CS 553 Design of Internet Services

Adaptive Feedback-Based Load Balancing with Predictive Analytics for Enhanced Web Service Performance

Team Members:

Harshish Bedi hsb53

Aaditya Chinchkhedkar ac2968

Varun Vekaria vv381

Abstract

Internet Services, including ecommerce, social media or web streaming, require a good and efficient load distribution to maintain optimal performance under varying workloads. Traditional or static load balancing strategies like round-robin are not as efficient in terms of adaptability to real-time conditions. This project introduces a Feedback based Dynamic Load Balancer, which dynamically routes HTTP requests based on real-time metrics such as CPU load, memory usage, and response latency. In this project we implement various static load balancing algorithms and dynamic load balancing algorithms and do a rigorous performance evaluation and further determine the improvements in throughput and reduction in tail latency, and balanced resource utilization. We aim to create our own dataset based on this performance and propose a predictive analytics layer to enhance performance by forecasting server conditions proactively. Using the new predictive layer we perform the benchmarking again to get an idea over the performance boost that we anticipate to achieve using the new method.

Introduction

The key points in web service architecture include minimum latency, high throughput and a well balanced resource allocation. Standard static load balancing methods that distribute traffic do not take into account the current server conditions which can lead to suboptimal performance during unpredictable or varied traffic patterns. In our project we plan to make a feedback-driven load balancing system capable of intelligently routing incoming requests depending on server health indicators acquired in real time. This system not only adapts swiftly to changing conditions but also anticipates future states, proposing predictive adjustments to optimize routing decisions even further. The main aim here is basically to develop and create a robust and adaptive load balancing system that outperforms previous static approaches.

Methodology

This is what we anticipate to do and a general idea of how the architecture would follow.

1. Load Balancer:

- The role of LB would be to accept and route client HTTP requests.
- **Implementation:** We plan to develop the load balancer with Python's FastAPI to ensure concurrency and also efficient request handling.

2. Backend Servers (Mock Services)

- The role of Backend Servers would be to simulate the realistic backend server behaviors, responding to requests and to report the performance metrics.

- **Implementation:** Each backend service in our case the fast-api built server, would be provided an endpoint that exposes parameters such as CPU load, request latency, and queue length. We plan to develop this using Python and Python's Fast API.

3. Feedback Collection Module

- **Role:** We intend to use this to periodically gather data of health and performance from all backend servers, process and analyze this data, and maintain a continuously updated score that reflects each backend's suitability to handle new incoming requests.
- **Implementation:** We are not fixated on the steps for this process however temporarily we plan to implement this using Python's Asyncio library along with aiohttp to handle the async operation and data processing. We also plan to handle any backend failures using this and by spotting unresponsive servers and not use them in routing till they become responsive.

Routing Algorithms:

Our load balancer evaluated various dynamic routing strategies, comparing them against static methods The static methods again are not fixated, for now we are considering the following (Round Robin and Least Connections):

- **Least Response Time:** Prioritizes servers with lowest latency.
- **Lowest CPU Load:** Routes to the least CPU-utilized server.
- **Exponential Weighted Moving Average (EWMA):** Provides stable routing decisions by smoothing short-term metric fluctuations.
- **Composite Scoring Algorithm:** As it integrates and aggregates multiple metrics (latency, CPU load) into a single performance score/metric.

Performance Evaluation:

Performance benchmarks would be carried out using workload simulation tools. Some of the tools we think would work out well include wrk, ApacheBench, and Locust measuring critical metrics. The following metrics would be taken into consideration for evaluating performance. We may add additional metrics if required.

- Average response time
- 99th percentile latency (tail latency)
- Failed requests percentage

- Server utilization distribution

Additional Features (If time permits):

We plan to build a **Real-time Dashboard using ideally Grafana** for visualizing load balancer decisions and server health.

References

Chen, J., Wang, Y., Huang, X., Xie, X., Zhang, H. and Lu, X., 2022. ALBLP: Adaptive Load-Balancing Architecture Based on Link-State Prediction in Software-Defined Networking. Wireless Communications and Mobile Computing, 2022(1), p.8354150.

Sharma, S., Singh, S. and Sharma, M., 2008. Performance analysis of load balancing algorithms. *World academy of science, engineering and technology*, *38*(3), pp.269-272.

Alhilali, A.H. and Montazerolghaem, A., 2023. Artificial intelligence based load balancing in SDN: A comprehensive survey. *Internet of Things*, *22*, p.100814.