An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems

by Yu Gan et al.

Introduction

- Analysis of the implications of microservice architectures across the stack
  - Hardware architecture implications
  - OS & Networking implications
  - Cluster management implications
  - Application & programming framework implications
  - Tail at scale implications
- Developed DeathStarBench suite with different microservice applications to test

DeathStarBench suite

Social Network

Broadcast-style social network with follows

Media Service

Service for browsing movie information, renting and streaming movies

E-Commerce Service

E-Commerce site inspired by Sockshop

Banking system

Secure banking system for processing payments, requesting loans and credit cards

Swarm coordination

Routing of a swarm of programmable drones

Hardware architecture implications

- Lower amount of i-cache misses in microservices
- Due to simplicity of microservices
- Microservices more sensitive to poor single-core performance
- Likely caused by much stricter single-service time constraints

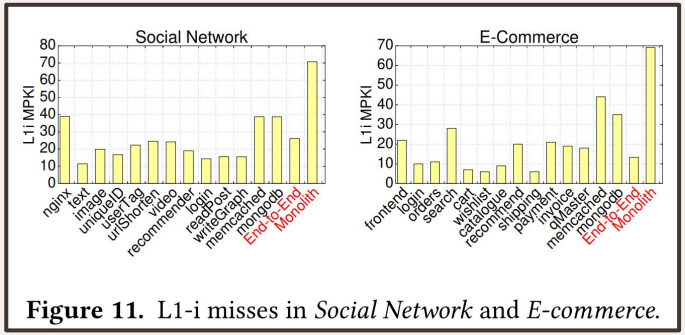

Figure 11. L1-i misses in *Social Network* and *E-commerce*.

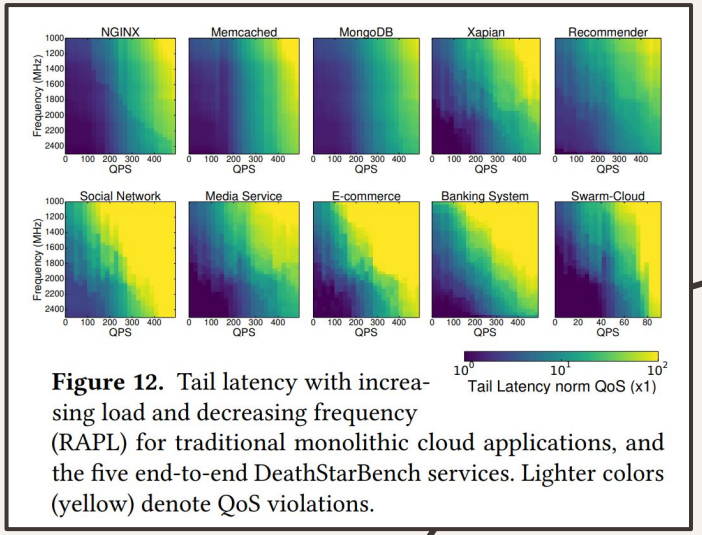

Figure 12. Tail latency with increasing load and decreasing frequency (RAPL) for traditional monolithic cloud applications, and the five end-to-end DeathStarBench services. Lighter colors (yellow) denote QoS violations.

OS & Networking implications

- Small percentage of time in user mode
- Extensive use of libraries because of high development speed
- MongoDB and memcached spend a lot of time in kernel processing TCP packets or handling interrupts
- Microservices spend more time network processing
- Relative amount increases with load

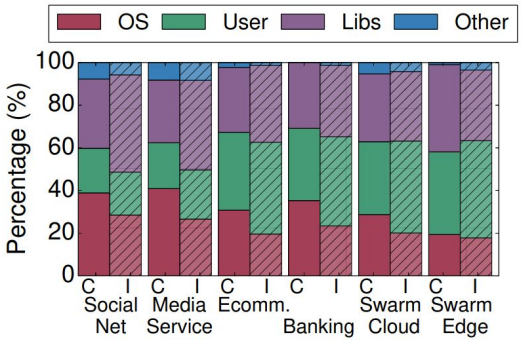

Figure 14. Time in kernel mode, user mode, and libraries for each service.

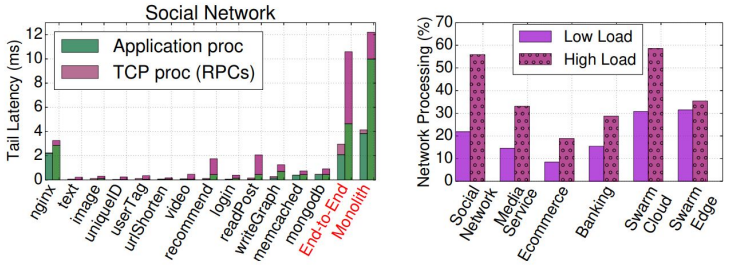

Figure 15. Time in application vs network processing for (a) microservices in *Social Network*, and (b) the other services.

Cluster management implications

- Microservice architecture causes hotspots increasing latency that are hard to identify
- Very hard to solve for autoscalers
- Recovery takes longer

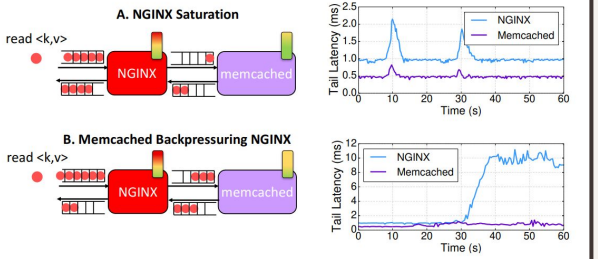

Figure 17. Example of backpressure between microservices in a simple, two-tier application. Case A shows a typical hotspot that autoscalers can easily address, while Case B shows that a seemingly negligible bottleneck in memcached can cause the front-end NGINX service to saturate.

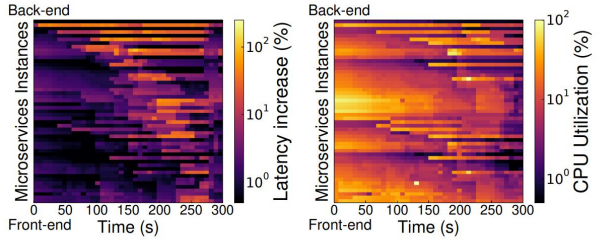

Figure 19. Cascading QoS violations in *Social Network* compared to per-microservice CPU utilization.

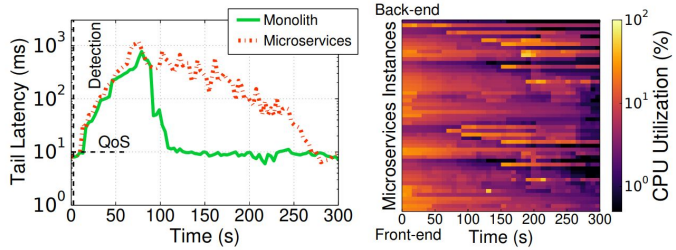

Figure 20. (a) Microservices taking longer than monoliths to recover from a QoS violation, even (b) in the presence of autoscaling mechanisms.

Application & programming framework implications

- Bottlenecks in microservices vary from service to service even if similar microservices are used in different applications
- Bottlenecks also change depending on load
- Serverless frameworks cheaper with similar average latency
- Latency variance is higher though
- Scales quicker than containerized version

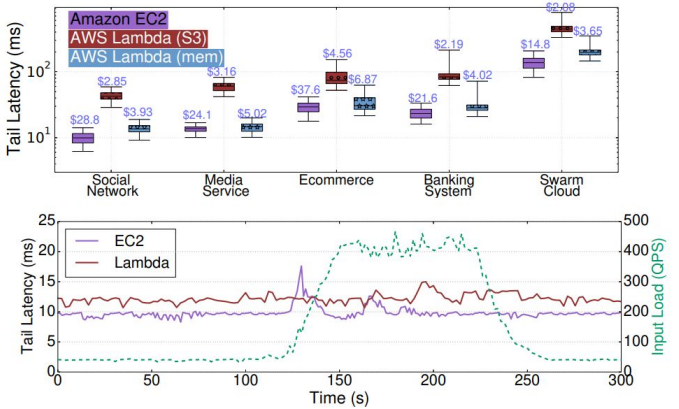

Figure 21. Performance and cost for the five services on Amazon EC2 and AWS Lambda (top). Tail latency for *Social Network* under a diurnal load pattern (bottom).

Tail at scale implications

- Hotspots still a really big problem
- Can be solved by rate limiting
- Uneven user-request distribution can disrupt service quality severely
- Single slow servers in big clusters can reduce goodput to almost zero

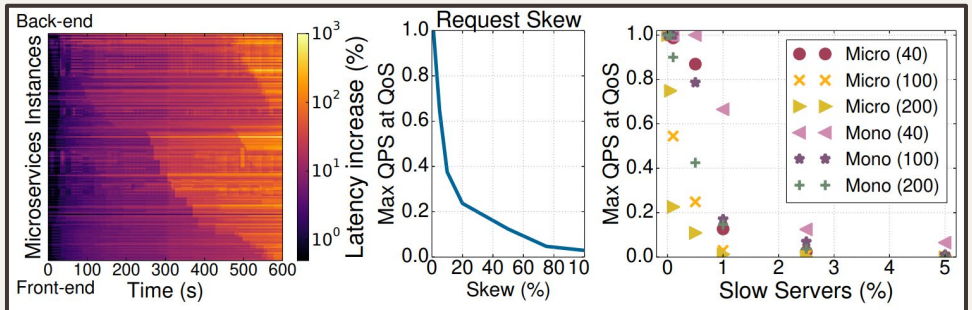

Figure 22. (a) Cascading hotspots in the large-scale *Social Network* deployment, and tail at scale effects from (b) request skew, and (c) slow servers.