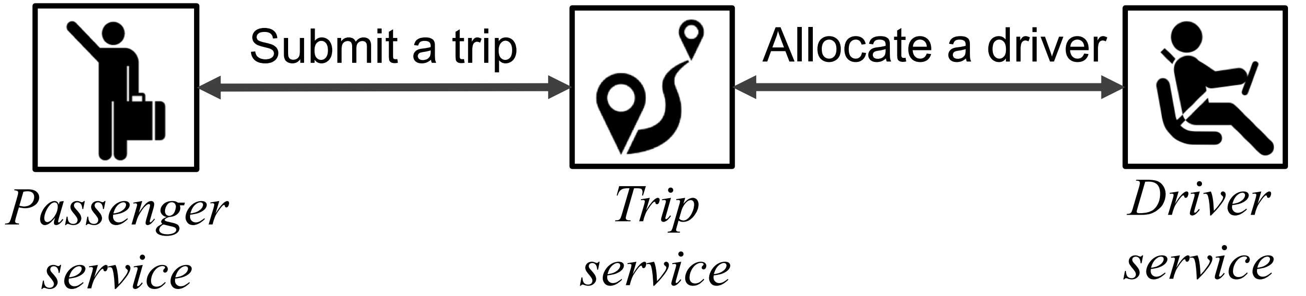
Load Balancing across Microservices

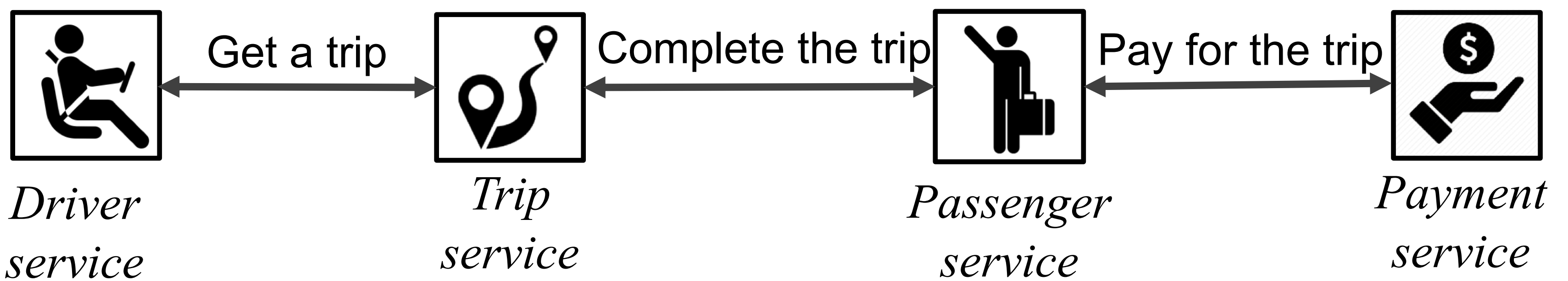
Yipei Niu[1], Fangming Liu[1], Zongpeng Li[2]

Email: fmliu@hust.edu.cn

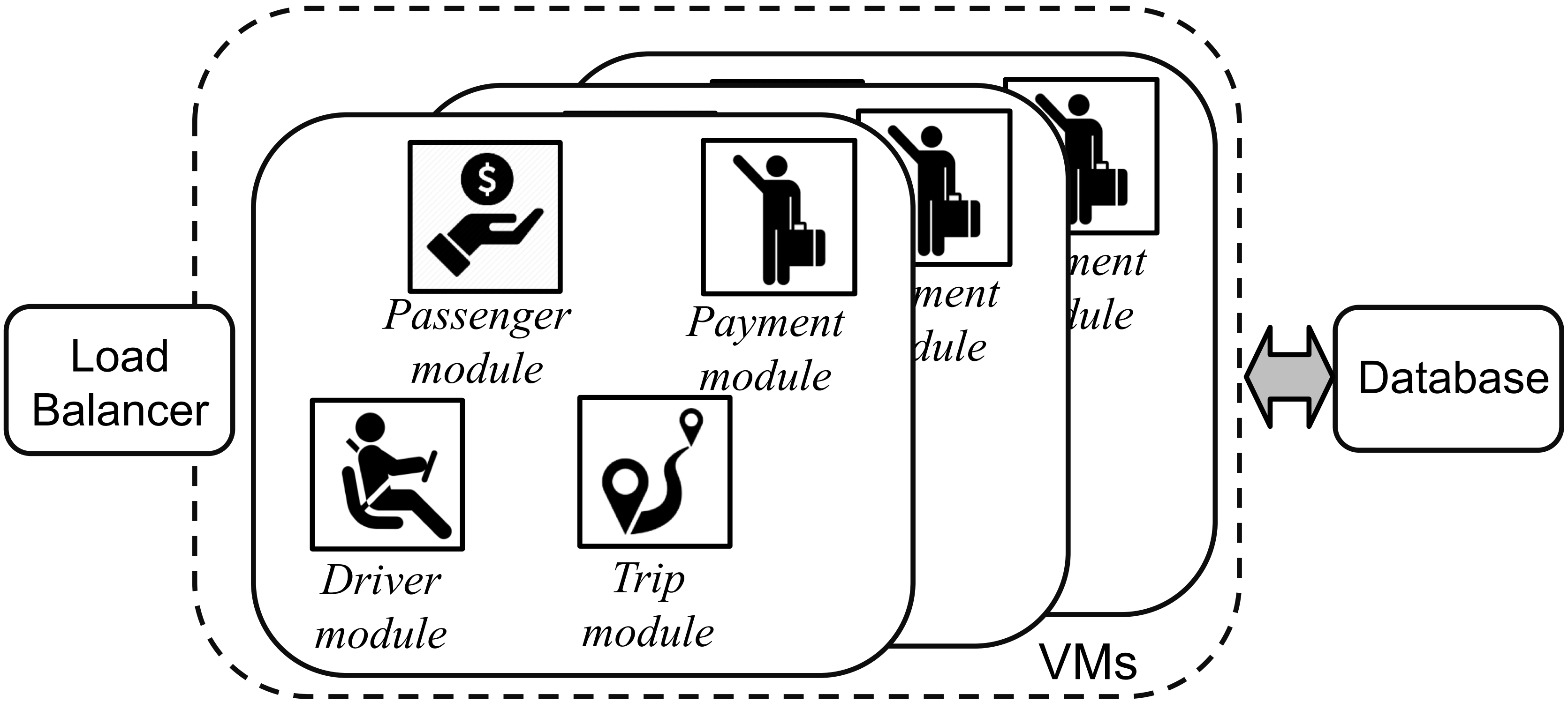
[1]*Cloud Datacenter & Green Computing/Communications Research Group*
[1]*Huazhong University of Science & Technology*
[2]*University of Calgary*

Let’s hail a cab





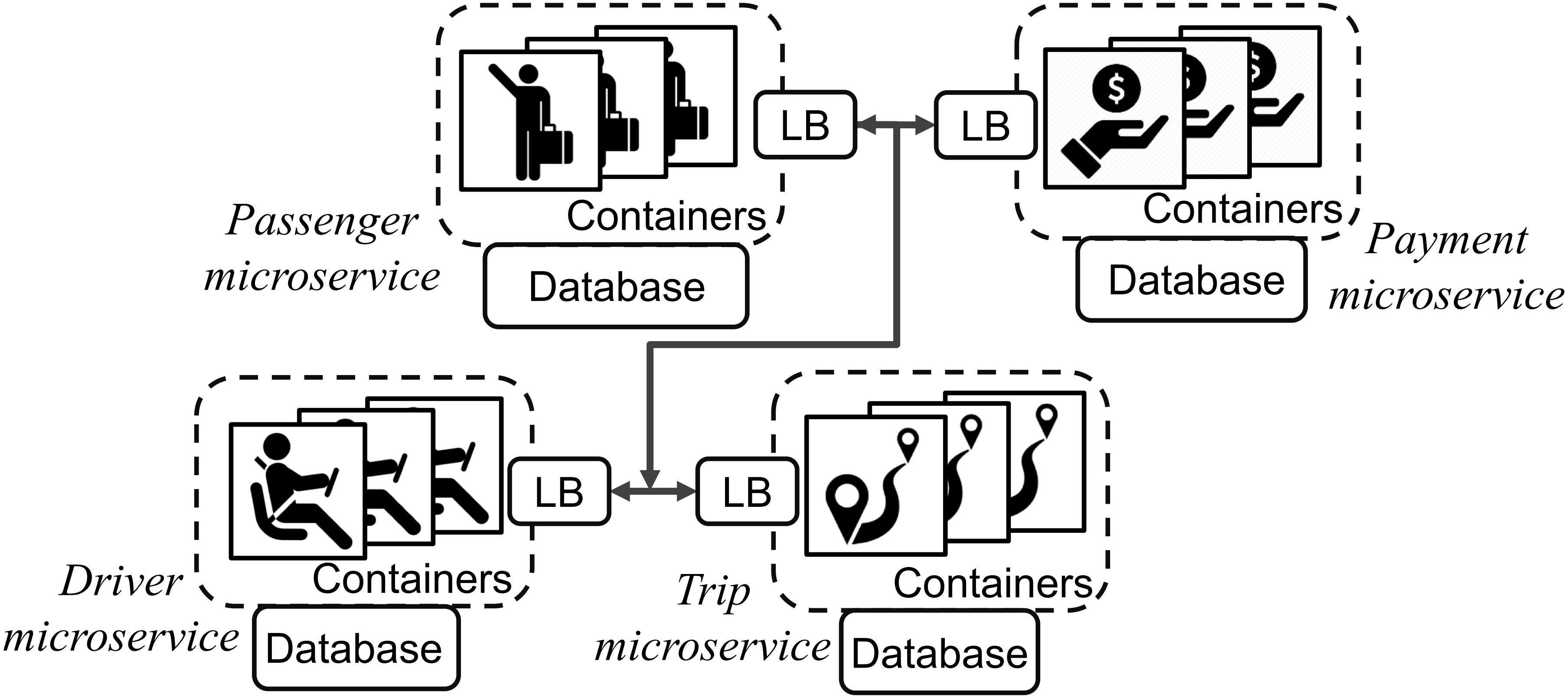
Toward monolithic hell



- The monolithic architecture
  - Services and modules are tightly coupled

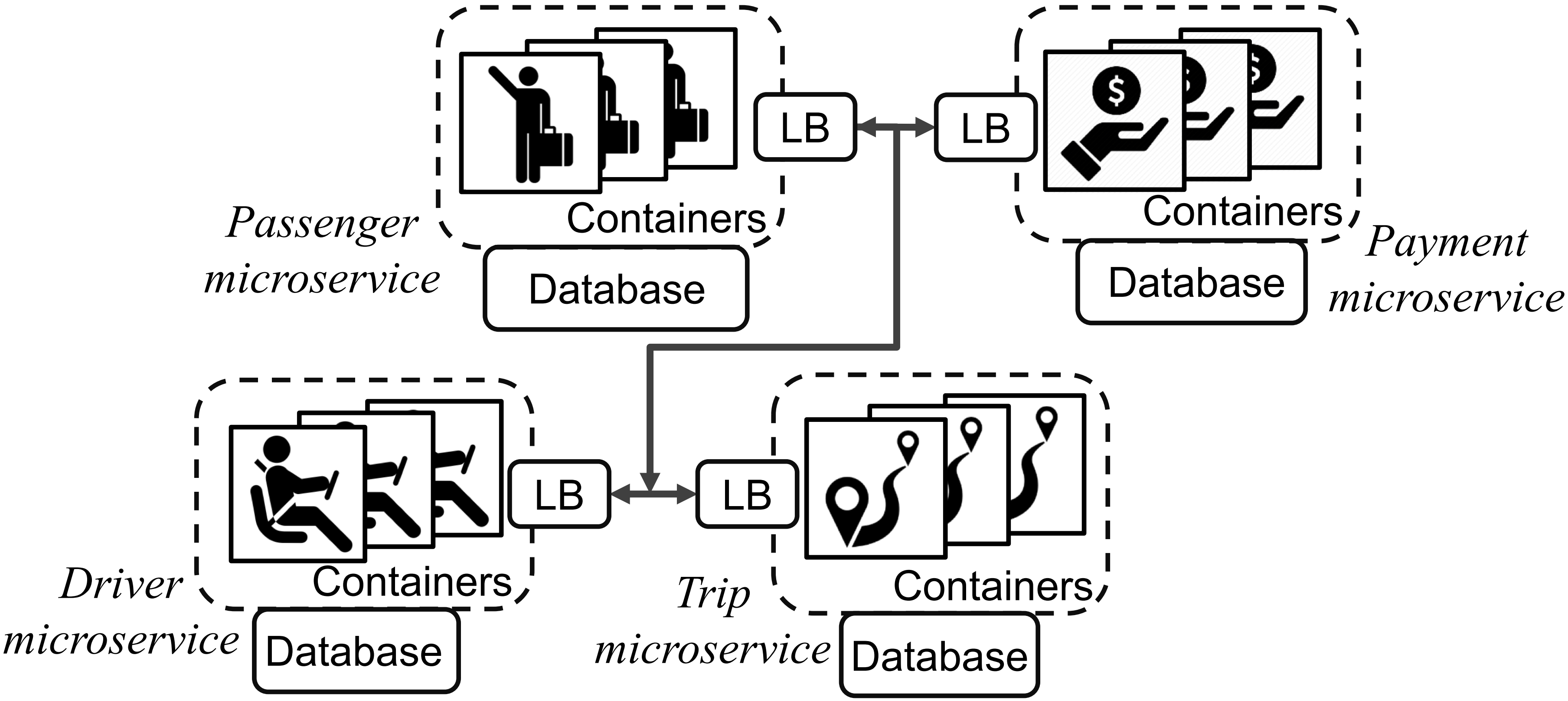
Complex to maintain, debug, and evolve!

Breaking monolithic software into microservices



A development technique that structures an monolithic application as a collection of loosely coupled services

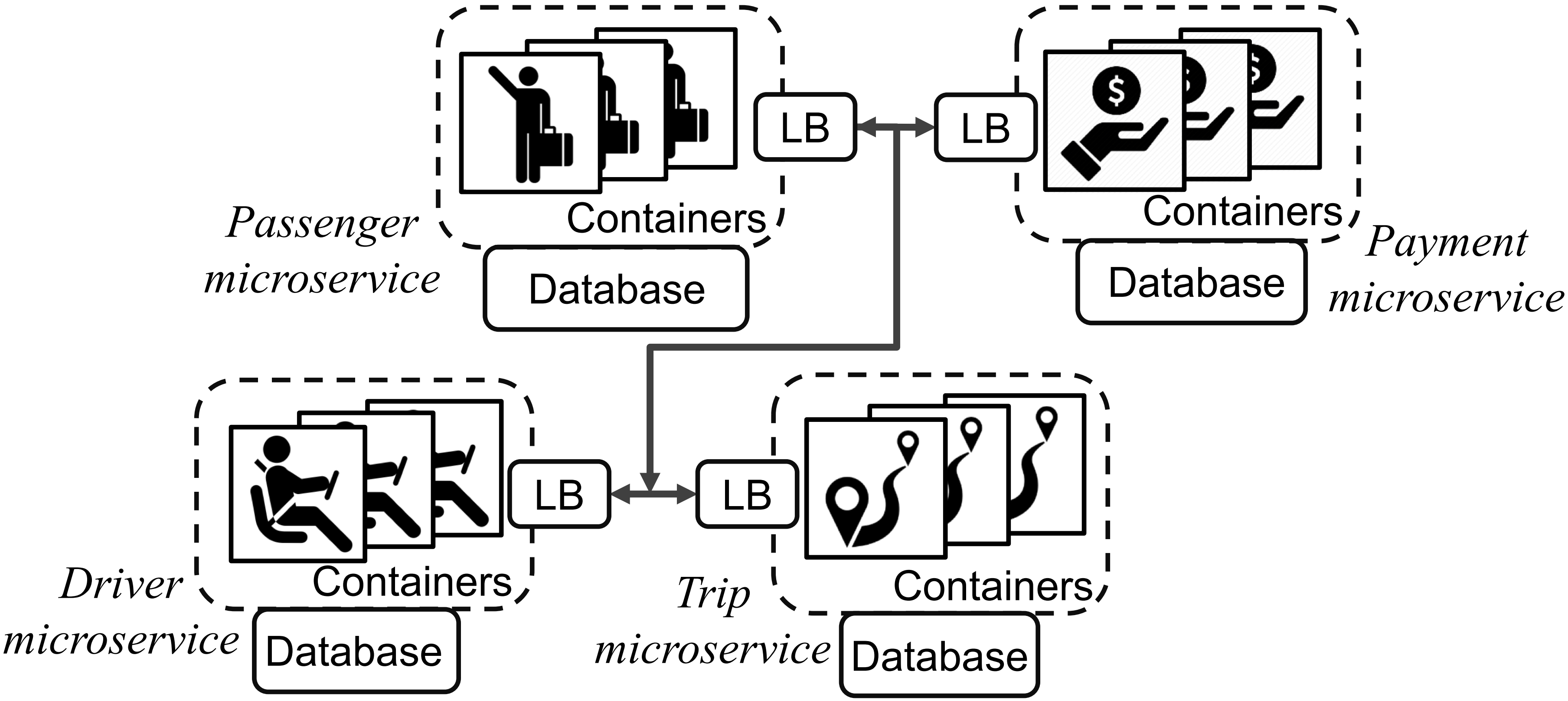
Breaking monolithic software into microservices



- Microservice has been adopted by Amazon, Netfix, and Uber

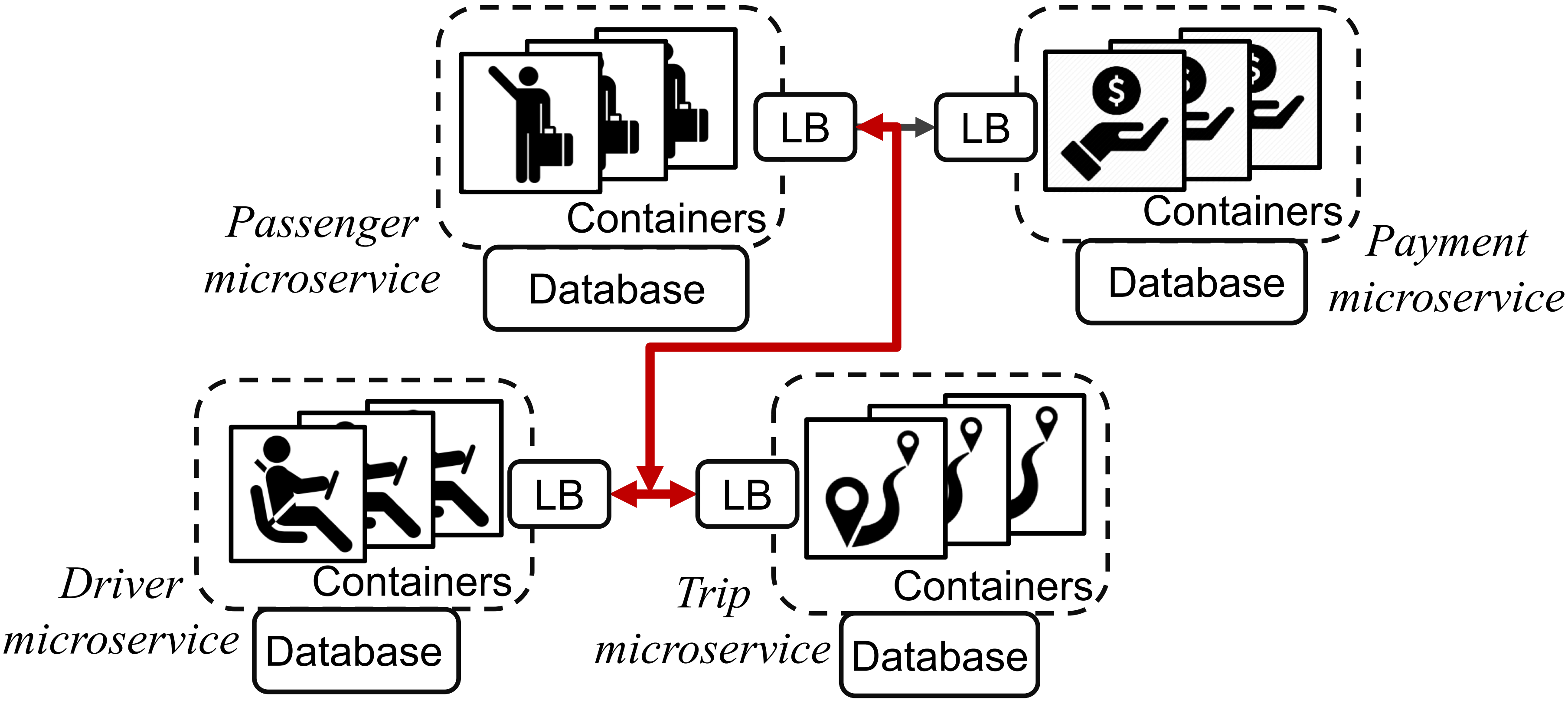
Source: https://thenewstack.io/ led- amazon- microservices- architecture/
https://www.nginx.com/blog/ microservices- at- netflix- architectural- best- practices/
https://eng.uber.com/soa/

Breaking monolithic software into microservices



- Each microservice has limited functionalities
- Instances runs independently on containers
- Requests are served by microservice chains

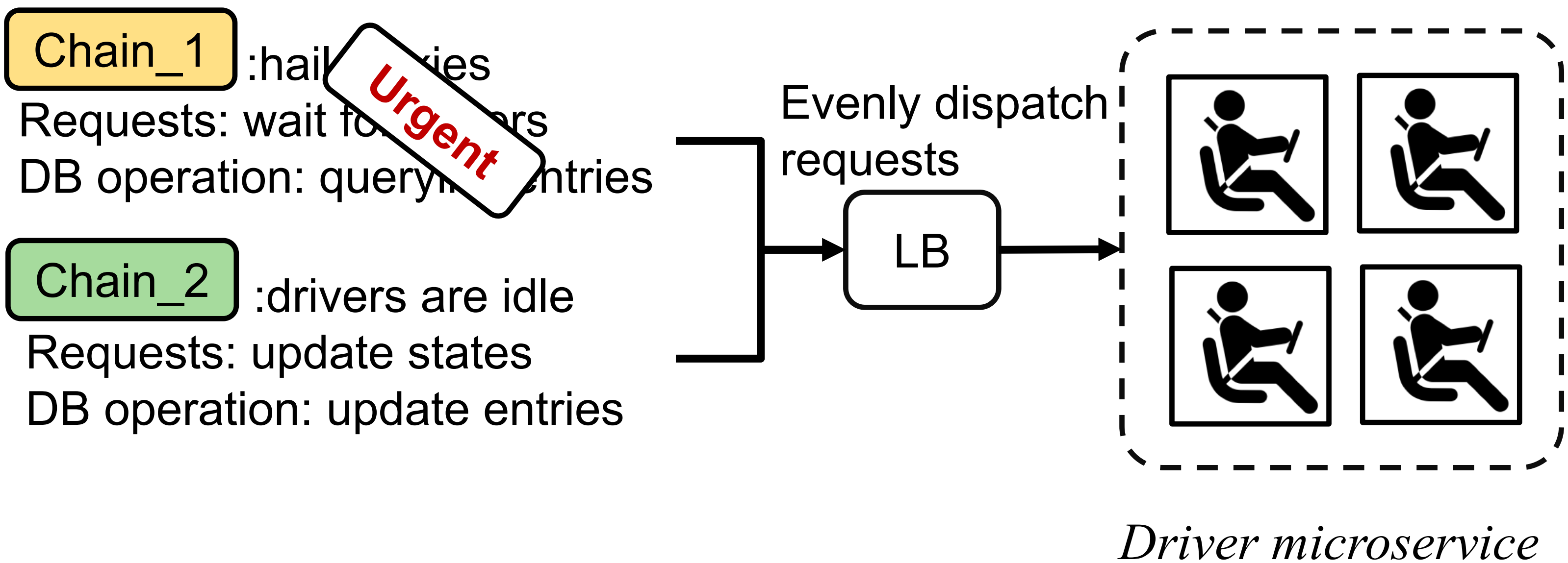
Breaking monolithic software into microservices



- Each microservice has limited functionalities
- Instances runs independently on containers
- Requests are served by microservice chains

Imbalance load across microservices

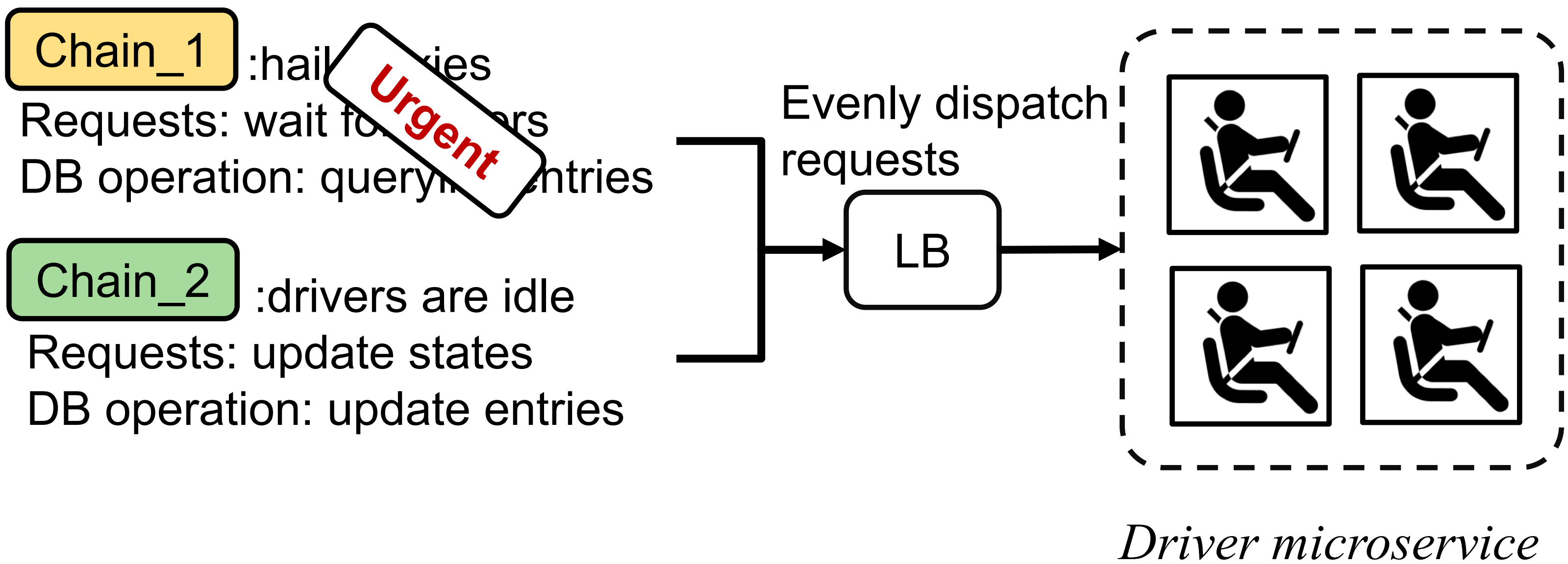
- Workload is fluctuating
- Different QoS of chains
- Different service time in microservices



We need to balance load from perspective of chains!

Imbalance load across microservices

- Workload is fluctuating
- Different QoS of chains
- Different service time in microservices



How to identify chains?

How to determine instance scale of a chain?

Tackling the complexity

- Typical load balancer
  - Haproxy, Nginx

Fail to identify different chains

- Communication pattern
  - Application-layer protocols
    - HTTP for RESTful API
    - AMQP* for message queue

Complicated to operate

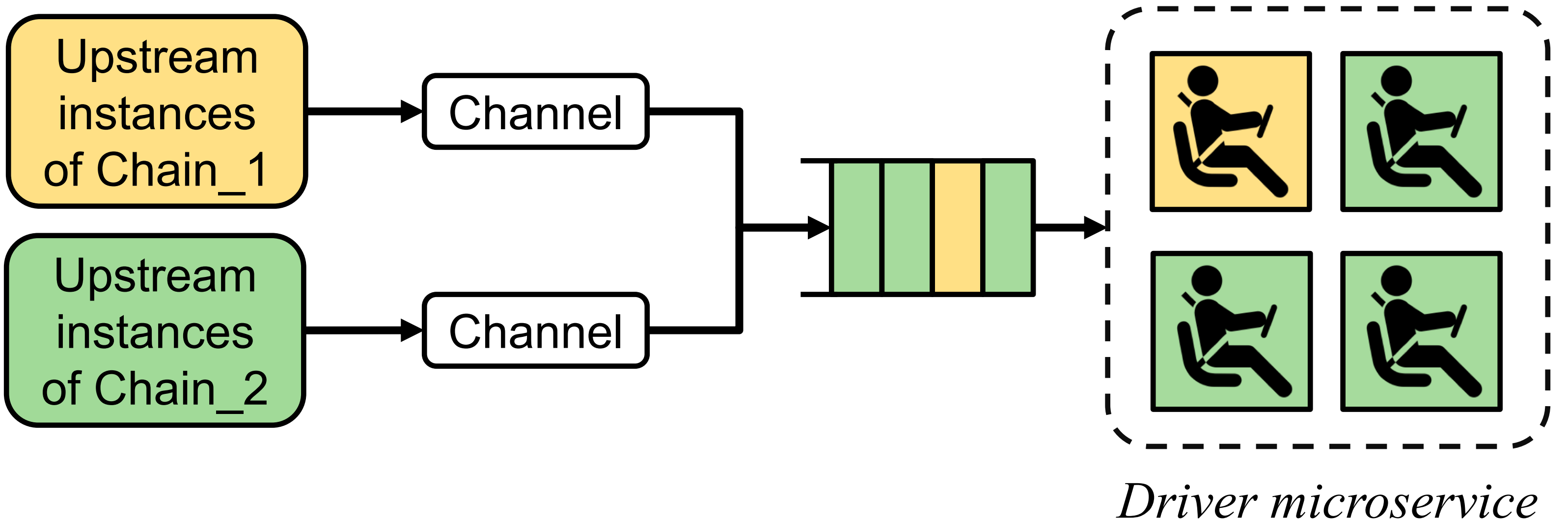
- Large scale
  - Netflix employs over 600 microservices

An efficient strategy is required

*AMQP: Advanced Message Queueing Protocol

Abandon load balancer

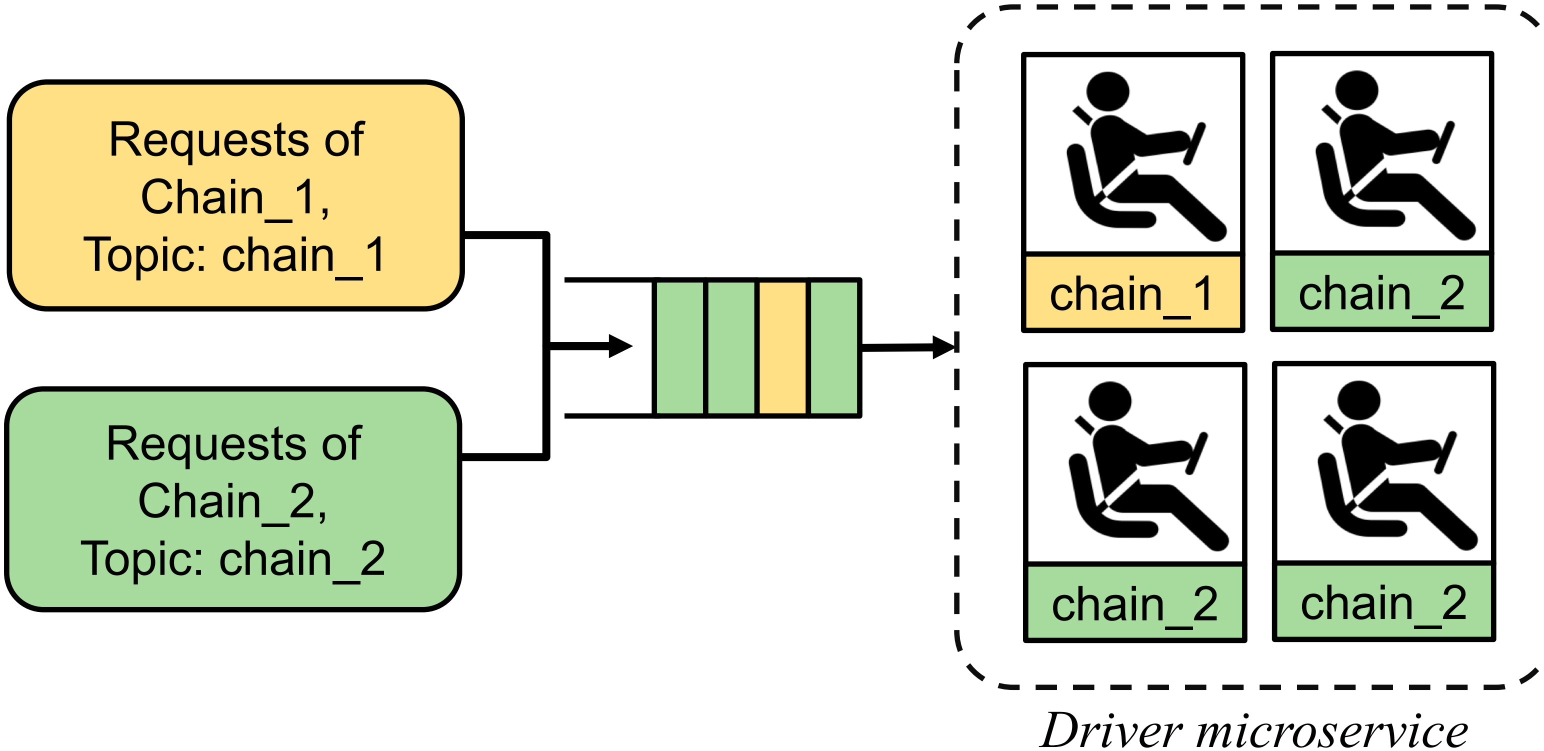
- Replace load balancer with message queue
- Upstream instances declare a channel
- Downstream instances are aware of what messages (requests) to process



We can purely employ message queue

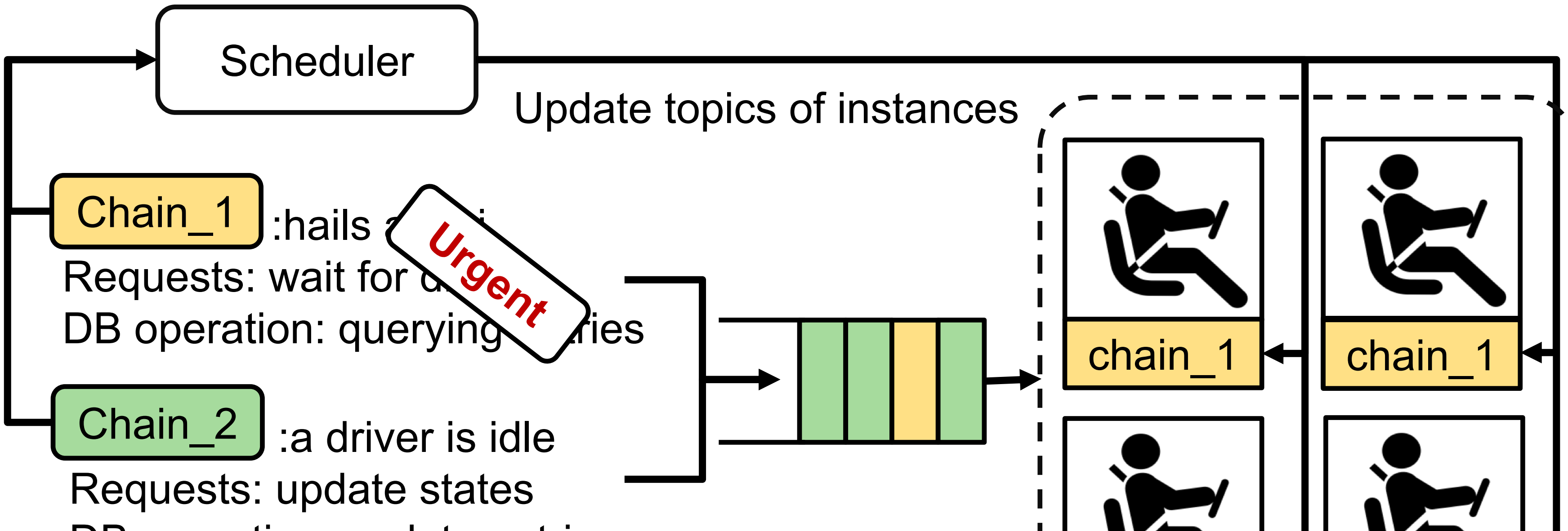
How to identify chains?

- Employ topic mode of RabbitMQ
- By marking instances with different topics



How to determine instance scale?

- Allocate more instances to urgent chain
  - Based on request arrival rate
  - Based on service times of microservices



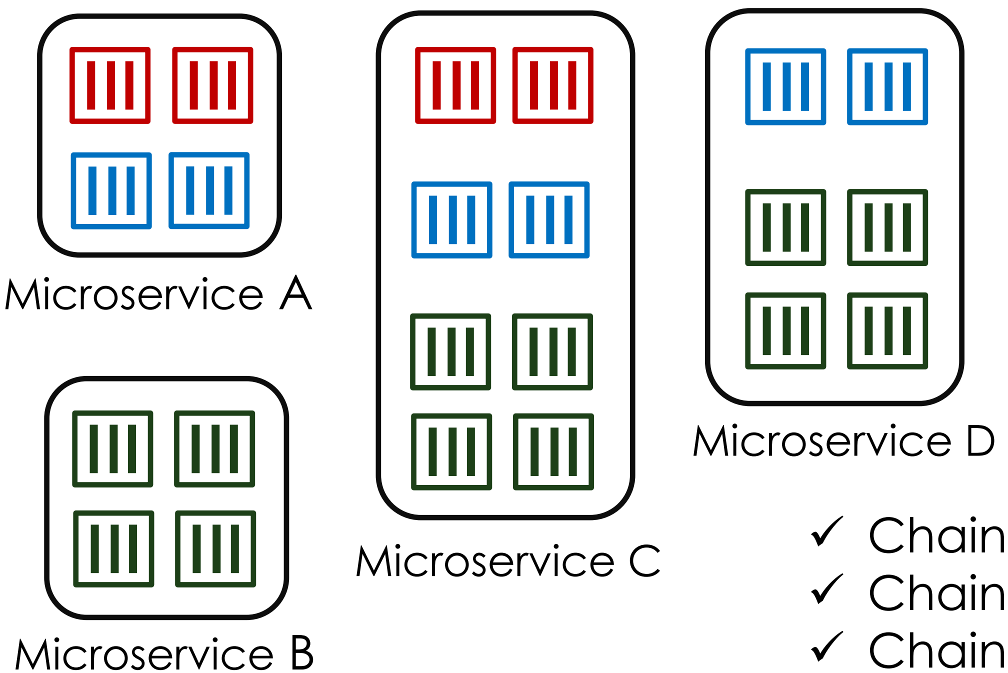
Sub-problem 1: evaluate response time of a chain

Sub-problem 2: coordinate instance allocation across chains

Driver microservice

Modeling microservice systems

- Matrix $R = (r_{c,m})_{C \times M}$
  - C is the total number of microservice chains
  - M is the number of microservice types
  - $r_{c,m} \in \{0, 1\}$ is whether chain **c** traverses microservice ***m***

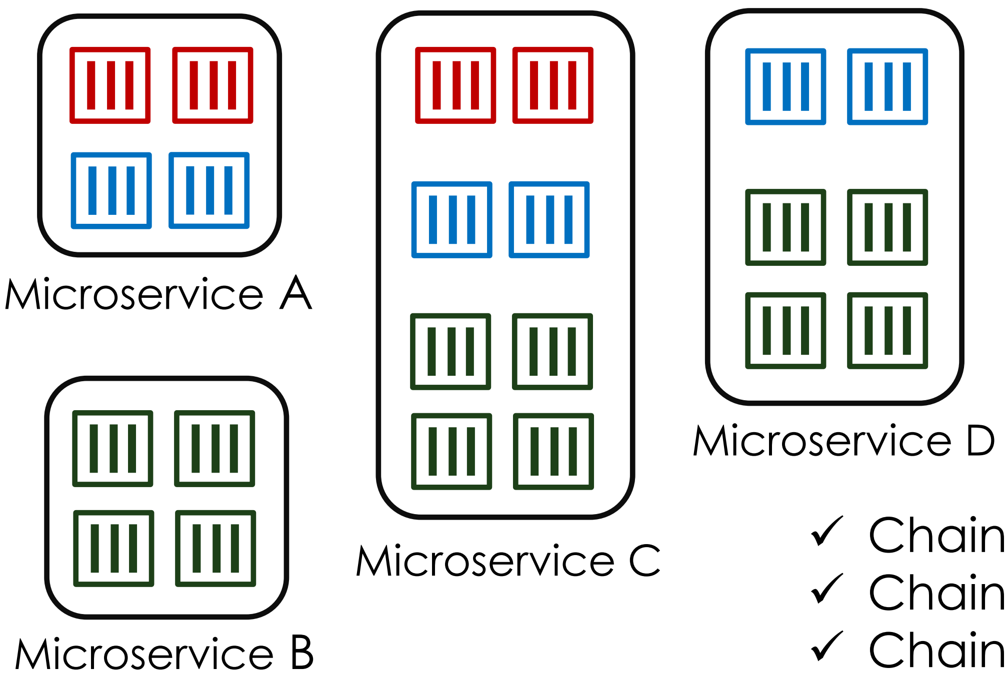


$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}_{3 \times 4}$$

- ✓ Chain Red: **A -> C**
- ✓ Chain Blue: **A -> C -> D**
- ✓ Chain Green: **B -> C -> D**

Allocating instances for chains

- Matrix $S = (s_{c,m})_{C \times M}$
  - $s_{c,m}$ denotes the number of microservice ***m*** instances assigned to chain ***c***



$$\begin{bmatrix} 2 & 0 & 2 & 0 \\ 2 & 2 & 0 & 2 \\ 0 & 4 & 4 & 4 \end{bmatrix}_{3 \times 4}$$

✓ Chain Red: **A -> C**
✓ Chain Blue: **A -> C -> D**
✓ Chain Green: **B -> C -> D**

Evaluating response time

- Model single microservice as an M/G/1 queue
  - Request arrival follows Poison process
  - Service time is generally distributed
  - Response time in of a microservice

$$u_{c,m}(s_{c,m}) = \frac{E[Z^{c,m}]}{1 - \rho_{c,m}\boxed{(s_{c,m})}}$$

- Based on Theorem 1, single microservice is extended to a chain
  - Response time of a chain

$$u_c(\boldsymbol{s}_c) = \sum_{m=1}^{M} r_{c,m} \cdot u_{c,m}\boxed{(s_{c,m})}$$

where $r_{c,m}$ indicates whether chain c passes microservice m.

Determining instance allocation

- Imagine every chain is a person
- Negotiate with each other

Chain Red: my request rate increases, I am urgent! Can any chain release some instances of ***A*** or ***C*** for me?

Chain Blue: I have instances of ***A*** and ***C***, I can release some for you. But I can hardly guarantee response times, I need more instance ***D***.

Chain Green: OK, I can release some instances of ***D***, because I have abandon instance ***B***.

Chain Blue

Chain Red

Chain Green

- ✓ Chain Red: **A -> C**
- ✓ Chain Blue: **A -> C -> D**
- ✓ Chain Green: **B -> C -> D**

Game theory based instance allocation

- Utility function
  - We introduce a predefined worst response time U_c^b
  - $U_c^b = u_c(\boldsymbol{s}_c^b)$ and $\boldsymbol{s}_c^b$ is the disagreement instance allocation of chain **c**
  - Utility function is defined as follows

$$\frac{U_c^b - u_c}{U_c^b}$$

- Utilizing Nash bargaining solution for instance allocation

$$\boxed{\max_{u^c} \quad \prod_{u^c \in U} \frac{U_c^b - u_c}{U_c^b}}$$

- There exists a Nash Bargaining solution solve the above problem

COLBA: Chain-Oriented Load Balancing Algorithm

- By relaxing integrality constraints
- The original problem is reduced to convex optimization

$$\max_{\boldsymbol{s}_c} \quad \sum_{c=1}^{C} \ln \frac{U_c^b - u_c}{U_c^b}$$

Equivalent to the original problem

$$\text{s.t.} \quad \sum_{c=1}^{C} r_{c,m} \cdot s_{c,m} \leq I^m,$$

Instance capacity limit

$$s_{c,m} > \frac{\lambda^{c,m}}{\mu_{c,m}},$$

Ensure traffic intensity of queues

- KKT-conditions with rounding

Optimality analysis

- The optimal value of objective function: ϕ^*
- The optimal value of objective function under relaxation: $\tilde{\phi} > \phi^*$
  - The solution is $\tilde{s}_{c,m}$
- The value of objective function under relexation with rounding ϕ
  - The solution is $\hat{s}_{c,m} = \lfloor \tilde{s}_{c,m} \rfloor$
  - So we have:

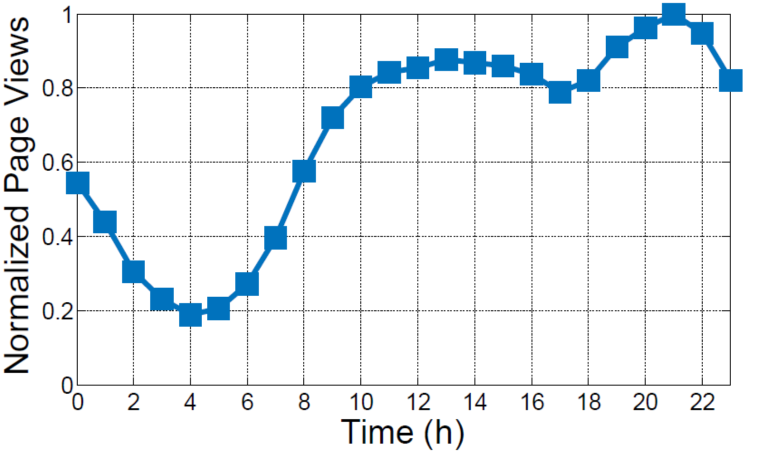
$$\boxed{\phi^* - \phi < \tilde{\phi} - \phi}$$

  - Finally we prove that

$$\boxed{\phi^* - \phi < \sum_{c=1}^{C} \ln(1 + M \cdot Q_c)}$$

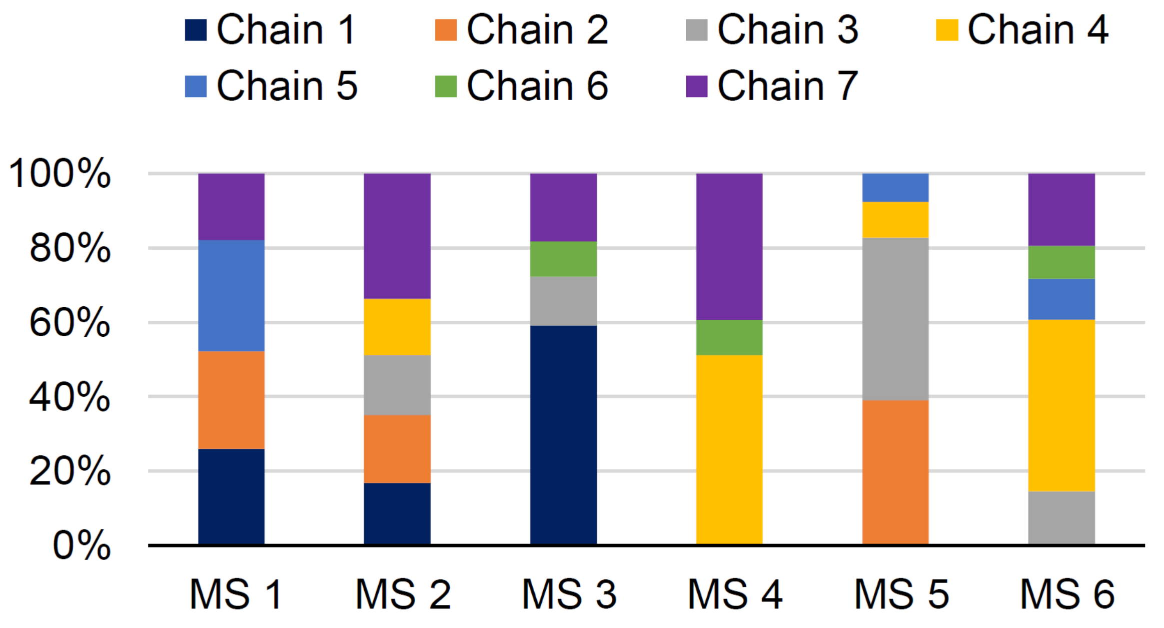
Simulation setup

- Real world trace
  - Online traffic in U.S. on Cyber Monday measured by Akamai



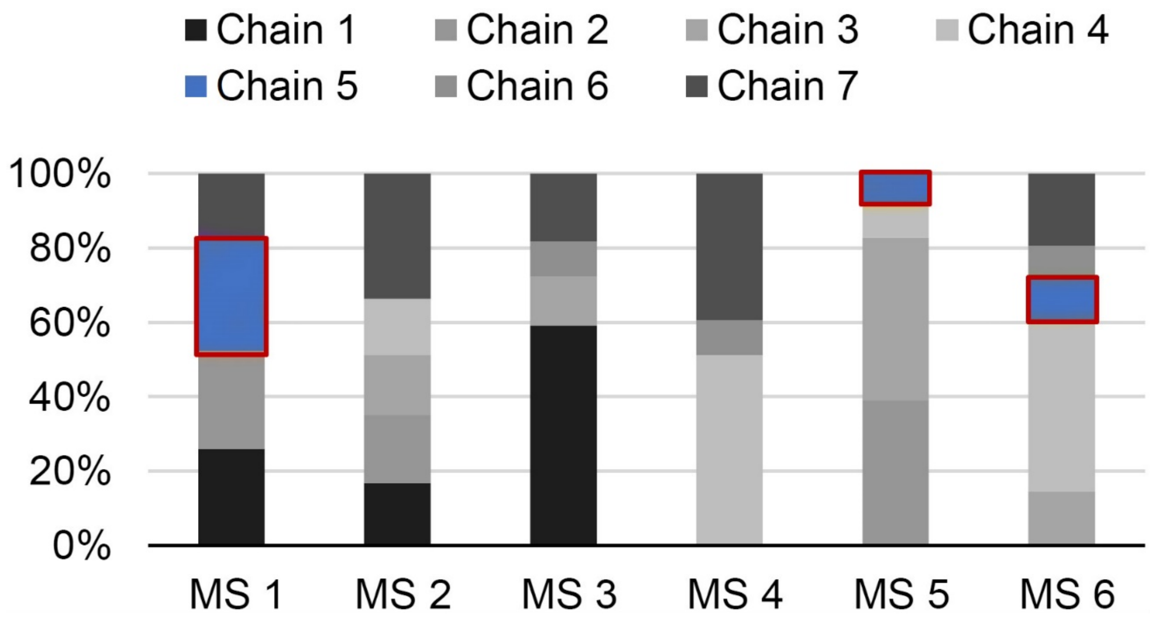
- Baselines
  - Instance-oriented load balancing
    - Apply the chain-oriented load balancing strategy for fairness.
  - Microservice-oriented load balancing
- Worst response time U_c^b
  - The upper bound of response times
  - The performance requirement of user requests in chain c
  - The initial instance assignment for each chain

Request arrival rate



Worst response time U=[10; 8; 6; 7; 2; 4; 2], Request rate: 1.0λ

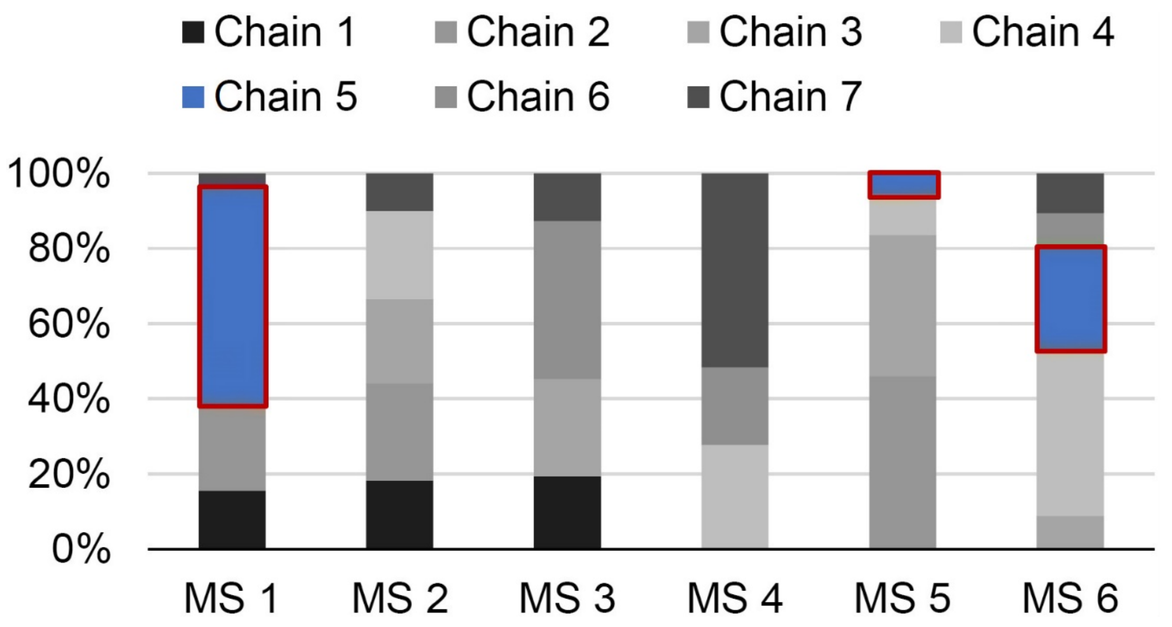
Request arrival rate



Worst response time U=[10; 8; 6; 7; **2**; 4; 2], Request rate: 1.0λ

- For Chain 5, the instances allocated to it increase significantly so as to overcome the bursty workload

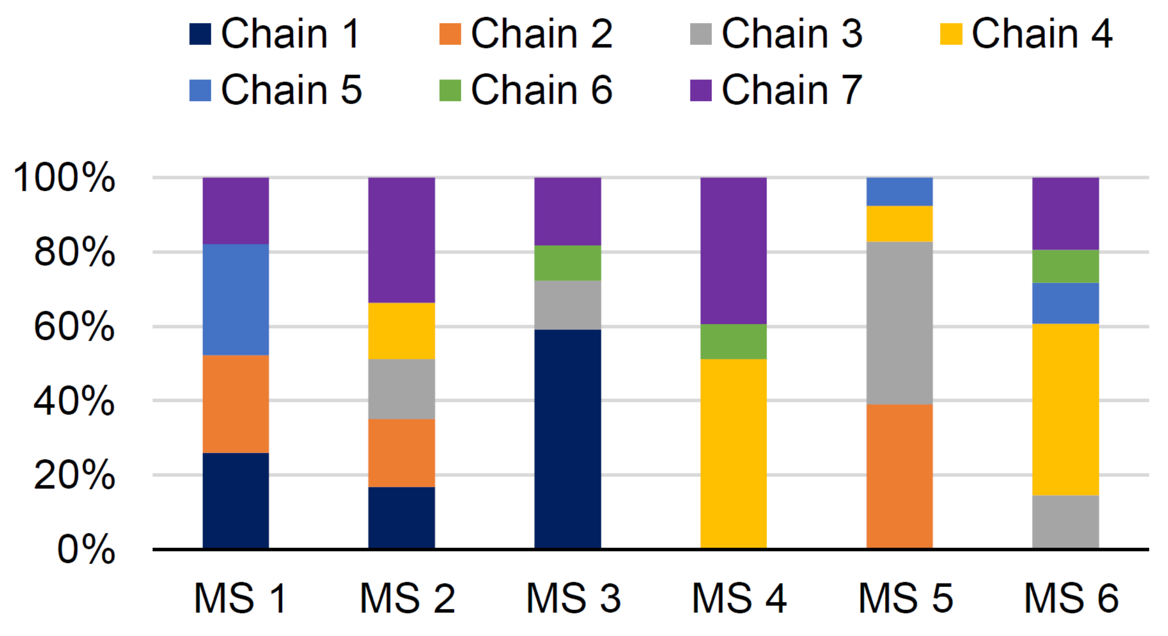
Request arrival rate



Worst response time U=[10; 8; 6; 7; **2**; 4; 2], Request rate: 1.5λ

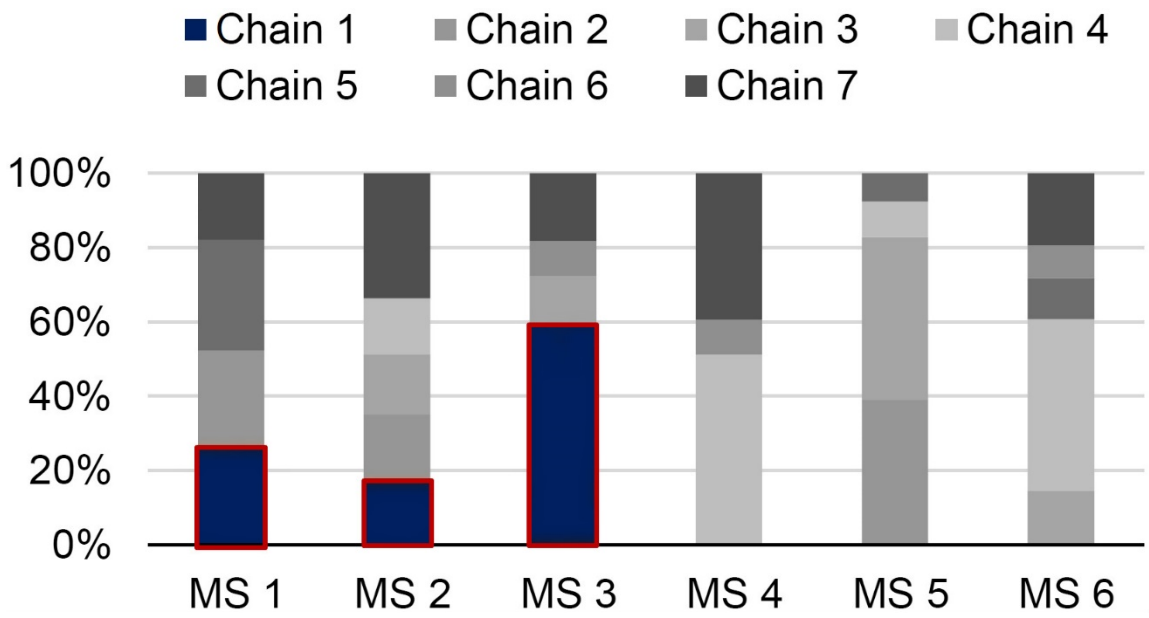
- For Chain 5, the instances allocated to it increase significantly so as to overcome the bursty workload

Request arrival rate



Worst response time U=[**10**; 8; 6; 7; 2; 4; 2], Request rate: 1.0λ

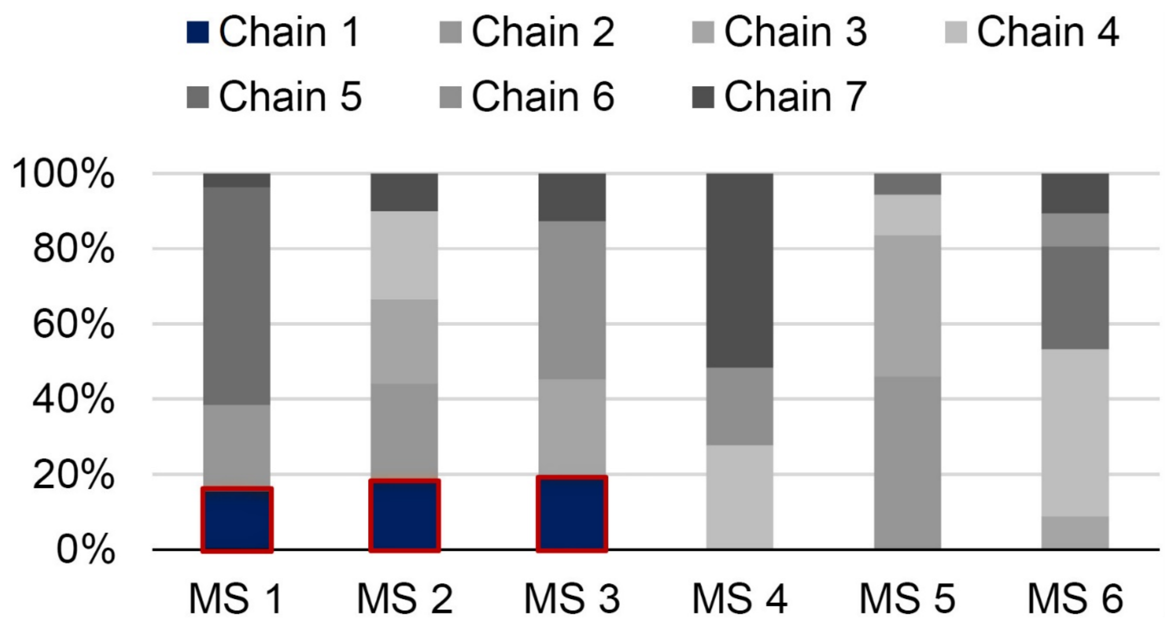
Request arrival rate



Worst response time U=[**10**; 8; 6; 7; 2; 4; 2], Request rate: 1.0λ

- Number of instances assigned to Chains 1 decreases sharply, leading to increase in response time.

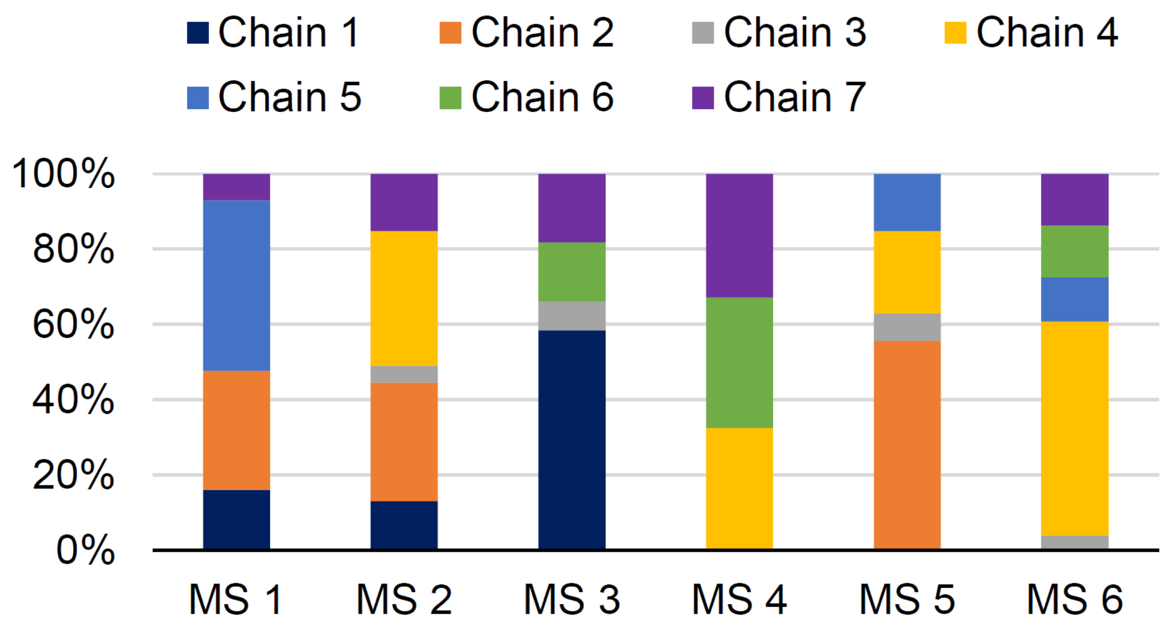
Request arrival rate



Worst response time U=[**10**; 8; 6; 7; 2; 4; 2], Request rate: 1.5λ

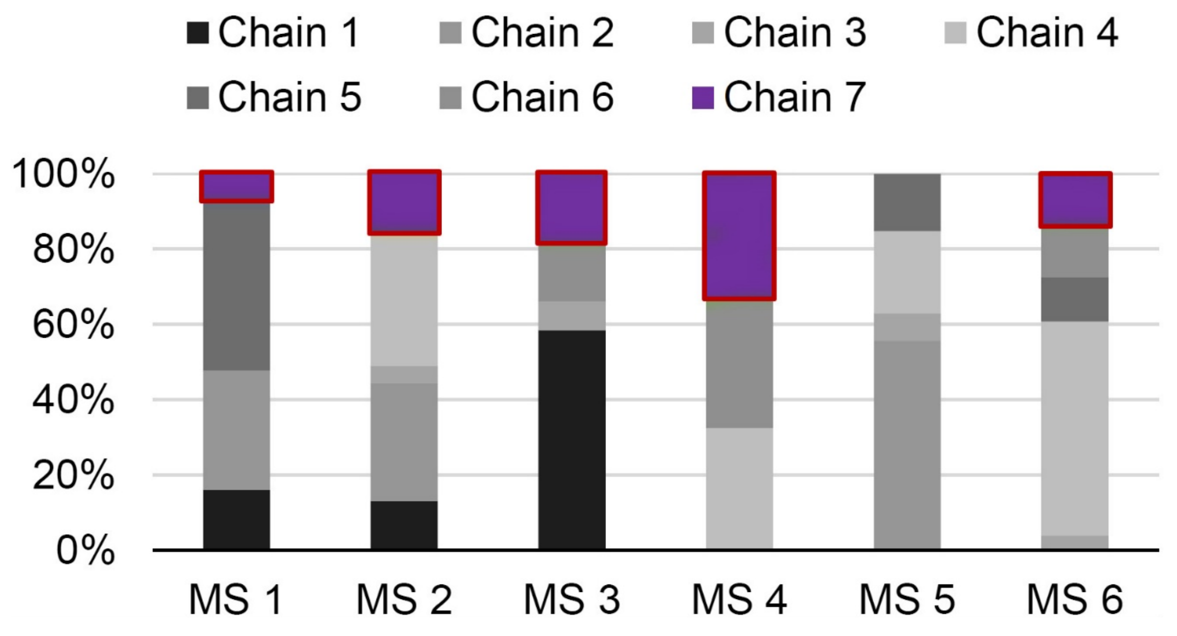
- Number of instances assigned to Chains 1 decreases sharply, leading to increase in response time.

Worst response time



Worst response time U=[3; 3; 2; **5**; 4; 2; **9**], Request rate: λ

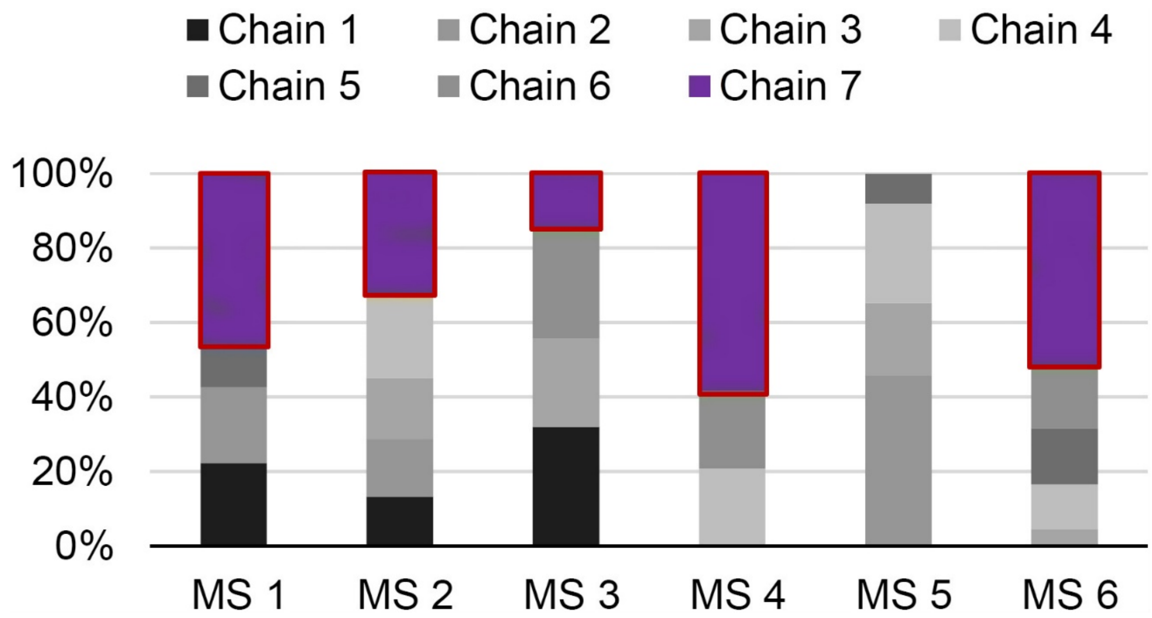
Worst response time



Worst response time U=[3; 3; 2; **5**; 4; 2; **9**], Request rate: λ

- The scale of instances allocated to Chain 7 increases remarkably, especially for Microservice 1 and 6

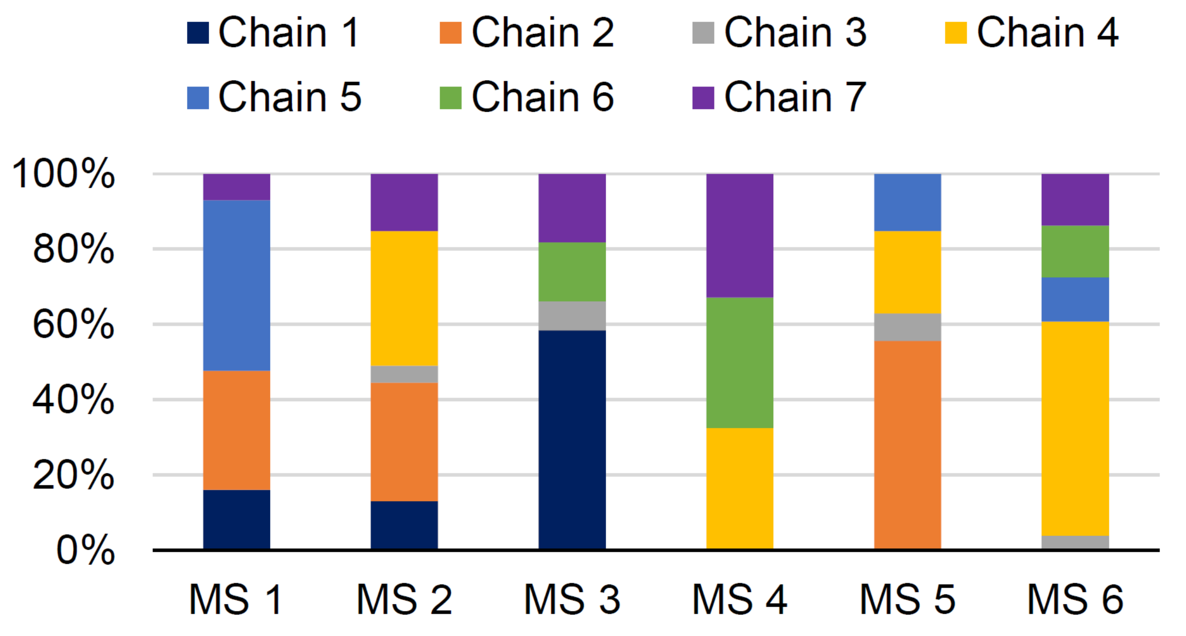
Worst response time



Worst response time U=[3; 3; 2; **9**; 4; 2; **5**], Request rate: λ

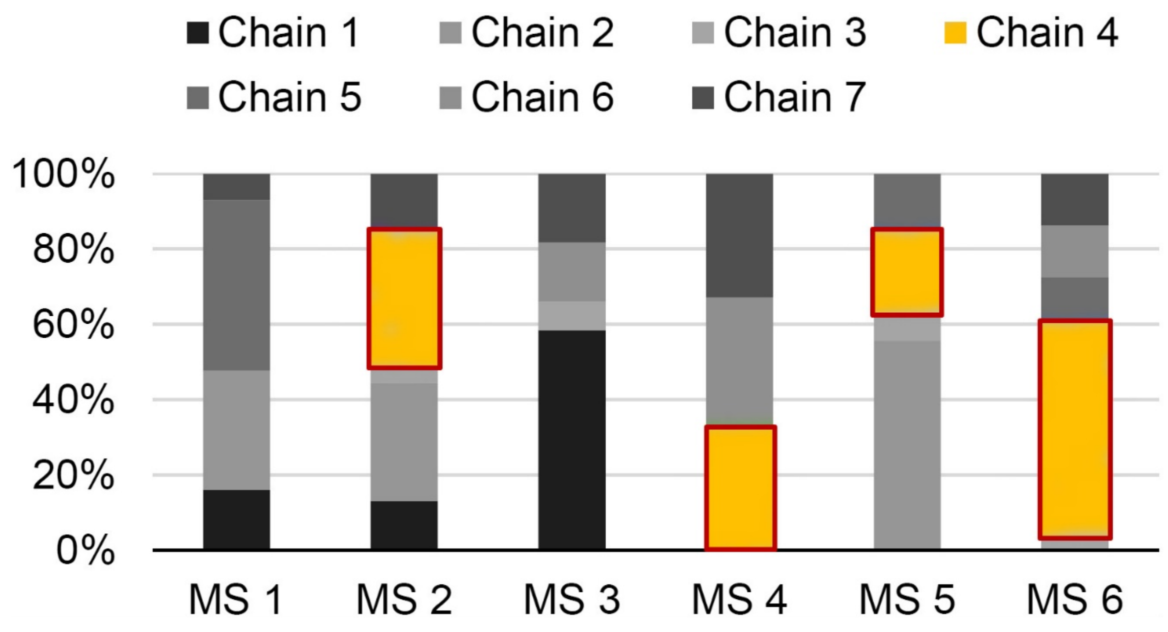
- The scale of instances allocated to Chain 7 increases remarkably, especially for Microservice 1 and 6

Worst response time



Worst response time U=[3; 3; 2; **5**; 4; 2; **9**], Request rate: λ

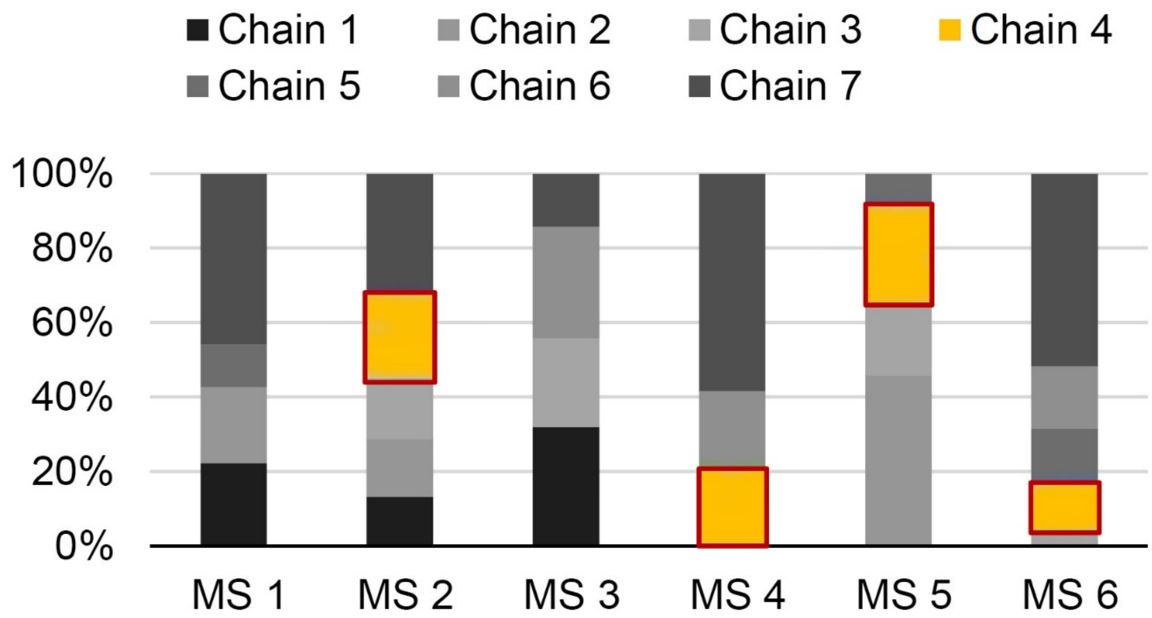
Worst response time



Worst response time U=[3; 3; 2; **5**; 4; 2; **9**], Request rate: λ

- Instance scale of Microservice 6 assigned to Chain 4 shrinks significantly

Worst response time



Worst response time U=[3; 3; 2; **9**; 4; 2; **5**], Request rate: λ

- Instance scale of Microservice 6 assigned to Chain 4 shrinks significantly

Conclusion

- We proposed COLBA to balance load from perspective of chains
- Leveraging message queues, requests can be efficiently balanced across microservice instances
- Simulation results show that COLBA can coordinate instance allocation, concerning expected QoS of chains as well as competition across chains

Q&A

Thank You!

"Cloud Datacenter & Green Computing" *Research Group*

Huazhong University of Science & Technology

http://grid.hust.edu.cn/fmliu/

fmliu@hust.edu.cn

Discussion & open problems

- Q: What are the differences between microservice chain and NFV chain?
  - A: Chain path is implemented in different layers.

- Q: Is it OK to abandon HTTP?
  - A: Modern web applications are developed using the Task Asynchronous Paradigm*.
  - The states of connections can also be enabled with message queue.

*Webperf: Evaluating what-if scenarios for cloud-hosted web applications," in Proc. of SIGCOMM, 2016.