AN OPEN-SOURCE BENCHMARK SUITE FOR MICROSERVICES AND THEIR HARDWARE-SOFTWARE IMPLICATIONS FOR CLOUD AND EDGE SYSTEMS

Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayantara Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla and Christina Delimitrou

Cornell University

ASPLOS 2019
Session Cloud I
EXECUTIVE SUMMARY

- **Cloud applications migrating from monoliths to microservices**
  - Monoliths: all functionality in a single service
  - Microservices: many single-concerned, loosely-coupled services
  - Modularity, specialization, faster development
  - Datacenters designed for monoliths → microservices have different requirements

- **An end-to-end benchmark suite for large-scale microservices**

- **Architectural and system implications**
  - Hardware design
  - OS/networking overheads
  - Cluster management
  - Application & programming frameworks
  - Tail at scale
**From Monoliths to Microservices**

- **Monolithic applications**
  - Single binary with entire business logic

- **Limitations**
  - Too complex for continuous development
  - Obstacle to adopting new frameworks
  - Poor scalability & elasticity

[Monolith Application diagram]

Monolith Application
FROM MONOLITHS TO MICROSERVICES

- **Microservices**
  - Fine-grained, loosely-coupled, and single-concerned
  - Communicate with RPCs or RESTful APIs

- **Pros**
  - Agile development
  - Better modularity & elasticity
  - Testing and debugging in isolation

- **Cons**
  - Different hardware & software constraints
  - Dependencies → complicate cluster management
FROM MONOLITHS TO MICROSERVICES
**Motivation**

- Explore implications of microservices across the system stack

  5. Tail at scale

  4. Application and frameworks

  3. Cluster management

  2. OS/Network overheads

  1. Hardware design
Motivation

- Explore implications of microservices across the system stack

Need representative, end-to-end applications built with microservices
Motivation

- Previous work in cloud benchmarking
  - CloudSuite [ASPLOS’12]
  - Sirius [ASPLOS’15]
  - TailBench [IISWC’17]
  - μSuite [IISWC’18]

Focus either on monolithic applications or applications with few tiers

- DeathStarBench suite
  - Focus on large-scale microservices that stress typical datacenter design
DEATHSTARBENCH SUITE

- **Design principles**
  - Representativeness
    - Use of popular open-source applications and frameworks
    - Service architecture following public documentation of real systems using microservices
**Design principles**

- Representativeness
- End-to-end operation
  - Full functionality using microservices
**DeathStarBench Suite**

- **Design principles**
  - Representativeness
  - End-to-end operation
  - Heterogeneity
    - Wide range of programming languages and microservices frameworks
- **Design principles**
  - Representativeness
  - End-to-end operation
  - Heterogeneity
  - Modularity
    - Single-concerned and loosely-coupled services
**Design principles**

- Representativeness
- End-to-end operation
- Heterogeneity
- Modularity
- Reconfigurability
  » Easy to update or change components with minimal effort
5 end-to-end applications, tens of unique microservices each

- Social Network
- Media Service
- E-Commerce Service
- Banking System
- Drone Coordination System
### Social network

**Frontend**
- Image Store Frontend
- Video Store Frontend

**Logic**
- Read Post
- Read Home Timeline
- Compose Post
- User Timeline
- Social Graph
- Recommender
- Unique ID
- URL Shorten
- Image
- Text
- Video
- User Tag
- Social Graph

**Caching & Storage**
- Memcached
- MongoDB
- User storage
- Post storage
- Index
- Index
- Index
- User timeline storage
- Redis
- Social graph storage
- Image storage
- Video storage

---

*Death Star Bench Suite*
Media service
E-commerce service
**Banking system**
### Drone coordination system

- **Frontend**
  - Client
  - Load Balancer
  - NGINX

- **Cloud**
  - Controller
  - Construct Route
  - OrientationDB
  - LuminosityDB
  - SpeedDB
  - LocationDB
  - VideoDB
  - ImageDB
  - TargetDB

- **Edge**
  - Edge Router
  - Controller
  - Location
  - Speed
  - Image
  - Video
  - Luminosity
  - Orientation
  - Obstacle Avoidance
  - MotionCtrl
  - Image Recognition
  - Stocking ImageDB
  - Log(node.js)

- **Edge Swarm**

---

**DEATHSTARBENCH SUITE**

Cornell University
Computer Systems Laboratory
### Case Study: Social Network

#### User sign up/login

**Frontend**
- Client
- Load Balancer
- NGINX

**Logic**
- User
- Unique ID
- Read Post
- Favorite
- Search
- Post Storage
- User Timeline
- RabbitMQ
- Write Home Timeline
- Social Graph
- Compose Post
- Read Home Timeline
- URL Shorten
- Image
- Text

**Caching & Storage**
- Memcached
- MongoDB
- User storage
- Redis
- Ajax
- Memcached
- MongoDB
- User timeline storage
- Social graph storage
- Image storage
- Video storage

**Storage**
- Home timeline storage
- Post storage
- User storage

**Databases**
- MongoDB
- Memcached
- Redis
- RabbitMQ
Write posts
CASE STUDY: SOCIAL NETWORK

- **Read home timeline**

```
Client \rightarrow Load Balancer \rightarrow NGINX
\rightarrow Frontend

Load Balancer
\rightarrow Image Store Frontend
\rightarrow Video Store Frontend
\rightarrow Frontend Logic

Logic
\rightarrow Read Home Timeline
\rightarrow Post Storage
\rightarrow Caching & Storage

Image Store Frontend
\rightarrow Image
\rightarrow Video
\rightarrow Social Graph

Read Home Timeline
\rightarrow Read Post
\rightarrow Post Storage

Post Storage
\rightarrow User Timeline
\rightarrow User Tag
\rightarrow Recommender

Caching & Storage
\rightarrow Memcached
\rightarrow Redis
\rightarrow MongoDB

Memcached
\rightarrow Post storage
\rightarrow Image storage
\rightarrow Video storage

User storage
\rightarrow User timeline storage
\rightarrow Social graph storage

MongoDB
\rightarrow Read Home Timeline
\rightarrow Write Home Timeline

MongoDB
\rightarrow User storage
\rightarrow Post storage

Redis
\rightarrow Home timeline storage

Memcached
\rightarrow Index_0
\rightarrow Index_1
\rightarrow Index_n

Index
\rightarrow User storage
\rightarrow Post storage
```

Cornell University
Computer Systems Laboratory
CASE STUDY: SOCIAL NETWORK

- **Search**

  Frontend:
  - Client
  - Load Balancer
  - NGINX

  Logic:
  - Unique ID
  - Read Post
  - Read Home Timeline
  - Favorite
  - Post Storage
  - User Timeline
  - User
  - Compose Post
  - Video
  - Social Graph
  - Write Home Timeline
  - RabbitMQ

  Caching & Storage:
  - Memcached
  - MongoDB
  - Index_0
  - Index_1
  - Index_n
  - User storage
  - Post storage
  - Home timeline storage
  - Social graph storage
  - Image storage
  - Video storage

  Components:
  - Video Store Frontend
  - Image Store Frontend
  - Text
  - URL Shorten
  - Image
  - Video
  - User Tag
  - Recommender
  - Video Storage
  - Image Storage
  - User Storage
  - Social Graph Storage
  - Home Timeline Storage
  - Post Storage
  - User Timeline Storage
  - RabbitMQ Storage
CASE STUDY: SOCIAL NETWORK

- **Recommendation**

  **Frontend**
  - Client
  - Load Balancer
  - NGINX
  - Image Store Frontend
  - Video Store Frontend
  - User Tag

  **Logic**
  - User
  - Recommender
  - Social Graph
  - Text
  - Read Home Timeline
  - Compose Post
  - Read Post
  - Favorite Search
  - Post Storage
  - User Timeline
  - Write Home Timeline
  - RabbitMQ

  **Caching & Storage**
  - Memcached
  - MongoDB
  - User storage
  - Post storage
  - User timeline storage
  - Home timeline storage
  - Social graph storage
  - Image storage
  - Video storage

---

Cornell University
Computer Systems Laboratory
ARCHITECTURAL AND SYSTEM IMPLICATIONS

- Explore implications of microservices across the system stack

5. Tail at scale

4. Application and frameworks

3. Cluster management

2. OS/Network overheads

1. Hardware design
**Hardware Design**

- **Brawny vs. wimpy cores**
  - Microservices are more sensitive to performance unpredictability than monoliths

---

Microservices

Monoliths

---

1. Hardware design
2. OS/Network overheads
3. Cluster management
4. Application and frameworks
5. Tail at scale
HARDWARE DESIGN

- **Brawny vs. wimpy cores**
  - Microservices are more sensitive to performance unpredictability than monoliths
Hardware Design

- **Brawny vs. wimpy cores**
  - Microservices are more sensitive to performance unpredictability than monolithic apps
  - Xeon vs Cavium servers

- **Cycle breakdown of each microservice**
  - Smaller fraction of frontend stalls than monoliths

- **I-cache pressure**
  - Lower I-cache pressure than monoliths
OS/Network Overheads

- **RPC overheads**
  - A large fraction of time spent in network stack

- **FPGA network acceleration**
  - Offload TCP stack on FPGA
  - 10 - 68x improvement on network processing latency
  - 43% - 2.2x improvement on end-to-end latency
**Latency back-pressure**

- Bottleneck services pressure upstreaming services
- Cause: Imperfect pipelining
  - HTTP/TCP HoL blocking
  - Limited number of worker threads/connections

Example: HTTP 1.1 HoL blocking
Cascading QoS violations

- Hotspots propagating along the dependency graph
- No obvious correlation to CPU utilization
- Difficulty in discovering the bottleneck and long time to recover from QoS violations
**Serverless frameworks**

- Compared long-running microservices on EC2 with short-running microservices on AWS Lambda
- Agile resource adjustments with diurnal load pattern
- Higher performance variability due to
  - No control of lambda placement
  - Communication through S3
  - Loading of dependencies
**Impact of slow servers**

- Larger cluster $\rightarrow$ larger impact of slow servers
- More severe tail latency increase compared to monoliths

![Graph showing impact of slow servers on Max OPS at QoS](image)
CONCLUSIONS

- Cloud applications from monoliths to microservices
- Study implications of microservices across the system stack
- Open-source benchmark suite for cloud and IoT microservices
- Explored the implications of microservices
  - More sensitive to performance unpredictability
  - Potential of hardware acceleration for networking
  - Need for cluster managers that account for dependencies
  - Tail at scale effects more prominent in microservices
Questions?

• Cloud applications from monoliths to microservices
• Study implications of microservices across the system stack
• Open-source benchmark suite for cloud and IoT microservices
• Explored the implications of microservices
  • More sensitive to performance unpredictability
  • Potential of hardware acceleration for networking
  • Need for cluster managers that account for dependencies
  • Tail at scale effects more prominent in microservices