Experimenting and Optimizing Servers with AWS RDS and ALB

Random Team: Yinuo Feng, Nuanxin Jin, Boyuan Sun, Zihe Chen, Chenyu Jiang

Table of contents

01 Introduction

02 The Problem

03 Proposed System Architecture

04 Experiments & Results

05 Analysis

06 Future Works

Tech Stack

Server: Go
Http Client: Java
Database: AWS RDS (MySQL)

Infrastructure: AWS EC2, Cloud 9, AWS ALB, Terraform, Packer, AWS AMI

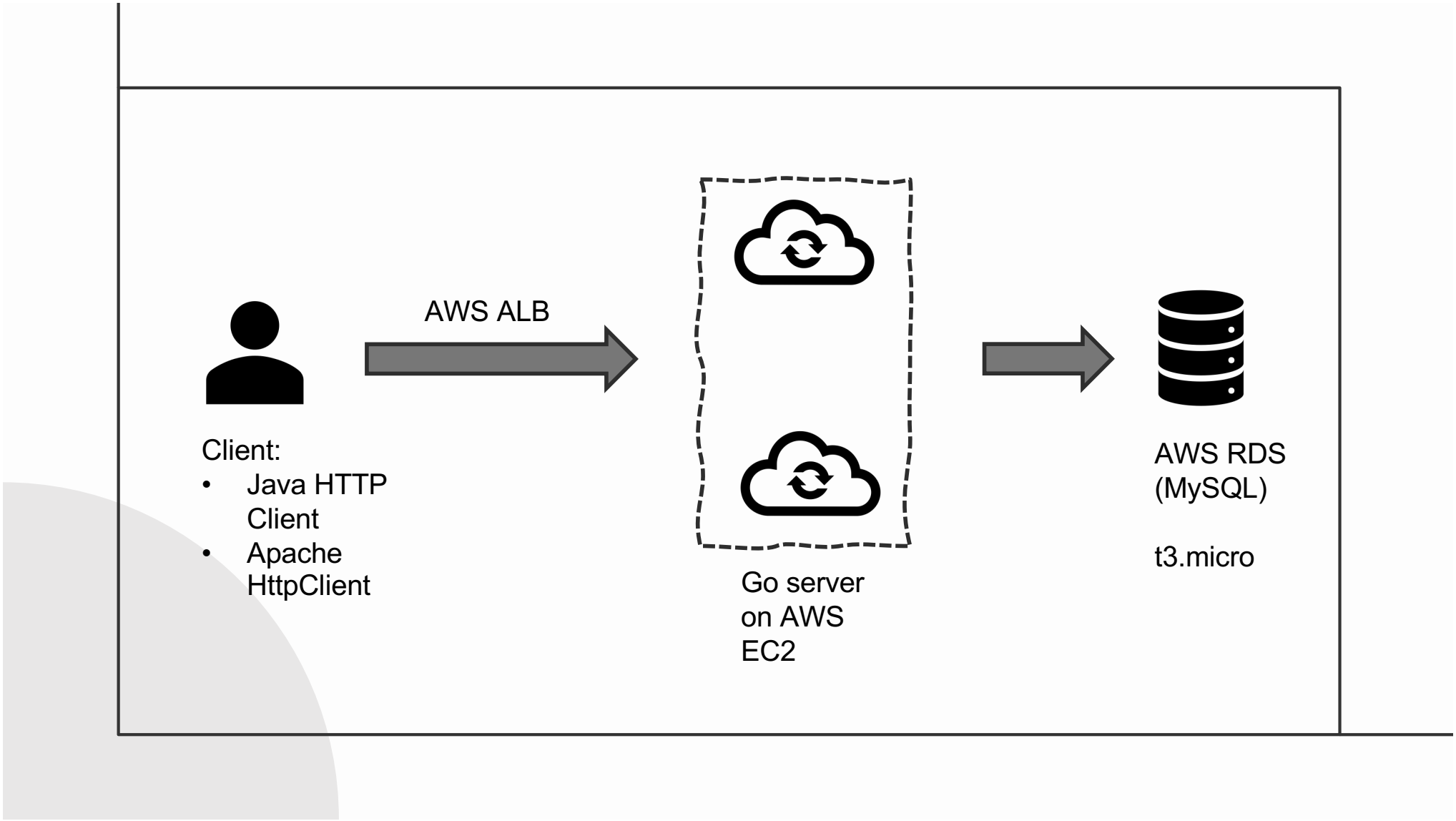
Testing & Metrics: Wall time, Throughput, Latency, p99 response time, Number of Successful GET/POST Requests

Data Structure

Column	Type	Description
id	INT AUTO_INCREMENT PRIMARY KEY	Unique album ID
artist	VARCHAR(255) NOT NULL	Artist's name
title	VARCHAR(255) NOT NULL	Album title
year	INT NOT NULL	Release year of the album
image	LONGBLOB NOT NULL	Album cover image in binary format
image_size	INT NOT NULL	Size of image file in bytes

How to maximize successful GET/POST requests?

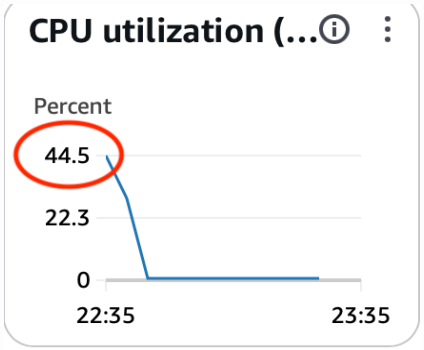
Proposed System Architecture

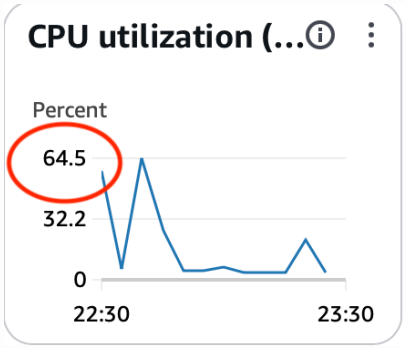

AWS ALB
Client:
• Java HTTP Client
• Apache HttpClient
Go server on AWS EC2
AWS RDS (MySQL)
t3.micro

Experiments & Results

Load Balancing Setup

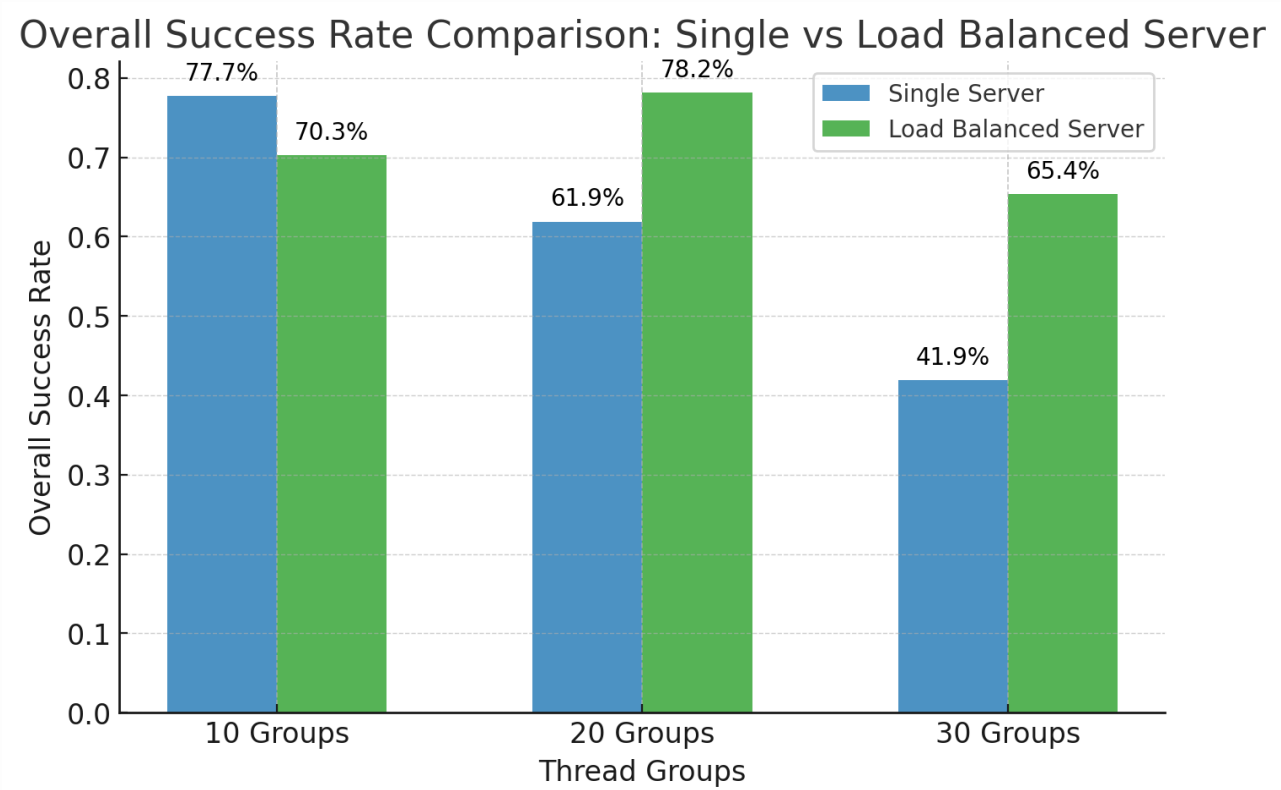
- What is Application Load Balancer?
- Round Robin: Not Truly Equal



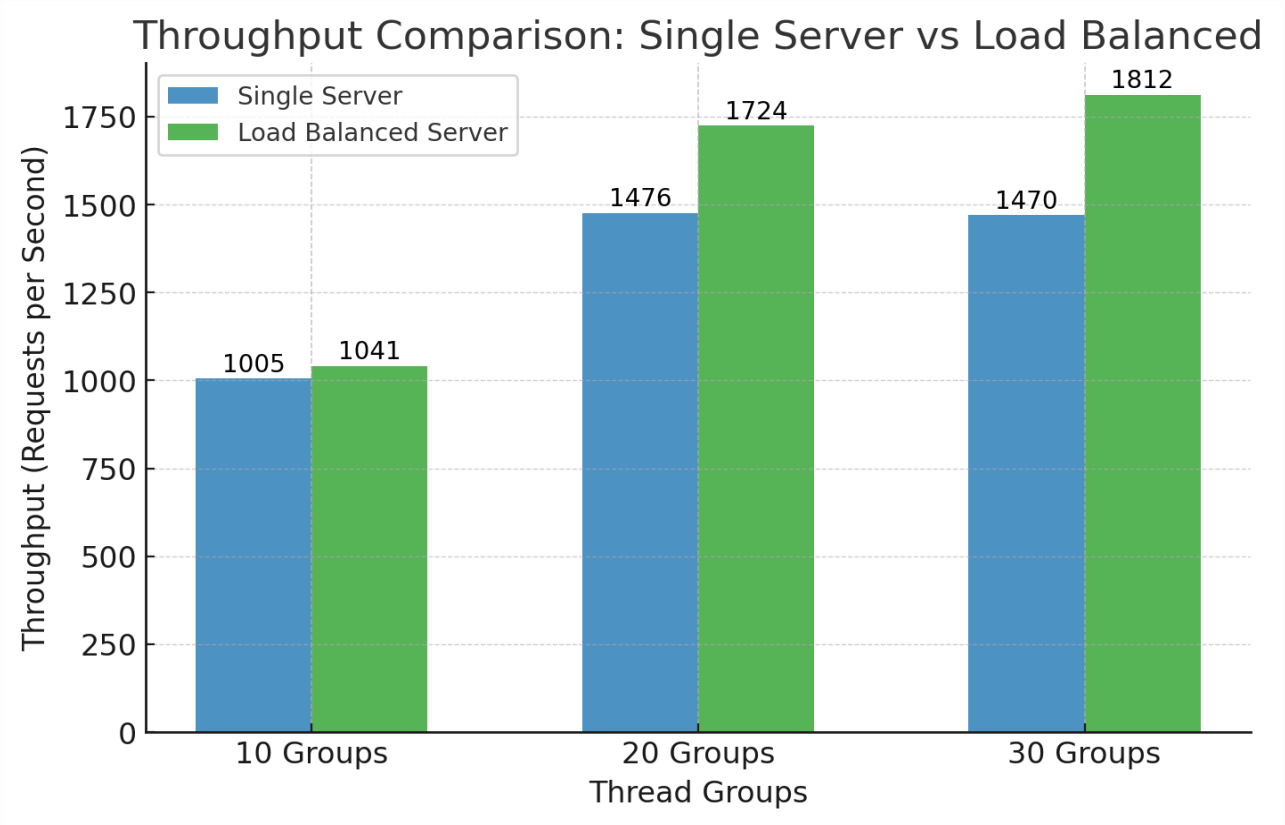


- Fixing the Imbalance: Least Outstanding Requests
- There is Always **COST**!

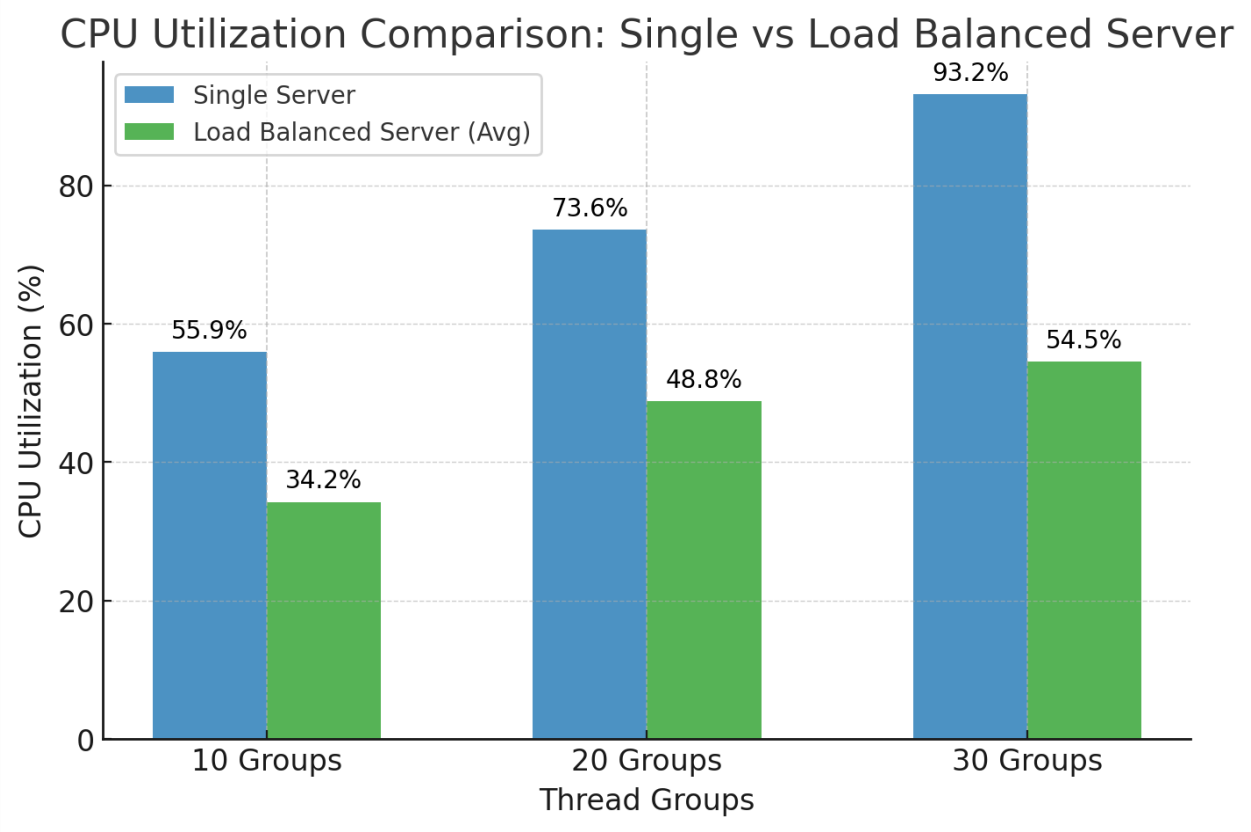
Load Balancing Has Higher Success Rate



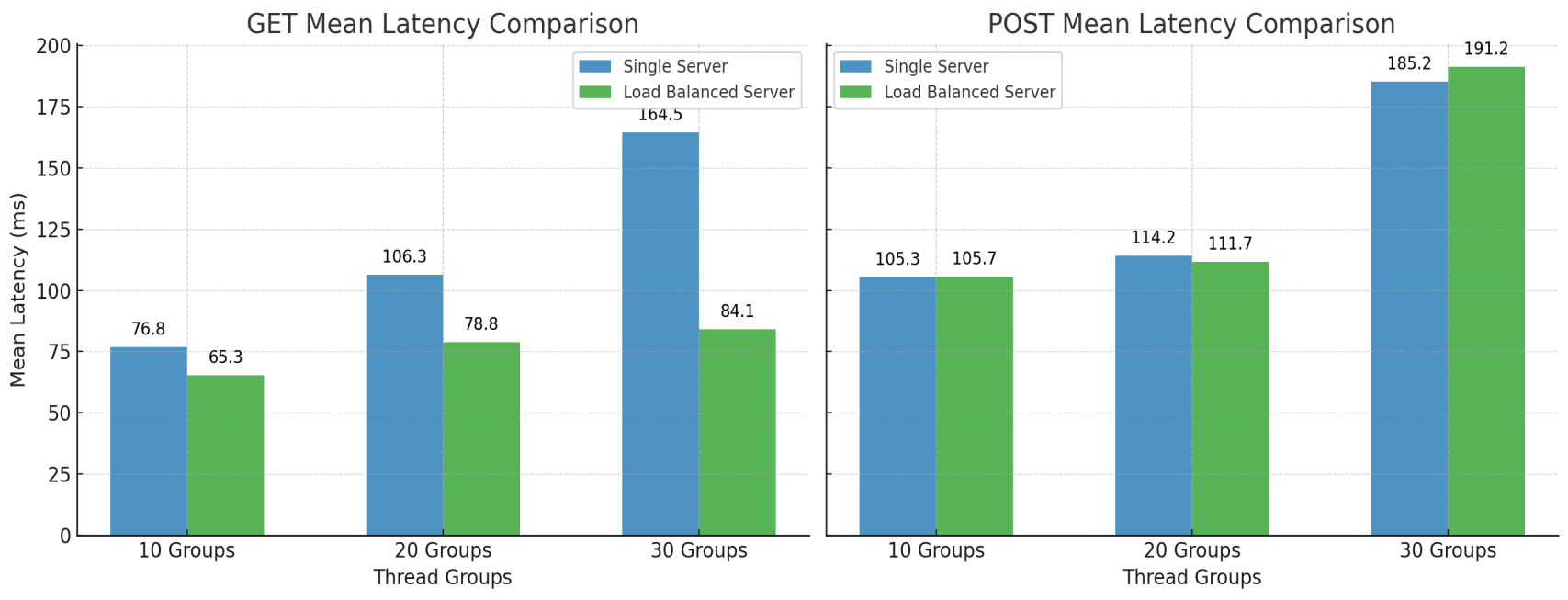
Load Balancing Has Higher Throughput



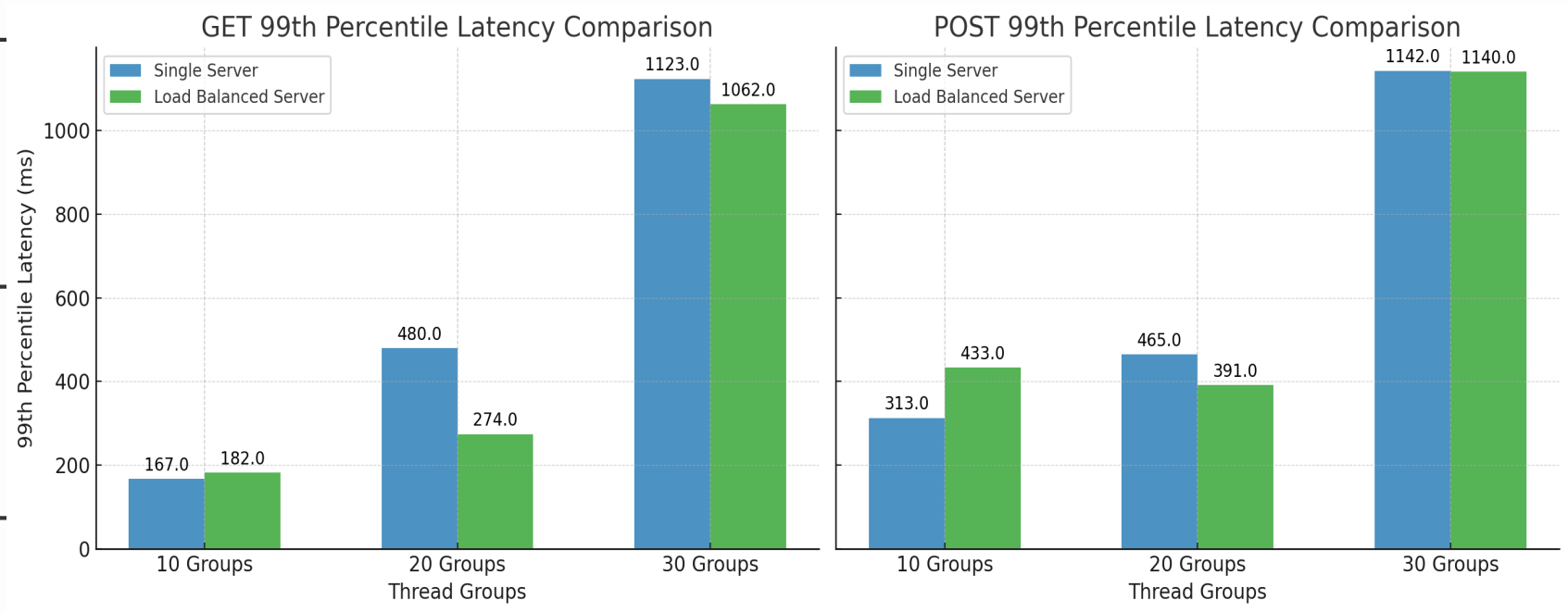
Load Balancing Optimizes CPU Utilization



Load Balancing Decreases Latency ... ?



Load Balancing Decreases Latency ...?



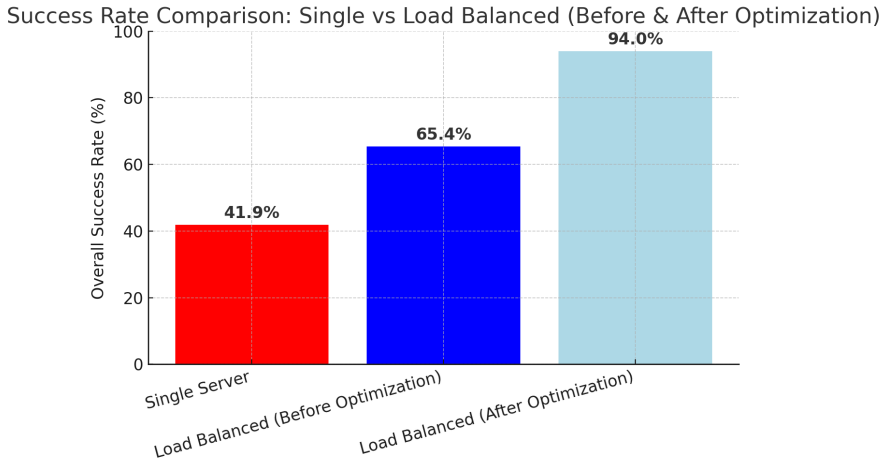
It is always about trade-offs.

From Unlimited to Smart Limits: Load Balancer Optimized

Single Server: Overloaded (from server)
Load Balancer (No Limits): Still overloaded (from DB)

Our bottleneck changes!

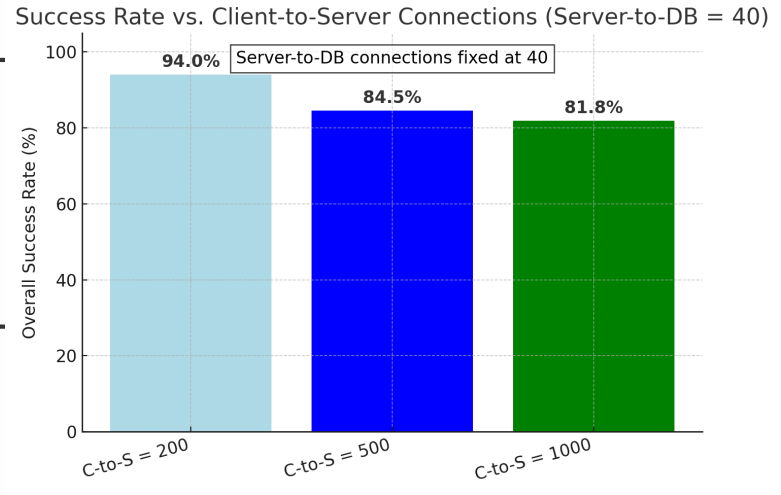
We limited server-to-database connections to 40, instead of unlimited.

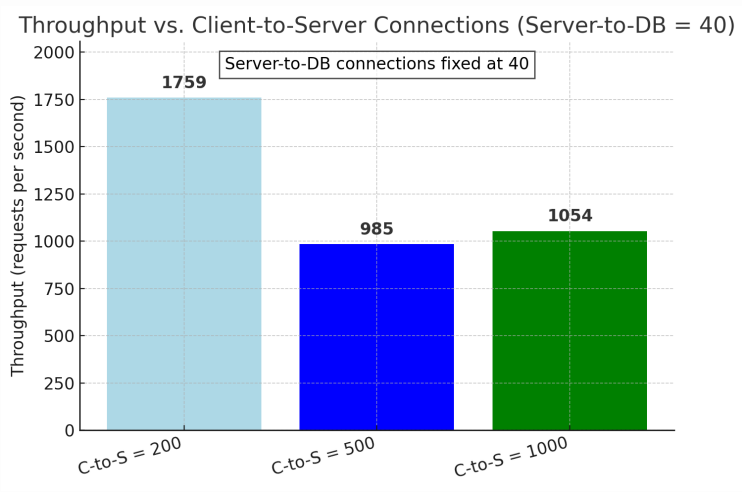


Success rate jumped to 94%!!

More connections ≠ Better Performance. Smart limits = Higher success

We Also Increased Client-to-Server Connections... But?





What we did:
Increased client-to-server connections
(from **200** → **500** → **1000)**

⇨

Expected: Higher throughput & success rate
Reality: Success dropped, throughput didn't improve

WHY?

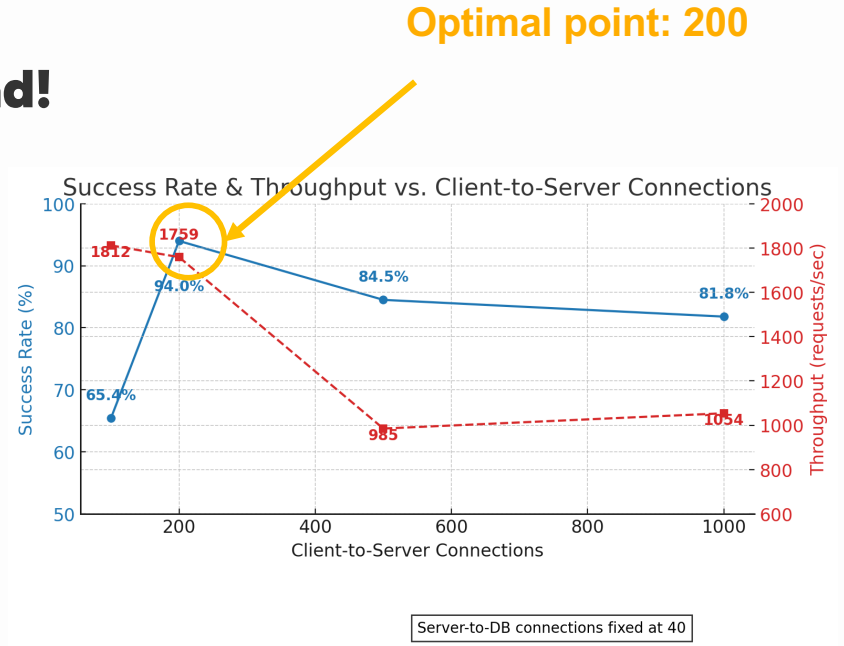
This is what we have learned

Too many connections = More overhead!

- New TCP connections increase **CPU & memory usage**
- **Possible contention issues** (e.g., DB locks)

200 might be the optimal value

- Increasing beyond 200 didn't bring benefits
- Extra overhead is expected

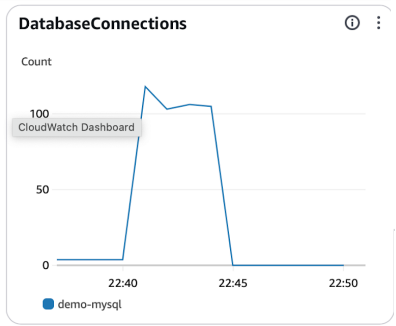


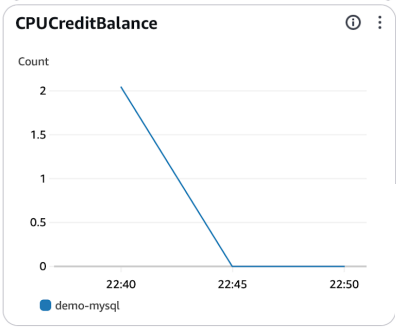
More is NOT always better. Scaling needs a strategy.

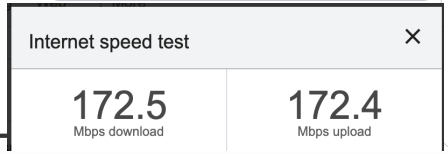
Maybe we need to ask ourselves: what kind of system are we building?

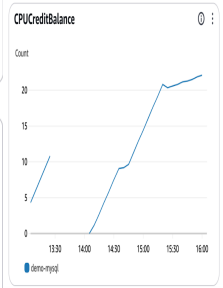
Future Works

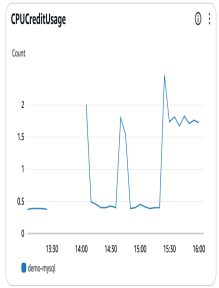
Db.t3.medium + Read Replica + Max Connection limits MongoDB

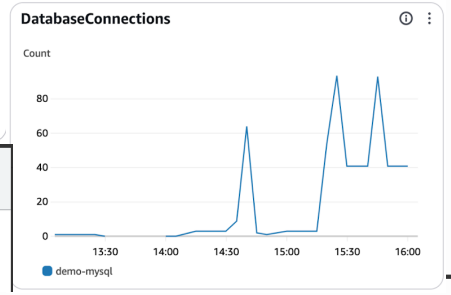


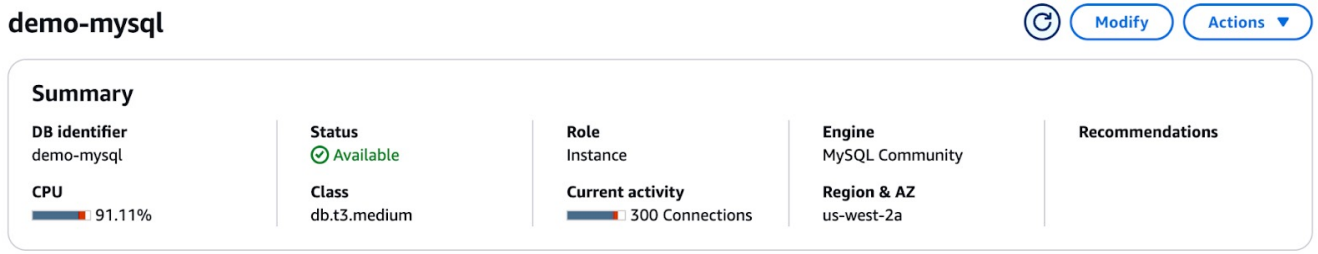


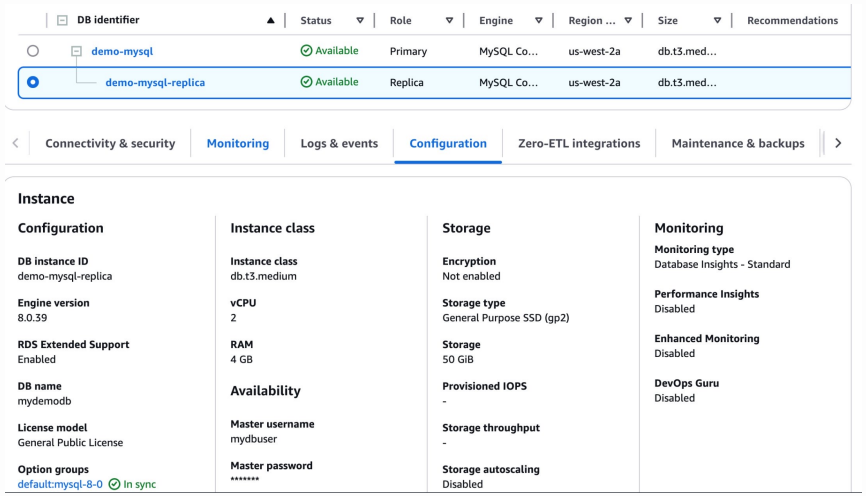












Thanks!