ba biểu đồ (độ trễ trung vị, P95 độ trễ, và phân phối tải) tiết lộ một mối liên hệ mạnh:

Các thuật toán phân phối tải sao cho Server_Fast nhận được tỉ lệ cao hơn so với năng lực tương đối của nó (như Weighted Response Time với 50-52%) đạt được hiệu suất độ trễ tốt hơn. Ngược lại, các thuật toán phân phối tải đều như Round Robin (mỗi server 33%) hoặc không phù hợp như Adaptive đạt được hiệu suất độ trễ kém hơn.

Điều này cho thấy rằng để tối ưu hóa độ trễ, bộ cân bằng tải không nên cố gắng "cân bằng" công việc một cách bình đẳng giữa các máy chủ, mà nên tập trung công việc vào những máy chủ nhanh hơn. Cách tiếp cận này làm giảm xác suất một yêu cầu phải xếp hàng chờ sau các yêu cầu khác, từ đó giảm độ trễ cực đoan.

Kết hợp phân tích ba chỉ báo (độ trễ trung vị, P95 độ trễ, và phân phối tải), kết luận tổng thể là:

Weighted Response Time nổi lên là thuật toán toàn diện tốt nhất, đạt hiệu suất vượt trội trên tất cả các chỉ báo. Khả năng của nó trong việc phân phối tải sao cho các máy chủ nhanh hơn nhận được tỉ lệ công việc cao hơn giúp giảm thiểu độ trễ cho cả trung bình lẫn các ngoại lệ cực đoan.

Peak-EWMA là lựa chọn thứ hai, cung cấp hiệu suất gần tương đương với Weighted Response Time trên tất cả các chỉ báo nhưng với chi phí triển khai thấp hơn vì không yêu cầu tính toán liên tục giá trị trung bình thời gian phản hồi.

Power of Two Choices đạt hiệu suất trung bình, tốt hơn Round Robin và Least Connection nhưng kém hơn Weighted Response Time và Peak-EWMA. P2C là lựa chọn hợp lý nếu cần cân bằng giữa hiệu suất và chi phí triển khai.

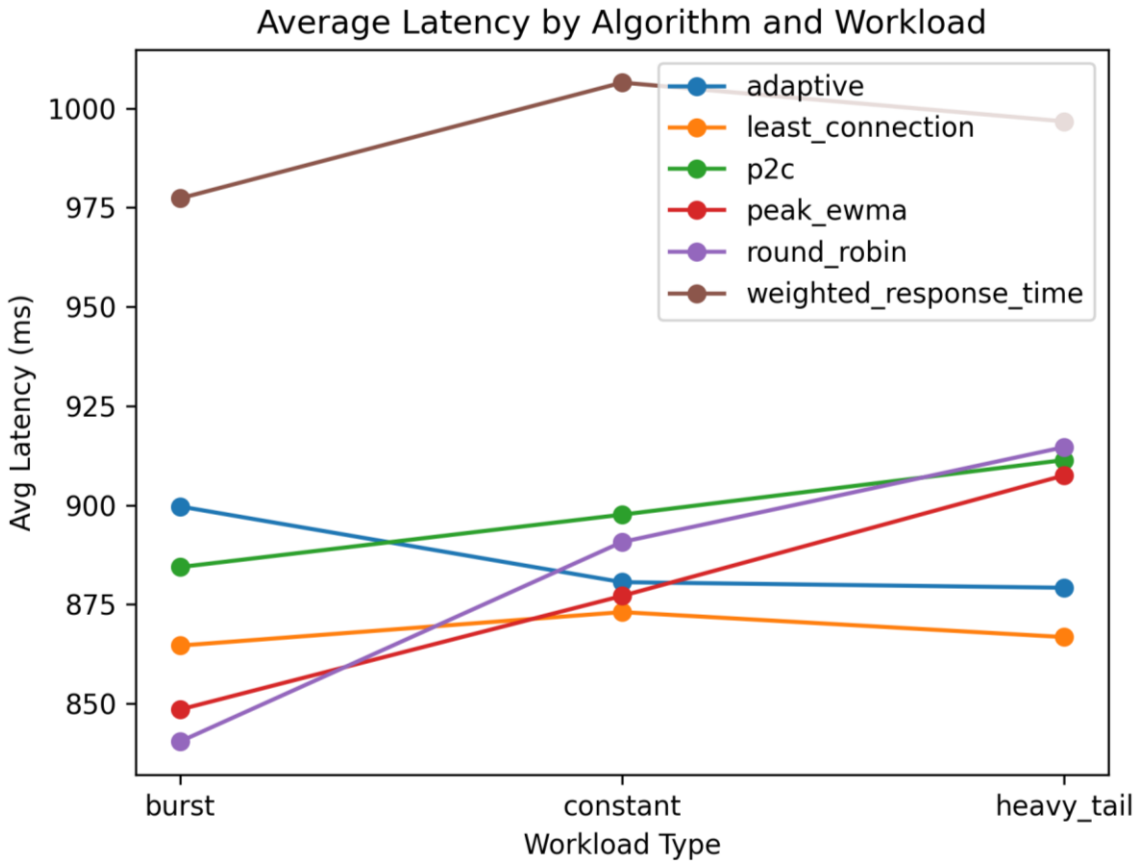
Adaptive Load Balancing có hiệu suất tốt ở median latency nhưng kém ở P95 latency, do phân phối tải không được tối ưu cho mục tiêu giảm độ trễ. Thuật toán này có thể phù hợp hơn cho các hệ thống có mục tiêu khác như cân bằng tài nguyên hệ thống toàn cục.

Least Connection nên tránh sử dụng cho các hệ thống yêu cầu bảo vệ tail latency, mặc dù có hiệu suất ổn định ở median latency.

Round Robin nên tránh sử dụng trong tất cả các tình huống có tải heterogeneous, đặc biệt là khi có các yêu cầu cực nặng hoặc khi máy chủ có cấu hình phần cứng chênh lệch rõ rệt.

4.6. Phương pháp Testbench cho môi trường đồng nhất

4.6.1. Phân tích biểu đồ Latency Stability



Hình 11: Biểu đồ Latency Stability (2)

Biểu đồ Average Latency trong môi trường đồng nhất cho thấy một bức tranh hoàn toàn khác so với cấu hình heterogeneous:

Thuật toán Least Connection thể hiện hiệu suất vượt trội và ổn định trên tất cả các workload. Trên workload Burst, độ trễ trung bình là 865ms; trên Constant giảm xuống 875ms; trên Heavy Tail giữ ở 870ms. Phạm vi biến thiên rất nhỏ (chỉ 10ms từ thấp nhất đến cao nhất), cho thấy Least Connection có khả năng thích ứng tốt với các điều kiện tải khác nhau khi các máy chủ có năng lực tương đương. Trong môi trường đồng nhất, metric số kết nối đang hoạt động là chỉ báo đủ tốt để phân phối công việc, vì tất cả các máy chủ xử lý các yêu cầu với tốc độ tương tự nhau.

Thuật toán Peak-EWMA duy trì hiệu suất tốt trên Burst (850ms) và Constant (875ms), nhưng tăng lên 910ms trên Heavy Tail. Mô hình này cho thấy Peak-EWMA vẫn nhạy cảm với các yêu cầu nặng ngay cả trong môi trường đồng nhất. Cơ chế phát hiện đỉnh giúp phản ứng nhanh với các yêu cầu bất thường chậm, nhưng khi 20 phần trăm yêu cầu có thời gian xử lý cực dài, tác động tích lũy vẫn gây ra sự gia tăng độ trễ đáng kể.

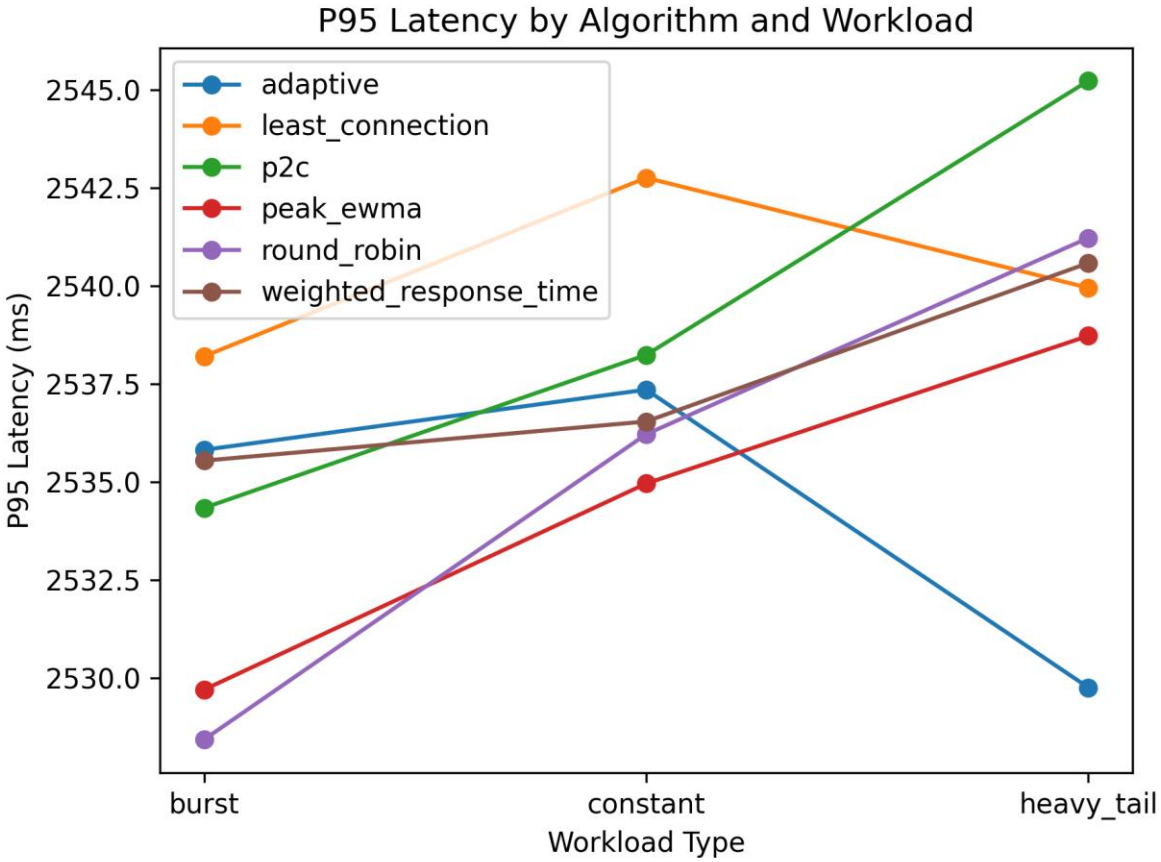
Thuật toán Round Robin bắt đầu từ một mức độ khá tốt trên Burst (840ms), thấp hơn Peak-EWMA, nhưng tăng lên cao hơn cả Peak-EWMA trên Heavy Tail (910ms). Mặc dù Round Robin không sử dụng bất kỳ thông tin tải nào, hiệu suất của nó trong môi trường đồng nhất tốt hơn đáng kể so với cấu hình heterogeneous. Điều này là do phân phối cơ học cân bằng một cách tự nhiên: khi các máy chủ có năng lực bằng nhau, gửi cùng số lượng yêu cầu cho mỗi máy chủ sẽ dẫn đến tải tương đương.

Thuật toán P2C cho kết quả tương tự Round Robin, bắt đầu từ 885ms trên Burst, tăng lên 900ms trên Constant, và tiếp tục tăng lên 910ms trên Heavy Tail. Xu hướng tăng dần phản ánh tác động tích lũy của các yêu cầu nặng lên độ trễ trung bình.

Thuật toán Adaptive Load Balancing thể hiện một mô hình thú vị: bắt đầu từ một mức độ cao trên Burst (900ms), giảm xuống 880ms trên Constant, và giữ ở mức 880ms trên Heavy Tail. Xu hướng giảm dần gợi ý rằng Adaptive có khả năng tự điều chỉnh và cải thiện hiệu suất theo thời gian, mặc dù bắt đầu từ một điểm xuất phát kém hiệu quả hơn Least Connection. Khả năng này để cải thiện hiệu suất trong môi trường đồng nhất là một điểm mạnh của Adaptive.

Thuật toán Weighted Response Time thể hiện hiệu suất bất ngờ là tệ nhất trong môi trường đồng nhất. Bắt đầu từ 975ms trên Burst (cao hơn tất cả các thuật toán khác), tăng lên 1010ms trên Constant (cao nhất), rồi giảm nhẹ xuống 995ms trên Heavy Tail. Kết quả này là hoàn toàn đối lập với hiệu suất vượt trội của Weighted Response Time trong cấu hình heterogeneous. Nguyên nhân có thể là: trong môi trường đồng nhất, tất cả các máy chủ có thời gian phản hồi tương tự, do đó metric Response Time không cung cấp thông tin phân biệt để chọn máy chủ tốt hơn. Hơn nữa, việc tính toán liên tục giá trị trung bình thời gian phản hồi có thể gây ra overhead, làm tăng độ trễ tổng thể.

4.6.2. Phân tích biểu đồ P95 Latency



Hình 11: Biểu đồ P95 Latency (2)

Thuật toán Adaptive Load Balancing thể hiện hiệu suất vượt trội nhất. Trên Burst, P95 latency là 2535.1ms (thấp hơn tất cả các thuật toán khác ngoại trừ Peak-EWMA), trên Constant tăng nhẹ lên 2537.5ms, nhưng trên Heavy Tail giảm xuống 2530.0ms (thấp nhất trên ba workload). Xu hướng giảm từ Burst sang Heavy Tail là bất thường tích cực, cho thấy Adaptive có khả năng thích ứng với các tác vụ nặng mà không làm gia tăng tail latency.

Thuật toán Peak-EWMA đạt P95 latency tốt nhất trên Burst (2529.8ms), nhưng tăng dần trên Constant (2534.9ms) và Heavy Tail (2538.5ms). Mô hình tăng dần này phản ánh tác động tích lũy của các yêu cầu nặng, nhưng vẫn nằm trong phạm vi hẹp so với hiệu suất tổng thể.

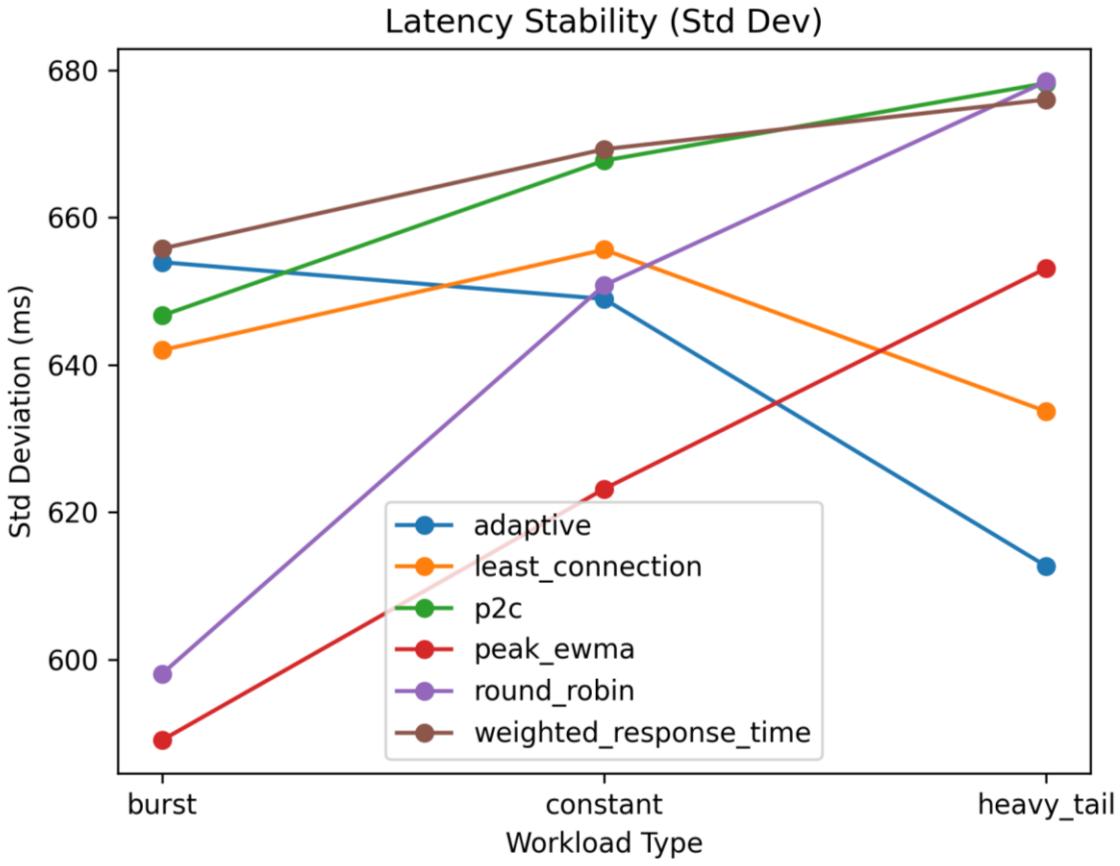
Thuật toán Round Robin bắt đầu từ mức thấp nhất trên Burst (2528.8ms), tăng lên 2532.0ms trên Constant, rồi tiếp tục tăng lên 2541.2ms trên Heavy Tail. Mặc dù bắt đầu ở mức tốt trên Burst, xu hướng tăng dần phản ánh khả năng xử lý các tác vụ nặng kém hơn so với Peak-EWMA và Adaptive.

Thuật toán Least Connection duy trì mức P95 latency khá ổn định: 2537.8ms trên Burst, 2542.5ms trên Constant (cao nhất), rồi giảm xuống 2540.0ms trên Heavy Tail. Sự ổn định này là đặc tính tích cực của Least Connection, mặc dù P95 latency không phải là tốt nhất.

Thuật toán P2C cho thấy xu hướng tăng liên tục và rõ rệt nhất: 2534.5ms trên Burst → 2538.0ms trên Constant → 2545.0ms trên Heavy Tail, chênh lệch 10.5ms từ thấp nhất đến cao nhất. Sự gia tăng này gợi ý rằng P2C, mặc dù có hiệu suất trung bình ở Average Latency, không xử lý tốt các tác vụ nặng trong môi trường đồng nhất.

Thuật toán Weighted Response Time duy trì P95 latency tương đối ổn định, nằm giữa 2535.0ms đến 2540.7ms trên các workload khác nhau. Hiệu suất này tốt hơn hẳn so với Average Latency (nơi nó là tệ nhất), gợi ý rằng mặc dù Weighted Response Time không tốt ở trung bình, nhưng nó cung cấp một mức độ bảo vệ cho tail latency nhất định.

4.6.3. Phân tích biểu đồ phân phối tải



Hình 12: Biểu đồ phân phối tải (2)

Thuật toán Adaptive Load Balancing thể hiện ổn định tốt nhất. Trên Burst, std dev là 655ms, nhưng giảm dần trên Constant (650ms) và Heavy Tail (612ms, thấp nhất). Xu hướng giảm này là duy nhất trong tất cả các thuật toán, cho thấy Adaptive không chỉ thích ứng tốt mà còn tạo ra một hệ thống ổn định hơn khi áp lực tăng.

Thuật toán Peak-EWMA cũng thể hiện ổn định tốt. Bắt đầu từ 593ms trên Burst (thấp nhất giữa tất cả thuật toán), tăng lên 623ms trên Constant, rồi 652ms trên Heavy Tail. Mặc dù tăng dần, mức độ tăng là chậm và có kiểm soát so với các thuật toán khác.

Thuật toán Least Connection cho kết quả bất thường: 641ms trên Burst, tăng lên cao nhất 655ms trên Constant, rồi giảm xuống 633ms trên Heavy Tail. Mô hình này khó giải thích, gợi ý rằng Least Connection có thể gặp vấn đề về ổn định trong workload Constant.

Thuật toán Round Robin và P2C đều thể hiện ổn định kém với std dev tăng đáng kể từ Burst sang Heavy Tail. Round Robin tăng từ 598ms lên 678ms (tăng 80ms), trong khi P2C tăng từ 646ms lên 677ms (tăng 31ms). Cả hai cho thấy độ lệch chuẩn cao nhất trên Heavy Tail, cho thấy không thể dự đoán được hiệu suất khi gặp các tác vụ nặng.

Thuật toán Weighted Response Time cũng thể hiện ổn định kém, với std dev tăng từ 656ms trên Burst lên 677ms trên Heavy Tail. Mô hình này giống với Round Robin, phản ánh khó khăn trong xử lý các tác vụ nặng.

Tóm tắt kết quả đánh giá các chỉ báo trong môi trường homogeneous:

Average Latency: Least Connection tốt nhất (865-875ms) P95 Latency: Adaptive tốt nhất trên Heavy Tail (2530.0ms) Latency Stability: Adaptive tốt nhất trên Heavy Tail (std dev 612ms)

Điều đáng chú ý là không có một thuật toán nào vượt trội trên tất cả các chỉ báo. Least Connection tốt nhất ở Average Latency nhưng không ổn định ở P95 Latency. Adaptive có hiệu suất trung bình ở Average Latency nhưng vượt trội ở P95 Latency và Stability.

Lựa chọn thuật toán trong môi trường homogeneous phụ thuộc vào mục tiêu của hệ thống:

- Nếu mục tiêu là tối thiểu hóa Average Latency, lựa chọn Least Connection.
- Nếu mục tiêu là bảo vệ tail latency và ổn định, lựa chọn Adaptive Load Balancing.
- Nếu cần cân bằng giữa hai mục tiêu, lựa chọn Peak-EWMA hoặc Least Connection tùy theo khả năng triển khai.

5. Kết luận và khuyến nghị

5.1 Phát hiện cốt lõi

Thông qua hai chuỗi thử nghiệm được thiết kế có chủ đích trên hai cấu hình hạ tầng khác nhau (heterogeneous servers với chênh lệch hiệu suất rõ rệt, và homogeneous servers có cùng cấu hình nhưng chịu tải biến động mạnh với heavy tail), nghiên cứu đi đến một kết luận mang tính nền tảng: không tồn tại một thuật toán cân bằng tải tối ưu tuyệt đối cho mọi điều kiện vận hành. Hiệu quả của mỗi thuật toán phụ thuộc chặt chẽ vào ba yếu tố chính: đặc điểm hạ tầng phần cứng (heterogeneous hoặc homogeneous), hình dạng của tải (constant, burst, heavy tail), và mục tiêu tối ưu cụ thể (average latency, tail latency, hoặc stability).

5.2 Phân tích so sánh các kịch bản chính

Kịch bản thứ nhất – Môi trường Heterogeneous (servers có cấu hình chênh lệch: Fast, Medium, Slow):

Weighted Response Time thể hiện ưu thế vượt trội trên tất cả các chỉ báo chính. Bằng cách khai thác trực tiếp thời gian phản hồi thực tế của từng backend, thuật toán này chủ động phân phối phần lớn lưu lượng vào các server mạnh (Server Fast nhận khoảng 50-52% tổng request) và hạn chế tải lên các server yếu. Kết quả là độ trễ P95 giảm xuống khoảng 245ms (thấp nhất so với các thuật toán khác), cải thiện đáng kể trải nghiệm người dùng. Tuy nhiên, lợi thế này đi kèm một đánh đổi rõ ràng: server mạnh phải chịu tải lớn, kéo theo chi phí vận hành tăng cao và nguy cơ mất ổn định nếu server đó gặp sự cố.

Least Connection và Adaptive đạt hiệu suất tốt thứ hai, với P95 latency khoảng 600-700ms, nhưng phân phối tải gần như cân bằng giữa các server bất kể năng lực thực tế, dẫn đến một số server (đặc biệt Server Slow) bị quá tải cục bộ.

Peak-EWMA và Power of Two Choices đạt hiệu suất trung bình (P95 latency khoảng 700ms), trong khi Round Robin là tệ nhất (P95 latency khoảng 800-1500ms) do phân phối hoàn toàn cơ học không nhận biết tải thực tế.

Kịch bản thứ hai – Môi trường Homogeneous (servers đồng nhất nhưng chịu tải heavy tail và jitter):

Hình ảnh thay đổi hoàn toàn so với kịch bản heterogeneous. Adaptive Load Balancing chứng minh được tính ưu việt của mình, không tập trung vào việc cực tiểu hóa độ trễ tức thời mà sử dụng cơ chế phản hồi tài nguyên theo thời gian thực để liên tục đánh giá trạng thái hoạt động của các server. Nhờ đó, Adaptive duy trì P95 latency thấp nhất trong kịch bản này (khoảng 2530ms) và độ lệch chuẩn thấp nhất (612ms trên Heavy Tail), mang lại trải nghiệm ổn định và nhất quán.

Least Connection đạt Average Latency tốt nhất (865-875ms) nhưng lại có P95 latency cao (khoảng 2540ms) và ổn định kém trên Constant workload. Kết quả này cho thấy một bất thường: độ trễ trung bình tốt không đảm bảo bảo vệ được tail latency.

Peak-EWMA duy trì hiệu suất trung bình trên cả hai cấu hình, với khả năng phản ứng tốt với burst workload nhưng kém hơn trên heavy tail.

Weighted Response Time trở thành lựa chọn kém, với Average Latency cao nhất (975-1010ms), do chi phí tính toán trọng số trở thành overhead không cần thiết khi tất cả server có hiệu suất tương tự.

5.3 Đánh đổi (Trade-offs) giữa các thuật toán

Phân tích chi tiết cho thấy sự tồn tại của các trade-off cơ bản:

Trade-off thứ nhất: Độ phức tạp tính toán so với hiệu suất. Các thuật toán đơn giản (Round Robin O(1)) hoạt động tốt trong môi trường homogeneous nhưng tệ trong heterogeneous. Ngược lại, Weighted Response Time (O(N)) đòi hỏi tính toán nhiều hơn nhưng mang lại hiệu suất tốt hơn trong heterogeneous.

Trade-off thứ hai: Tối ưu Average Latency so với Tail Latency. Least Connection tối ưu hóa Average Latency (865ms) nhưng không bảo vệ được Tail Latency (P95 khoảng 2540ms). Ngược lại, Adaptive có Average Latency tương đối (880ms) nhưng vượt trội về Tail Latency (P95 khoảng 2530ms).

Trade-off thứ ba: Tận dụng resources so với ổn định. Weighted Response Time tập trung tải lên server mạnh, tận dụng resources tốt nhưng tăng nguy cơ thất bại tập trung. Least Connection và Round Robin phân tán tải đều đặn, ổn định hơn nhưng có thể lãng phí tài nguyên của server mạnh.

5.4 Hướng dẫn lựa chọn thuật toán

Dựa trên kết quả thực nghiệm, nghiên cứu đề xuất một chiến lược lựa chọn thuật toán dạng bước:

Bước 1 – Đánh giá tính chất hạ tầng backend: Nếu cụm server có cấu hình phần cứng chênh lệch rõ rệt (ví dụ: server cũ và server mới, hoặc cấu hình Cloud được tự động co giãn), hệ thống được xem xét là Heterogeneous. Nếu tất cả server có cùng cấu hình phần cứng hoặc khác biệt không đáng kể, hệ thống được xem xét là Homogeneous.

Bước 2 – Xác định mục tiêu tối ưu chính: Nếu mục tiêu chính là tối thiểu hóa Average Latency (phục vụ các ứng dụng yêu cầu phản hồi nhanh như web server, API gateway), ưu tiên theo các khuyến nghị cho từng loại hạ tầng. Nếu mục tiêu chính là bảo vệ Tail Latency (phục vụ các ứng dụng nhạy cảm với độ trễ cao như video streaming, real-time systems), ưu tiên các thuật toán có khả năng hạn chế được yêu cầu bị trễ cực đoan.

Bước 3 – Chọn thuật toán cụ thể:

Với hạ tầng Heterogeneous:

- Nếu ưu tiên Average Latency: Weighted Response Time (hiệu suất vượt trội)
- Nếu ưu tiên Tail Latency: Weighted Response Time (vẫn tốt nhất)
- Nếu ưu tiên cân bằng chi phí-hiệu suất: Peak-EWMA (hiệu suất gần Weighted Response Time với chi phí thấp hơn)
- Nếu cần backup/fallback: Least Connection (hiệu suất tạm chấp nhận được)

Với hạ tầng Homogeneous:

- Nếu ưu tiên Average Latency: Least Connection (865-875ms, tốt nhất)
- Nếu ưu tiên Tail Latency: Adaptive (P95 2530ms, ổn định nhất)
- Nếu ưu tiên cân bằng: Peak-EWMA (hiệu suất trung bình trên cả hai chỉ báo)
- Nếu cần đơn giản hoá: Round Robin (cải thiện đáng kể so với heterogeneous)

5.5 Khuyến nghị cho ba kịch bản ứng dụng điển hình

Kịch bản A – Tối ưu Chi phí Phần cứng (hệ thống kết hợp server cũ và mới): Sử dụng Weighted Response Time để khai thác tối đa năng lực của server mạnh, giảm thiểu ảnh hưởng của server yếu. Lợi ích: cải thiện hiệu suất tổng thể và tiết kiệm chi phí mở rộng. Nhược điểm: tăng tải lên server mạnh, yêu cầu giám sát chặt chẽ.

Kịch bản B – Ổn định Cao trong Tải Phức tạp (backend đồng nhất chịu heavy tail, flash sale, hoặc tác vụ tính toán nặng): Sử dụng Adaptive hoặc Least Connection tùy theo ưu tiên: Adaptive nếu cần bảo vệ Tail Latency, Least Connection nếu cần Average Latency tốt. Lợi ích: duy trì ổn định cao, hạn chế jitter. Nhược điểm: Adaptive có chi phí triển khai cao hơn (yêu cầu agent trên backend).

Kịch bản C – Tải Nhẹ, Burst Ngắn Hạn (ứng dụng có lưu lượng biến động nhưng không có tác vụ nặng): Sử dụng Peak-EWMA để phản ứng nhanh với biến động ngắn hạn và tránh overhead không cần thiết. Lợi ích: chi phí triển khai thấp, hiệu suất đủ tốt. Nhược điểm: không tối ưu cho heavy tail.

5.6 Hạn chế nghiên cứu và hướng tương lai

Mặc dù đạt được các phát hiện quan trọng, nghiên cứu có một số hạn chế:

Thứ nhất, thử nghiệm được tiến hành trên môi trường mô phỏng với số lượng server hạn chế (3 loại). Các hệ thống thực tế có thể có hàng trăm hoặc hàng nghìn server, nơi độ phức tạp tính toán O(N) của Least Connection có thể trở thành bottleneck đáng kể.

Thứ hai, các workload được tạo theo các mô hình đơn giản hóa (constant, burst, heavy tail). Các hệ thống sản xuất thực tế có thể gặp các dạng tải phức tạp hơn, ví dụ như tải theo chu kỳ tuần, tải tương phụ thuộc giữa các service trong kiến trúc microservice, hoặc tác động của caching layer.

Thứ ba, nghiên cứu không đánh giá chi phí của việc triển khai các thuật toán nâng cao (ví dụ: setup agent cho Adaptive, overhead của việc đo latency cho Peak-EWMA).

Hướng tương lai: Nghiên cứu đề xuất các hướng mở rộng, bao gồm (1) đánh giá các thuật toán trên quy mô lớn với hàng nghìn server, (2) tích hợp machine learning để tự động chọn thuật toán phù hợp dựa trên đặc điểm hạ tầng và tải theo thời gian thực, (3) xây dựng các hybrid algorithm kết hợp ưu điểm của nhiều thuật toán, (4) đánh giá hiệu suất trong kiến trúc microservice phức tạp.

5.7 Kết luận chính

Đóng góp chính của nghiên cứu không nằm ở việc tìm ra một thuật toán tốt nhất, mà ở việc chứng minh rằng lựa chọn thuật toán cân bằng tải là một bài toán mang tính ngữ cảnh. Việc hiểu rõ đặc điểm hạ tầng, dạng tải, và mục tiêu vận hành sẽ cho phép các kỹ sư hệ thống đưa ra quyết định triển khai phù hợp, từ đó đạt được sự cân bằng tối ưu giữa hiệu suất, chi phí, ổn định và khả năng mở rộng dài hạn của hệ thống.

6.Tài liệu tham khảo

1. The Study On Load Balancing Strategies In Distributed Computing System, April 2012, *International Journal of Computer Science & Engineering Survey* Vol.3,(No.2):19-30. https://www.researchgate.net/publication/257465354_The_Study_On_Load_Balancing_Strategies_In_Distributed_Computing_System

2. https://viettelidc.com.vn/tin-tuc/kham-pha-cac-thuat-toan-can-bang-tai-pho-bien?fbclid=IwY2xjawOrtNRleHRuA2FlbQIxMABicmlkETE5QVlTTVUxamF3VjB3cU5Ec3J0YwZhcHBfaWQQMjIyMDM5MTc4ODIwMDg5MgABHvWaMuBAR81V5Lu-t622GZkem8-CAXWHfL_1pTJnBoADqlin4CTwhwLJVsC1_aem__Lq3ENRiSXiJK0tpR-KQGA

3. https://www.geeksforgeeks.org/system-design/what-is-load-balancer-system-design/?fbclid=IwY2xjawOrtjFleHRuA2FlbQIxMABicmlkETE5QVlTTVUxamF3VjB3cU5Ec3J0YwZhcHBfaWQQMjIyMDM5MTc4ODIwMDg5MgABHrGktwLLSS169mYCLDW-c90r93Ad7D81HBBbdrwhXi2JnjUPGKTC75hrtbSN_aem_QtWtd05zaHPcxrxOya_MTg

4. https://aws.amazon.com/vi/what-is/load-balancing/?fbclid=IwY2xjawOruMpleHRuA2FlbQIxMABicmlkETE5QVlTTVUxamF3VjB3cU5Ec3J0YwZhcHBfaWQQMjIyMDM5MTc4ODIwMDg5MgABHlhvS_CsnHQu4Gk6h5xWOFcsUCS1tXwg65DV9eVQJEghs9Nsr9suK2-Fq453_aem_z3IBpucWdj37SSdcxjPbew