Exploring Efficient Microservice Level Parallelism

Xinkai Wang, Chao Li, Lu Zhang, Xiaofeng Hou, Quan Chen, Minyi Guo

Department of Computer Science and Engineering

May 2022

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Contents

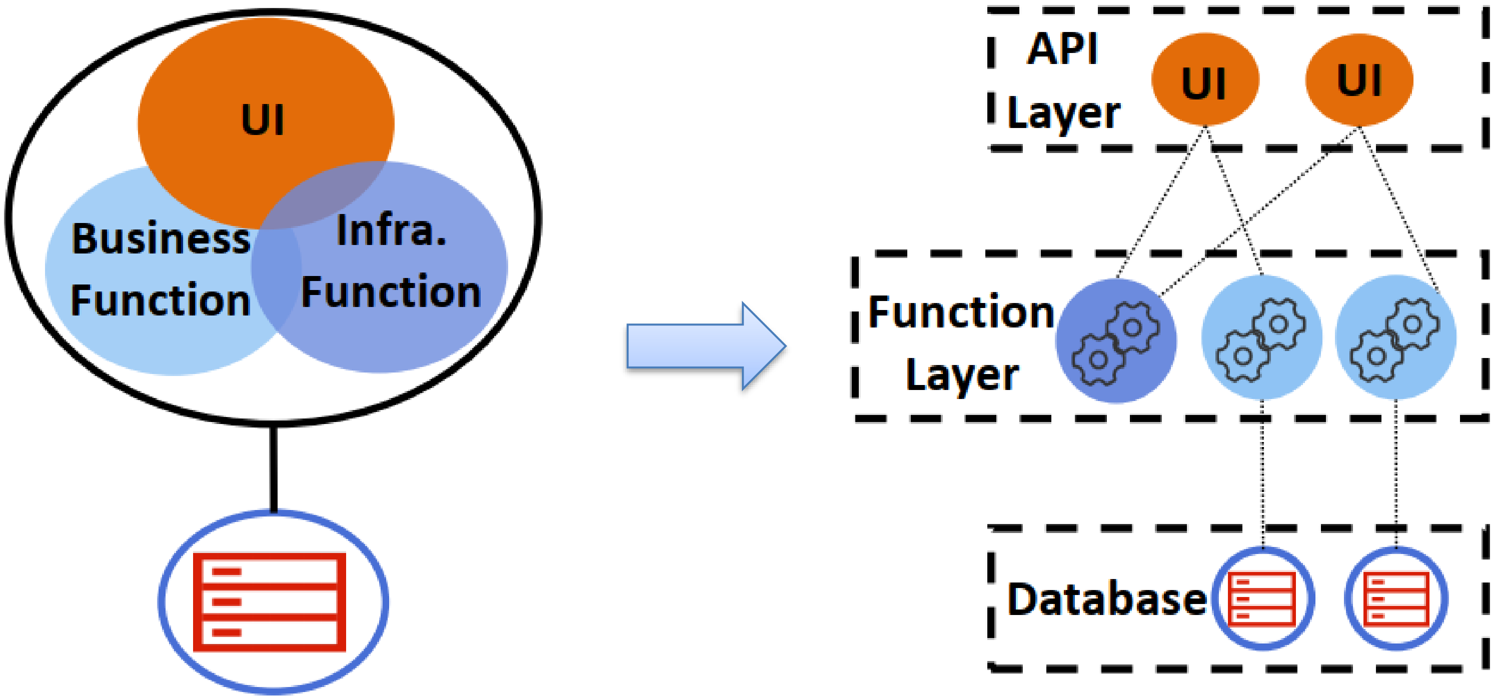
1. Introduction: Monolithic to Microservice
2. Motivation: Microservice Characterization
3. Microservice Level Parallelism
4. Evaluation: Effectiveness, Efficiency, Performance
5. Conclusions and Future Work

Contents

1. **Introduction: Monolithic to Microservice**
2. Motivation: Microservice Characterization
3. Microservice Level Parallelism
4. Evaluation: Effectiveness, Efficiency, Performance
5. Conclusions and Future Work

The Emerging Microservice Architecture

Microservice disaggregates a monolithic application into many tiny services, each of whom can be managed and tested independently.

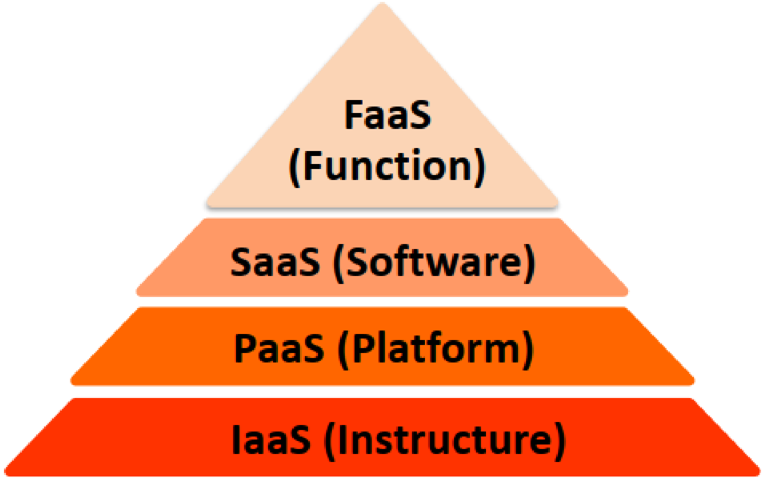


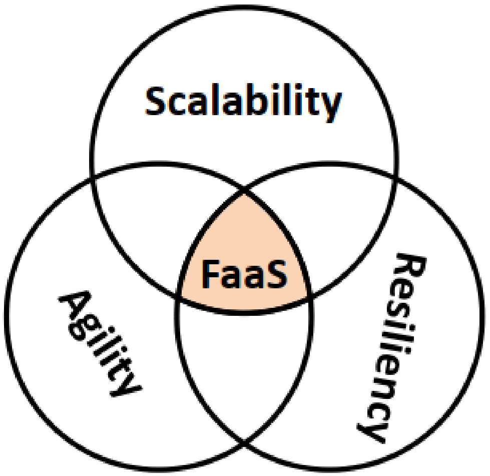
Monolithic Architecture

Microservice-based Architecture

The Revolution of Cloud Services

Microservices empower organizations to build and run scalable, agile, and resilient applications in dynamic environments.

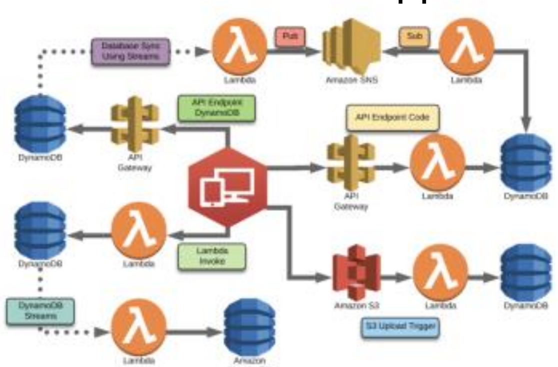




Industrial Applications of Microservices

Many IT companies such as Alibaba and Amazon are actively embracing this new software development paradigm.

Alibaba Commercial Applications



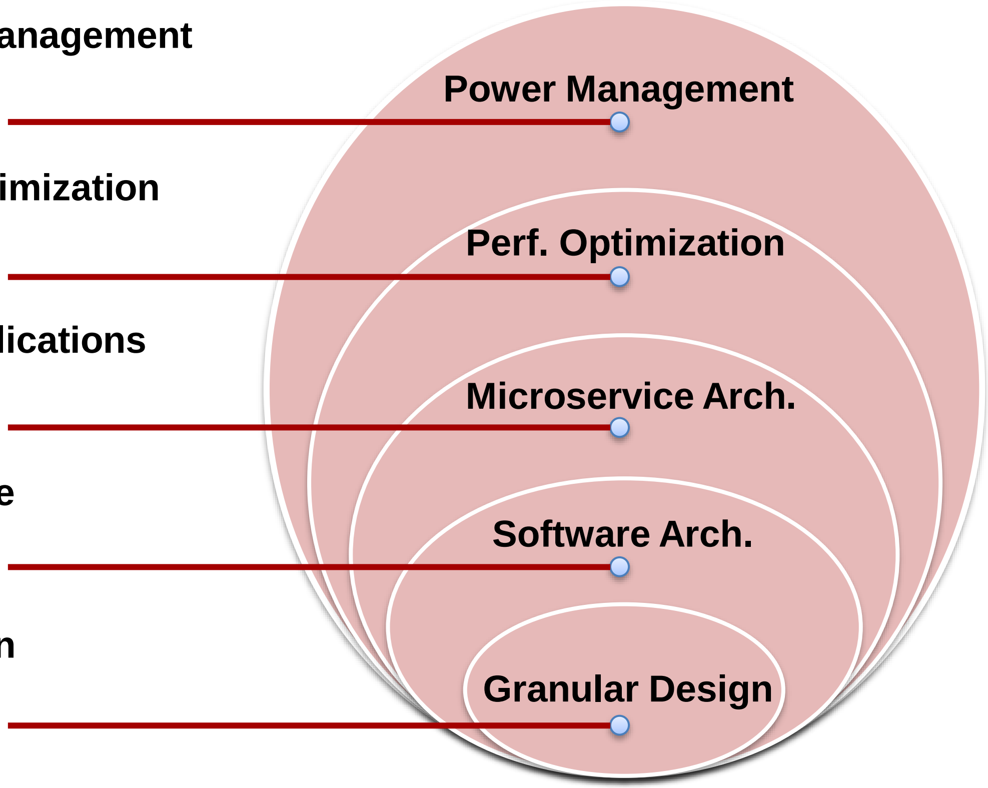
Amazon DynamoDB Built with Microservices

Partial Customer List

Prior Work on Microservices

Prior work: microservice design and optimization.

- **Microservice-scale power management**
  [SC’20, HPCA’19]
- **QoS-aware performance optimization**
  [IPDPS’21, SoCC’21]
- **Designing microservice applications**
  [ICSE’18, ASPLOS’19]
- **Service-oriented architecture**
  [AWS, Google, Microsoft, etc.]
- **Fine-grained software design**
  [Container, Lambda, etc.]



We enhance the QoS-aware performance optimization for microservices.

- **Fully exploiting the unique characteristics of microservices.**

Different Level of Parallelisms

Parallelism has been exploited at various levels of the system design for better performance and efficiency.

- **Instruction Level Parallelism (ILP)**

  Multiple instructions can be executed concurrently.

- **Thread Level Parallelism (TLP)**

  Multiple threads can be executed concurrently.

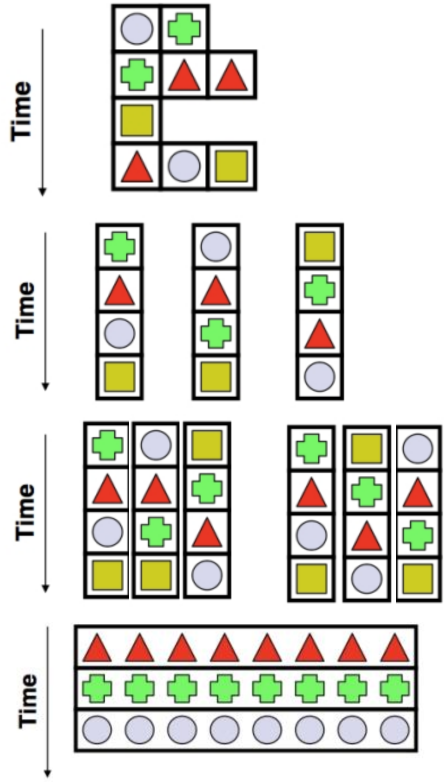
- **Request Level Parallelism (RLP)**

  Multiple requests can be executed concurrently.

- **Data Level Parallelism (DLP)**

  Instructions operate concurrently on several data.

- **Other emerging parallelisms…**

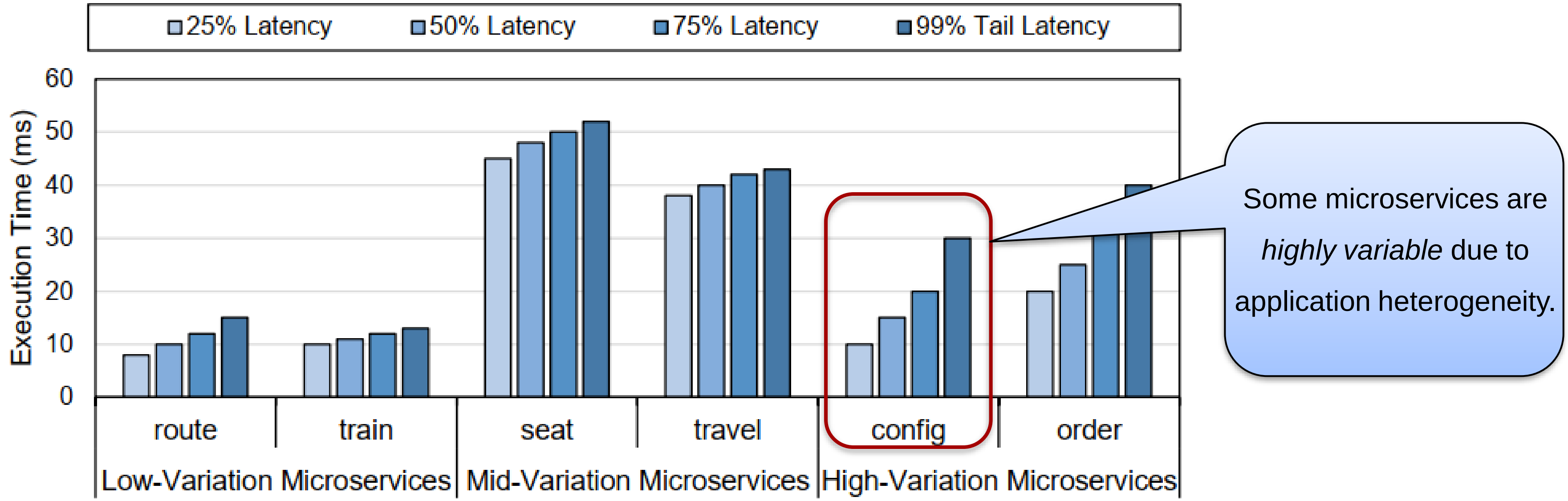


There are new types of parallelisms with new architectures and applications.

Contents

1. Introduction: Monolithic to Microservice
2. **Motivation: Microservice Characterization**
3. Microservice Level Parallelism
4. Evaluation: Effectiveness, Efficiency, Performance
5. Conclusions and Future Work

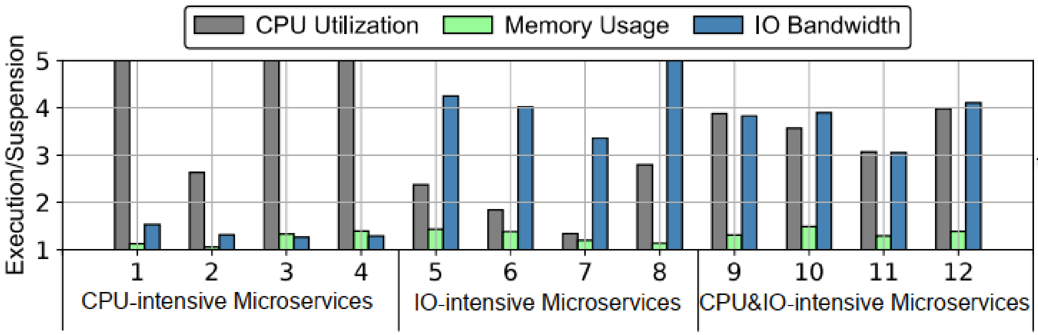
Impact of Application Heterogeneity



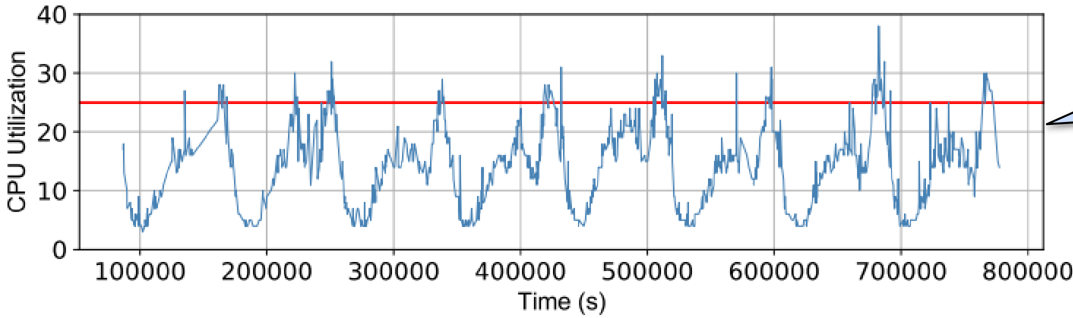
The application heterogeneity will cause the variation of microservice execution time under various user invocations.

- It is often caused by the actual execution logic of the microservice program.
- Different microservice exhibits various degree of application heterogeneity.

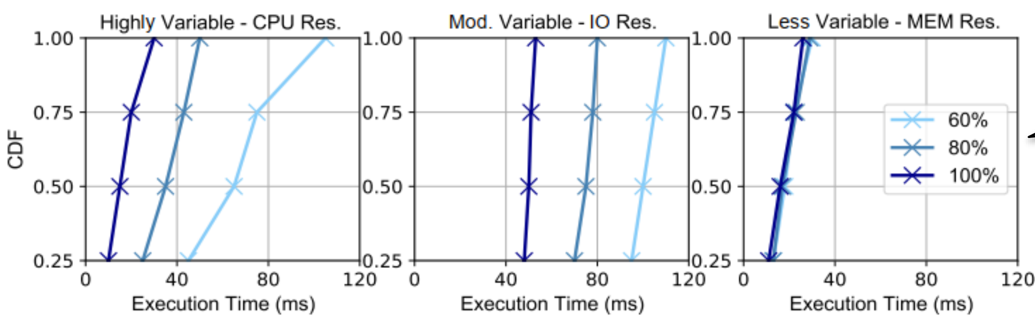
Impact of Resource Provisioning



Obs.1: Microservices have ***unique resource requirements*** and can be collocated.



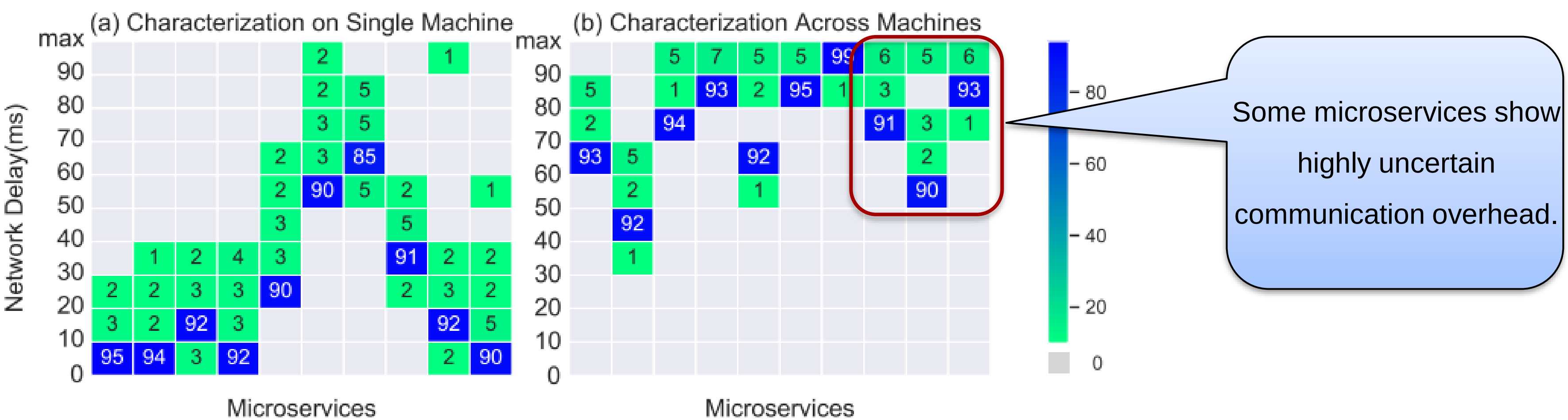
Obs.2: The resource demand may ***not be always met*** under highly dynamic outer traffic.



Obs.3:Microservices are ***differently sensitive*** to resource shortage from the perspective of performance.

The performance implication of microservice resource relationship is complex.

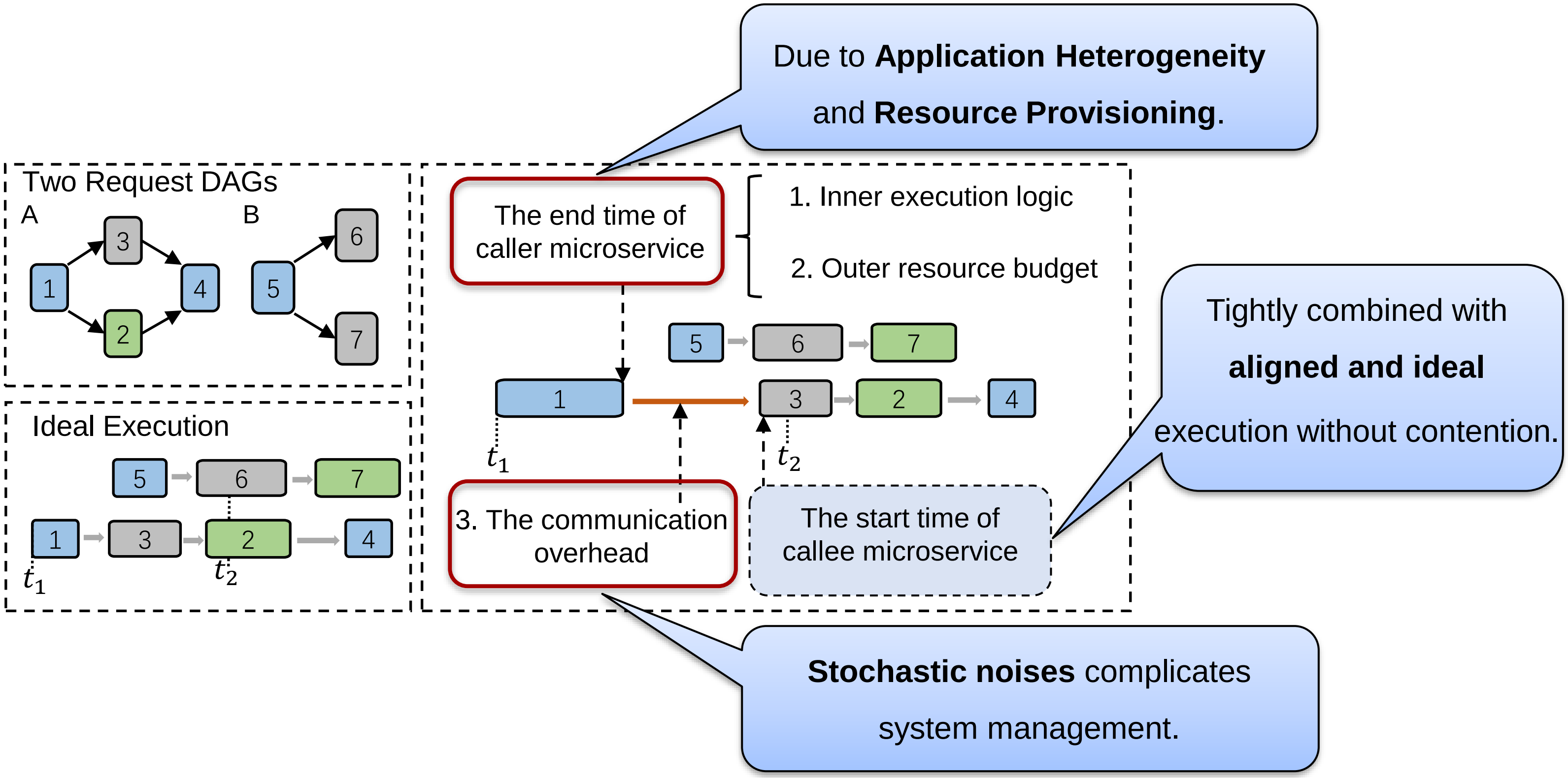
Impact of Communication Overhead



Stochastic noises in the microservice environment add up the complexity of scheduling of normal system operations.

- Here we focus on the variation of **communication overhead** between microservices.
- Communication time variations on single machine are more stable than across machines.

Summary of Design Challenges



The crux of the problem is two-fold:

- We need to know how different microservices should be coalesced.
- We must ensure fairly accurate alignment throughout the process.

Contents

Potential of Microservice Level Parallelism

Parallelism	ILP	TLP	MLP	RLP
Scheduling Level	Chip Level		System Level	
Granularity	Instruction	Instruction Stream	Microservice	Monolithic Application
Key Opti. Approach	Temporal	Spatial	Temporal	Spatial

Microscopic instruction scheduling (ILP, TLP)

Critical Void (MLP)

Macroscopic request scheduling (RLP)

MLP is critical and necessary:

- MLP is orthogonal to existing parallelism model.
- MLP focuses on microservice chain scheduling.
- MLP considers interrelationship and uncertainty of microservices.

Core Metric: Volatility of Requests

Based on characterization, microservices exhibit volatile behaviors.

- **We define volatility of request (V_r), indicating the likelihood of the request to deviate from its ideal execution conditions.**

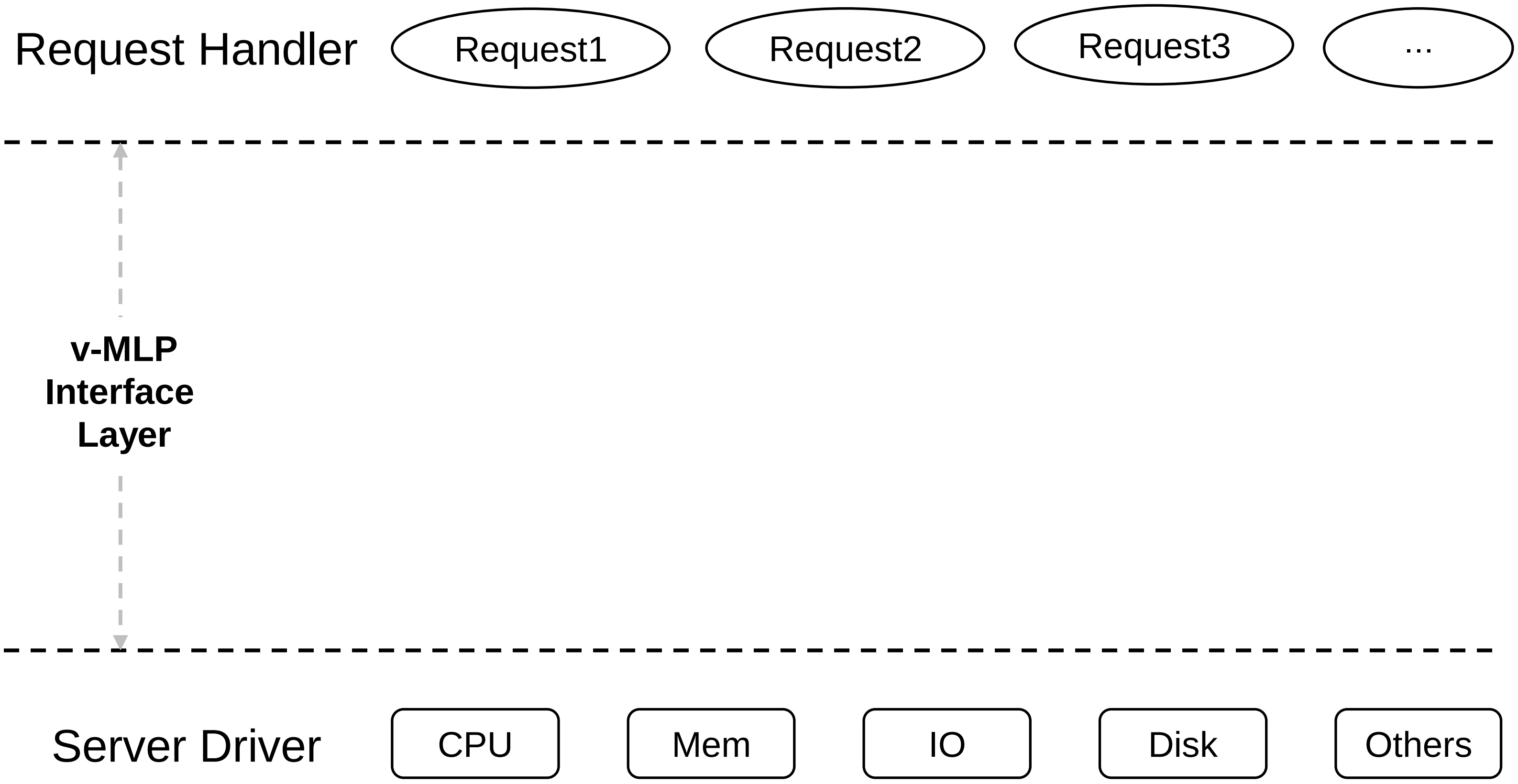
$$V_r = \alpha \times \sum_{i=1}^{n} I_i \times S_i \times C_i / n$$

Abbr.	Value Range	Descriptions
I	1(low) – 3(high)	Inner logic variability
S	1(low) – 3(high)	Sensitivity to resource
C	1-3 with Var(RTT)	Communication overhead

Understanding volatility helps make prudent decisions.

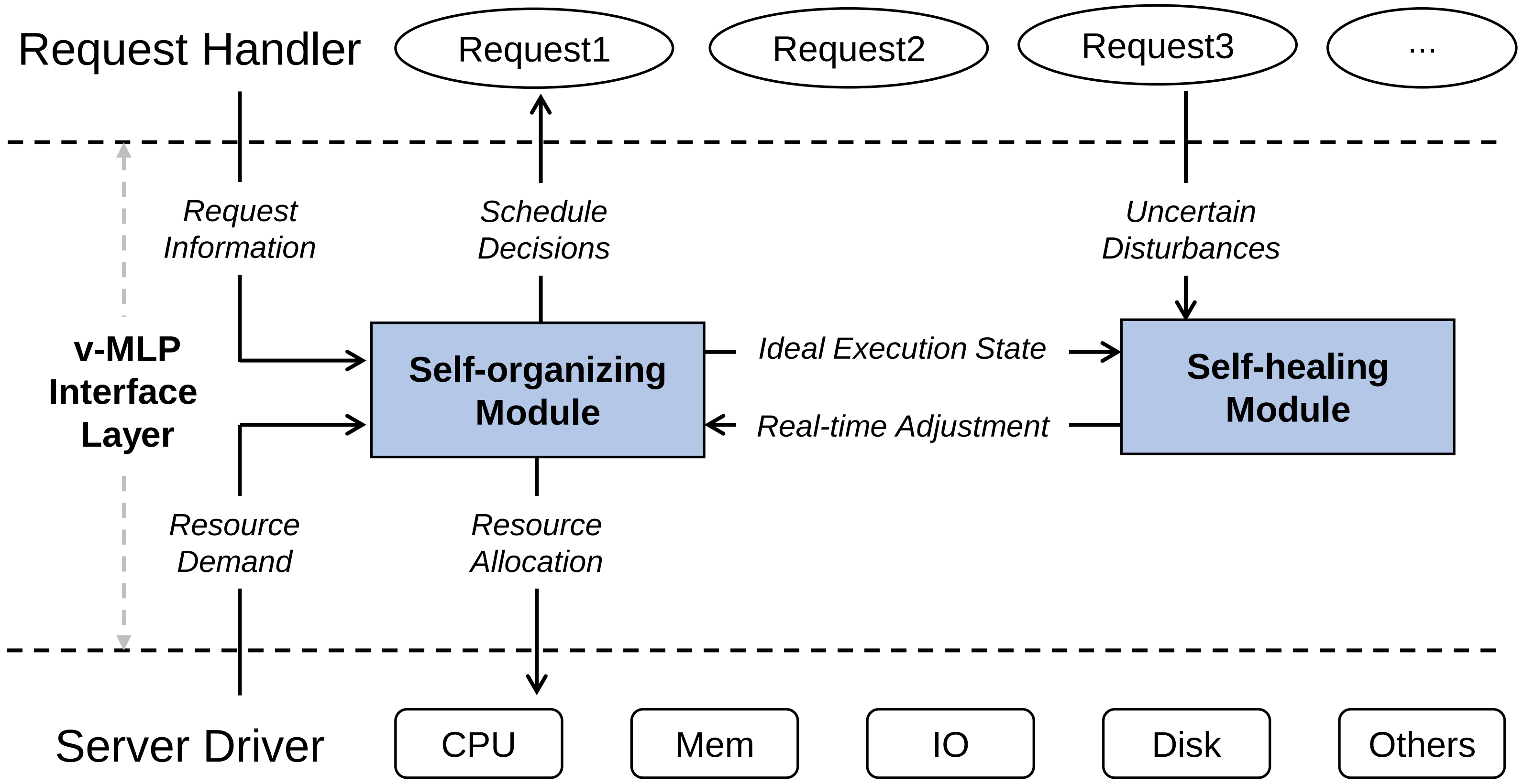
- Low volatility implies the start time of microservices is more predictable and less volatile.
- High volatility implies the start time of microservices is less predictable and more volatile.

Volatility-aware Microservice Level Parallelism



- v-MLP acts as the interface layer between the request handler and the server driver.
- v-MLP aims at the efficient resource management for microservices in datacenters.

Volatility-aware Microservice Level Parallelism



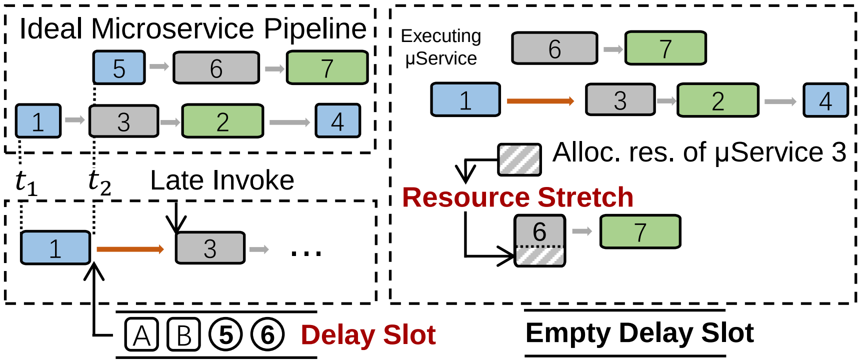
- Self-organizing module considers request information to coalesce microservices.
- Self-healing module handles uncertain disturbances during executions.

Design Principles of Self-organizing Module

- **Periodically refreshes the status of machines in the scheduling cluster by:**
  - Future remaining resource status
  - Future microservice execution status
- **Periodically reorder the request waiting queue by:**
  - Volatility of the request
  - Arrival time of the request
  - Shortest execution time of the microservices
  - SLA level of the request
- **Assign the microservices to machines with enough resource by volatility:**
  - Satisfy resource demand within 1st percentile of latency for low volatility.
  - Satisfy resource demand within 50th percentile of latency for mid volatility.
  - Satisfy resource demand within 99th percentile of latency for high volatility.

Design Principles of Self-healing Module

- Self-healing module handles **uncertain disturbances** in **real-time execution** based on the ideal microservice pipeline produced by self-organizing module.
- **Delay Slot Mechanism:**
  - Working with waiting independent microservices (nonempty delay slot).
  - Advance the execution of independent microservices to fill the resource vacancy.
- **Resource Stretch Mechanism:**
  - Working with no waiting independent microservice (empty delay slot).
  - Adjust the resource usage of executing microservices to fill the resource vacancy.



Contents

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Experiment Methodology

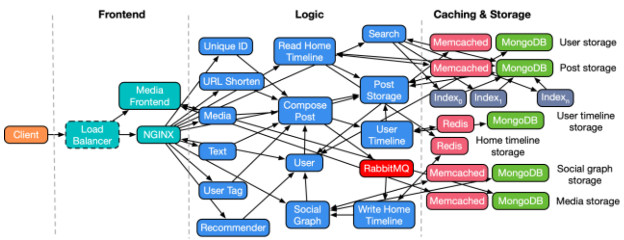
- **Experiment Platform**

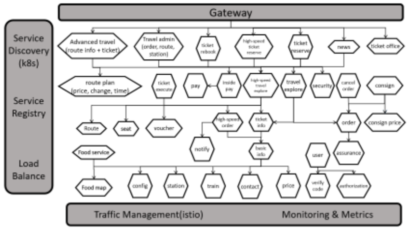
Characterization Platform Configurations	
Cluster	4 worker nodes (total 24 cores) + 1 manager node
Server	Dell R730, Intel® Xeon® E5-2620
Memory	32GB, DDR4 for each node
Host OS	Ubuntu 18.04.5 LTS, Docker 20.10.3
Simulation Platform Configurations	
Server	Dell R740 Intel® Xeon® Gold 5218
Host OS	Ubuntu 18.04.5 LTS, Docker 20.10.3

- **Experiment Benchmarks**

Microservice Apps

- DeathStarBench [Cornell]
- TrainTicket [Fudan]





Experiment Methodology

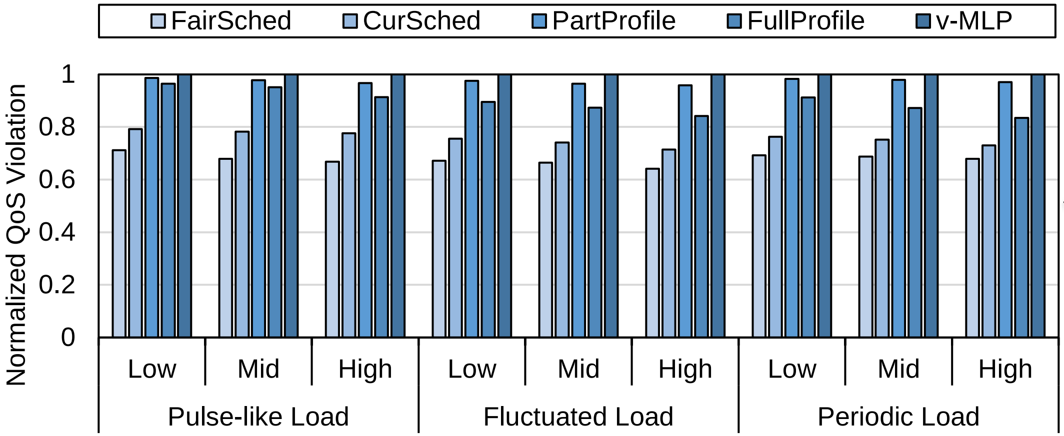
- **Evaluated workloads**

Category	Requests
High V_r	Compose-post in SN
	getCheapest in TT
Mid V_r	basicSearch in TT
Low V_r	Read-home-timeline in SN
	Read-user-timeline in SN

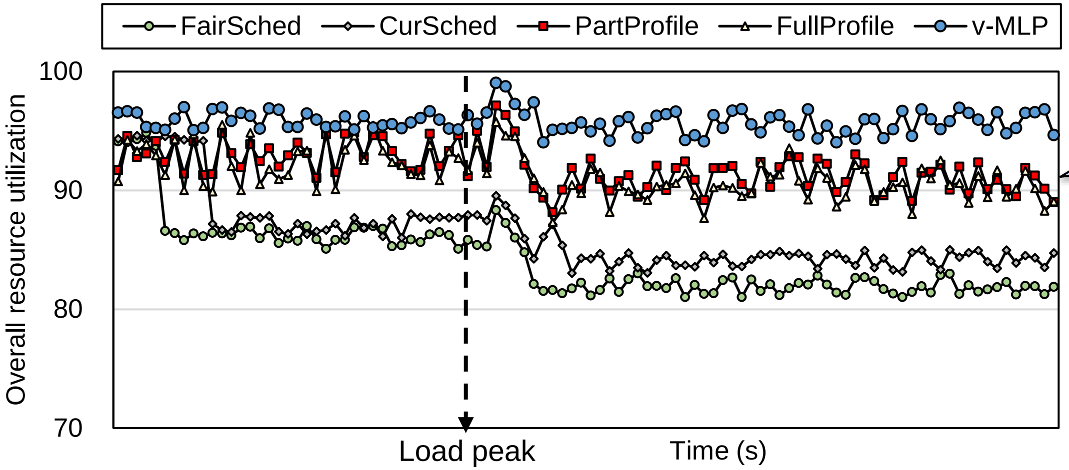
- **Existing scheduling schemes**

Category	Scheme	Descriptions
Simple Scheduler	FairSched	*FCFS, Allocate equal resource*
	CurSched	FCFS, Allocate by current load
Advanced Scheduler	PartProfile	Priority, Allocate by performance profile
	FullProfile	Priority, Allocate by overall profile
MLP scheme	v-MLP	Our Proposal

Evaluation Results: Effectiveness & Efficiency

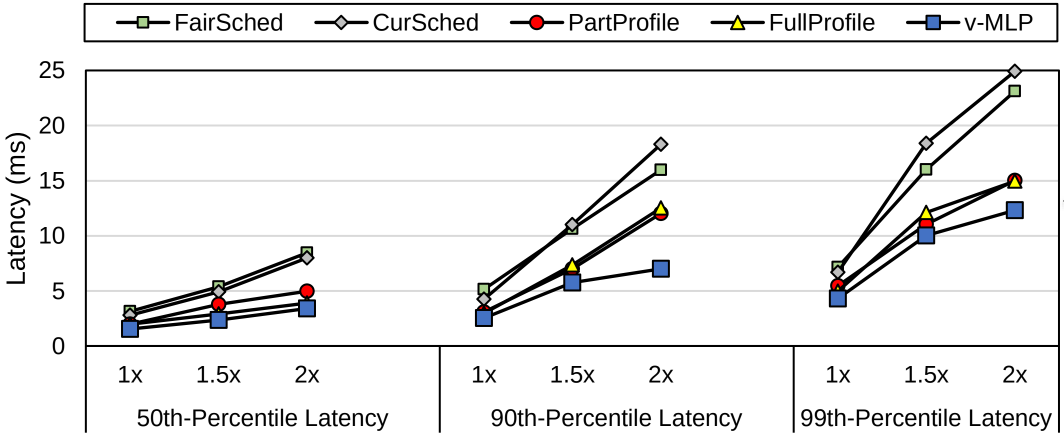


- v-MLP can maintain the **QoS requirements** of microservices.
- v-MLP works better under **periodic load and high V_r requests.**

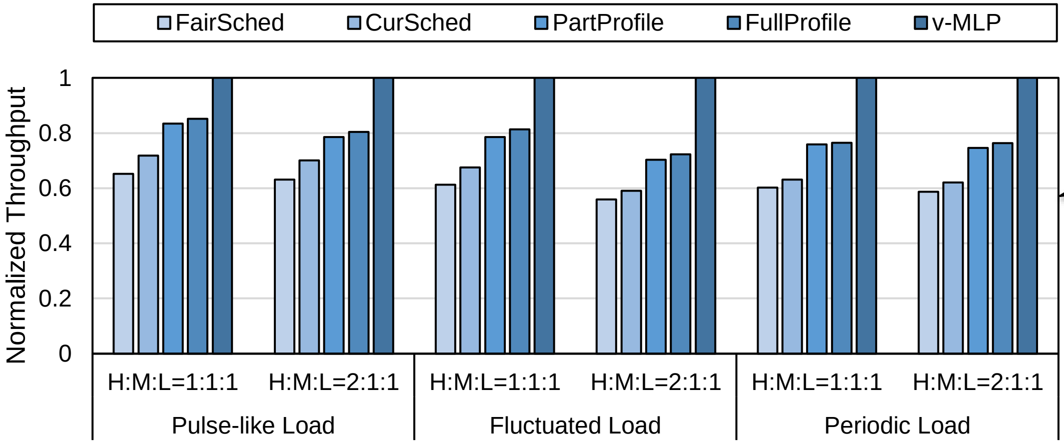


- v-MLP achieves **higher** resource utilization from beginning.
- v-MLP can maintain the resource utilization with **workload peaks**.

Evaluation Results: Performance



- v-MLP can **reduce request latency** at each percentile.
- v-MLP works better under **high load and 99th tail latency.**



- v-MLP achieves **higher throughput**.
- v-MLP works better under **periodic load and high V_r requests.**

Contents

1. Introduction: Monolithic to Microservice
2. Motivation: Microservice Characterization
3. Microservice Level Parallelism
4. Evaluation: Effectiveness, Efficiency, Performance
5. **Conclusions and Future Work**

Conclusions

- **Insights**
  - Microservice characteristics
  - Potential of new parallelism
- **Microservice Level Parallelism**
  - Interface layer between upper and lower
  - Coordinate various microservice chains
  - Tackle the uncertainty in dynamic cloud
- **Help for next-generation cloud-native design.**
- **We will expand MLP towards more directions in future.**

Thank You!

Exploring Efficient Microservice Level Parallelism

Xinkai Wang, Chao Li, Lu Zhang, Xiaofeng Hou, Quan Chen, Minyi Guo
unbreakablewxk@sjtu.edu.cn