ChinaSys 2024

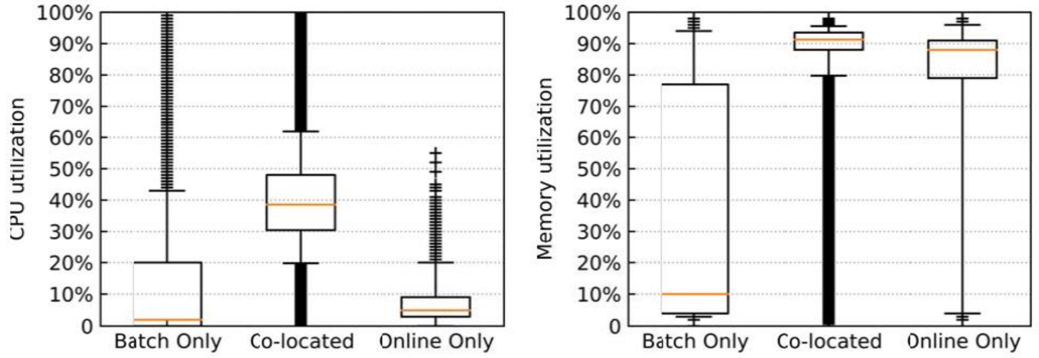
UFO: The Ultimate QoS-Aware CPU Core Management for Virtualized and Oversubscribed Public Clouds

***彭雅娟**，*陈爽，赵乙，喻之斌

CONTENT

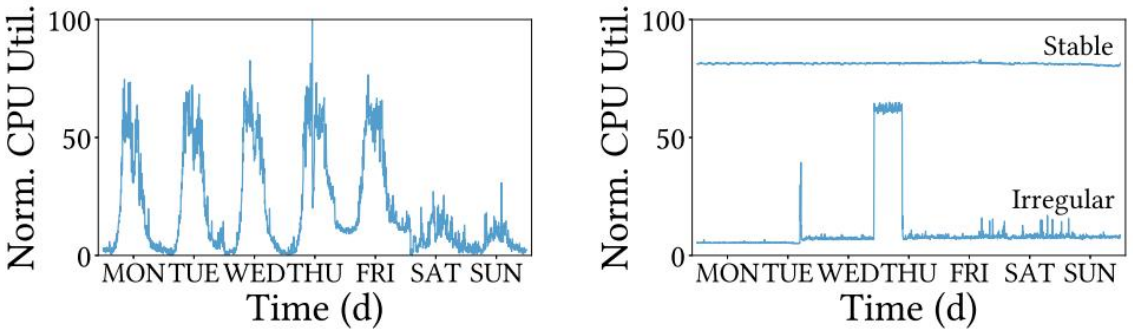
1 Background and Motivation

- The resource utilization in Cloud data center is very low!
- CPU utilization: **5 - 15 %**



(a) CPU usage. (b) Memory usage.

Alibaba Data Center
(Jing Guo: IWQoS'19)



(a) Diurnal (b) Stable and Irregular

VM CPU utilization patterns in Azure Cloud
(Xiaoting Qin: DSN'23)

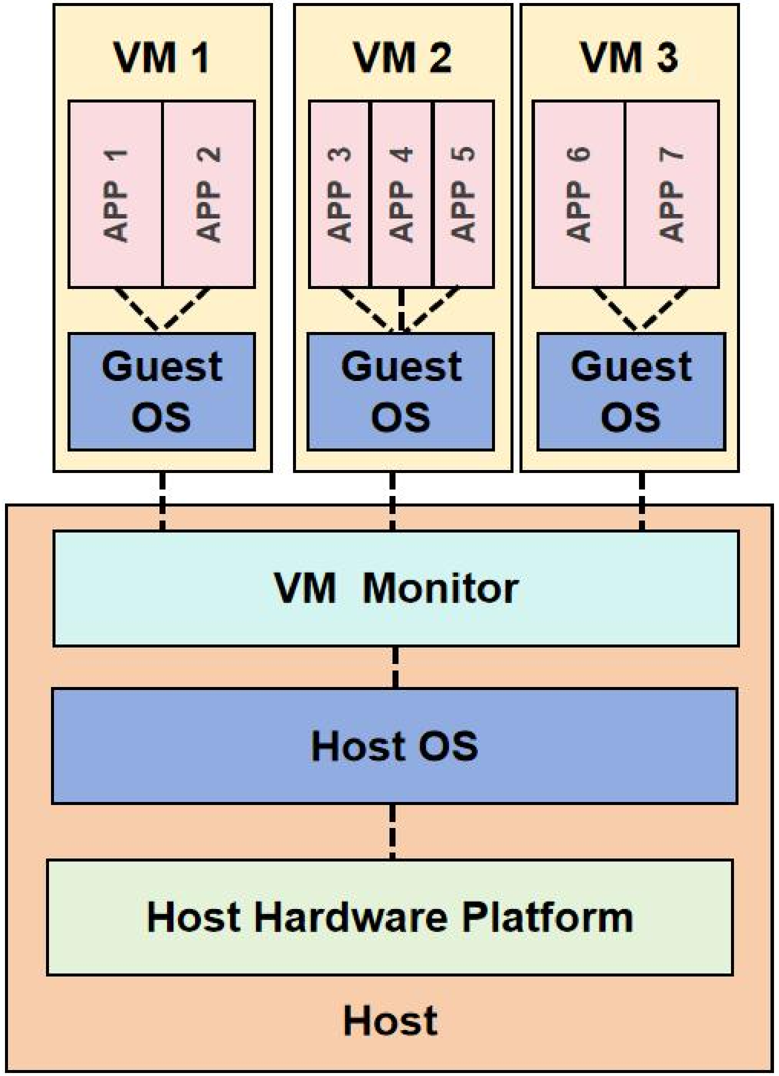
Multi-tenancy

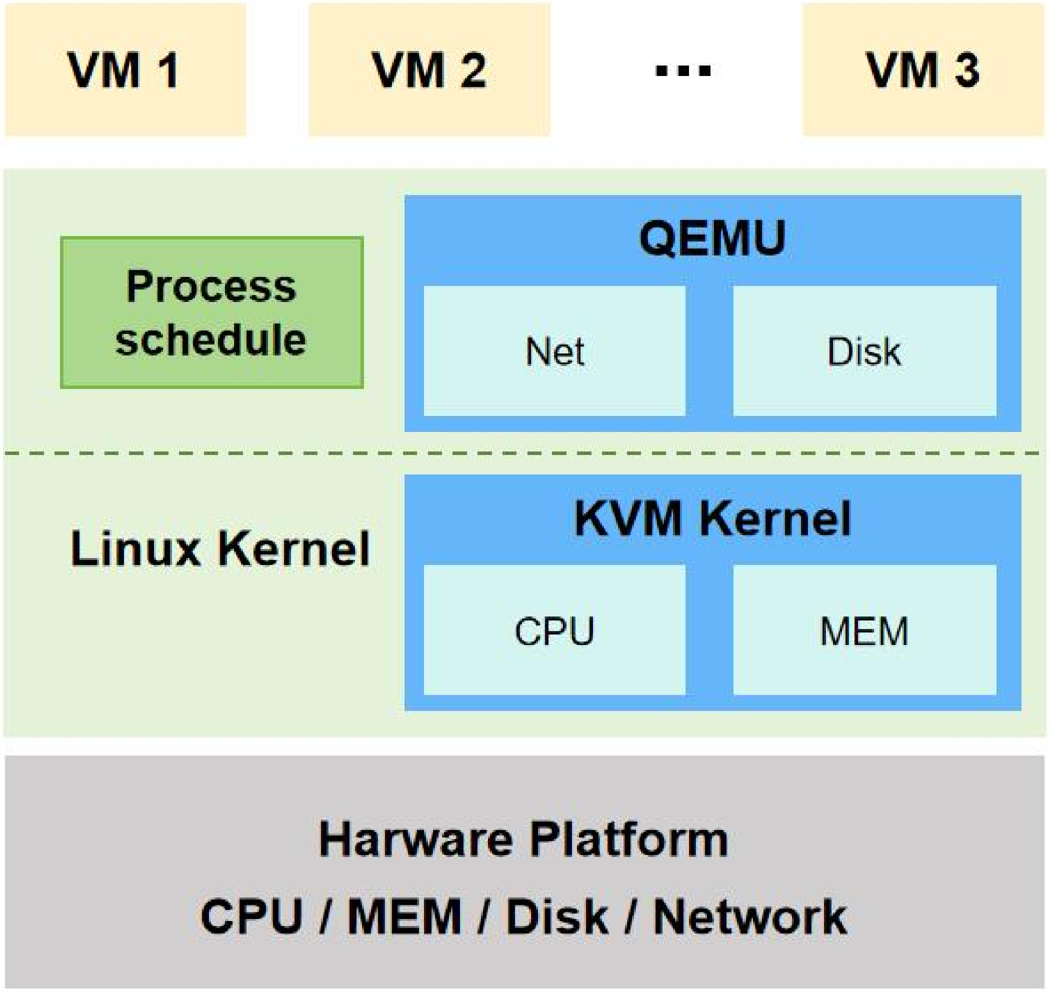
Virtualization

Oversubscription

① Background and Motivation

Virtualization



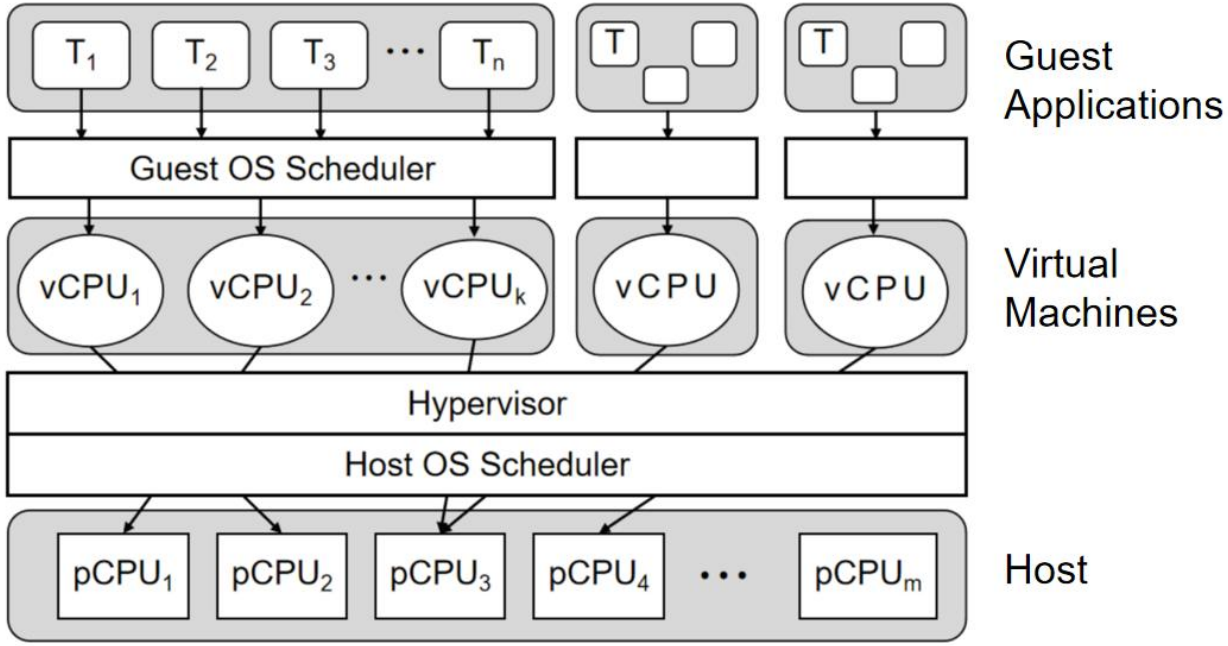


Public Cloud Applications

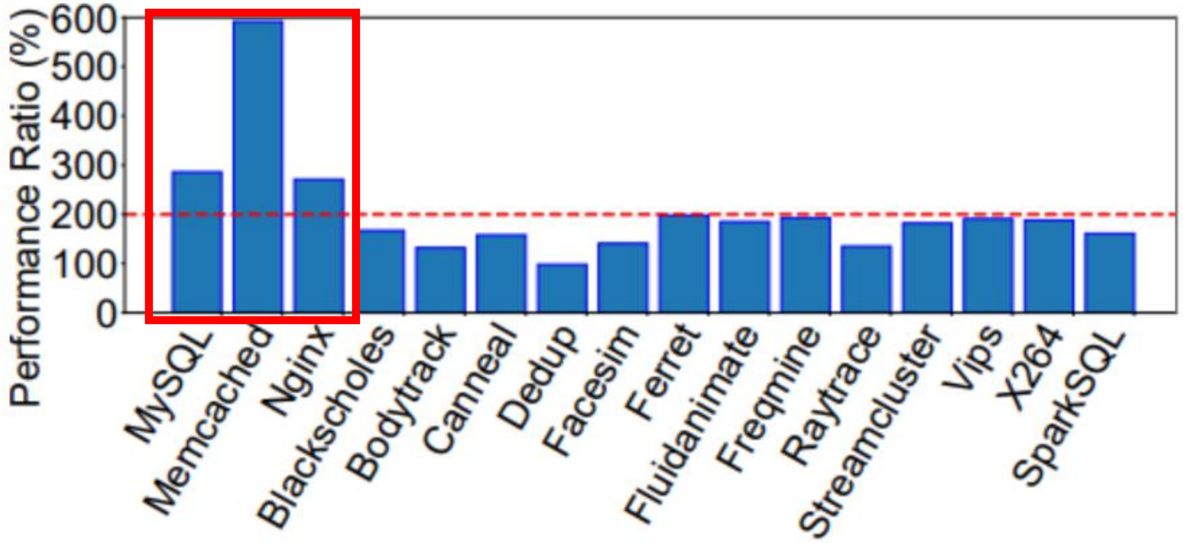
Best-effort Applications (BE)	Latency-critical Applications (LC)
• Throughput-oriented	• Tail latency
• No latency constraint	• Strict QoS constraint

① Background and Motivation

Due to double scheduling, **LC applications** suffer up to **10** times worse.



Double scheduling problem
(CPU virtualization)

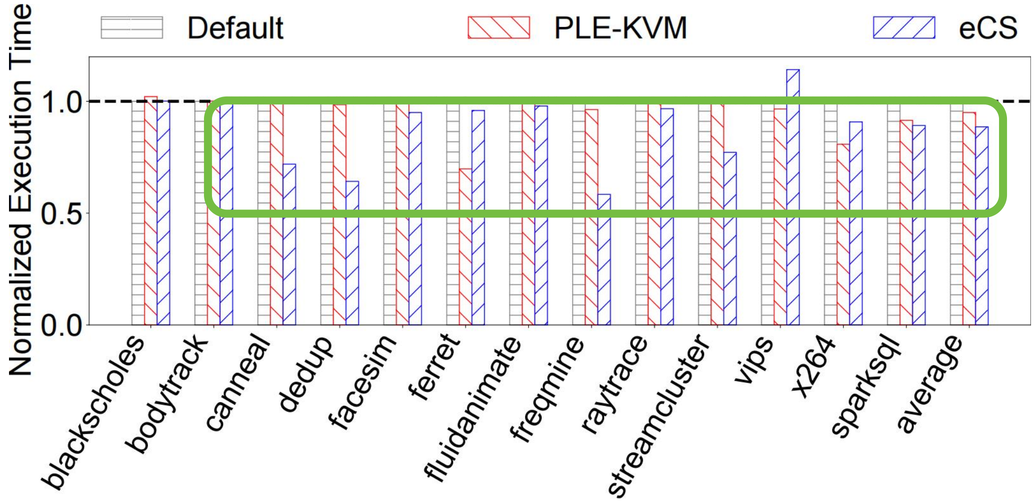


Performance comparison
(under CPU oversubscription of 2)

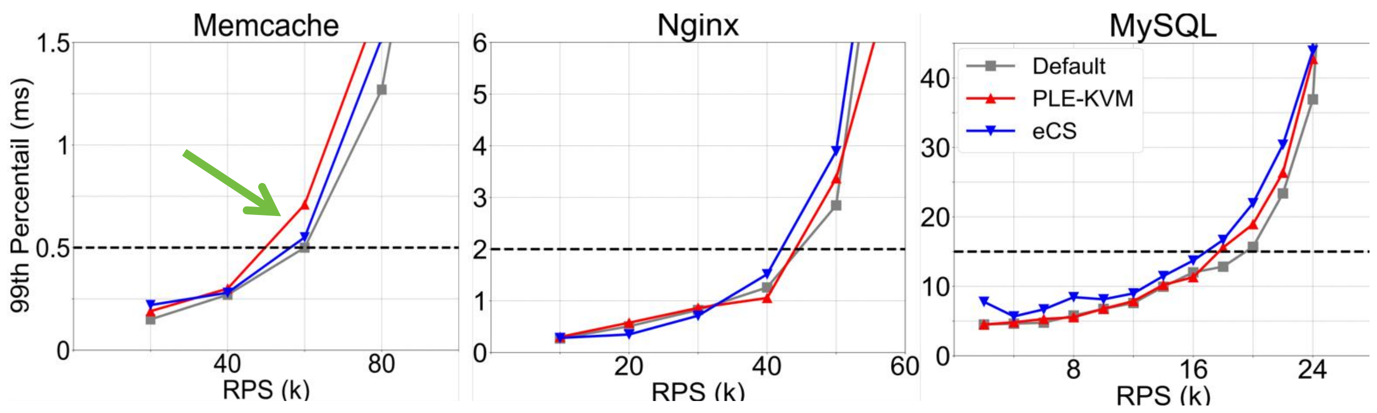
1 Background and Motivation

Previous studies for virtualization optimization focused on **BE** applications.

name	publication	year	benchmark
Revisiting VM-Agnostic ...	TPDS	2023	parsec3.0, mosbench...
PLE-KVM	VEE	2021	parsec3.0, mosbench...
Virtualization Overhead ...	TPDS	2021	PARSEC, SPLASH2X
Flexible Micro-sliced Cores ..	EuroSys	2018	gmake,swaptions,dedup...
eCS	USENIX ATC	2018	Apache,Psearchy,Pbzip2...



PLE-KVE & eCS on BE applications



PLE-KVE & eCS on LC applications

1 Background and Motivation

Previous work on LC colocation relies on **application-level inputs** to guide **QoS-aware** resource management.

Algorithm 1 ARQ Resource Scheduling Algorithm.

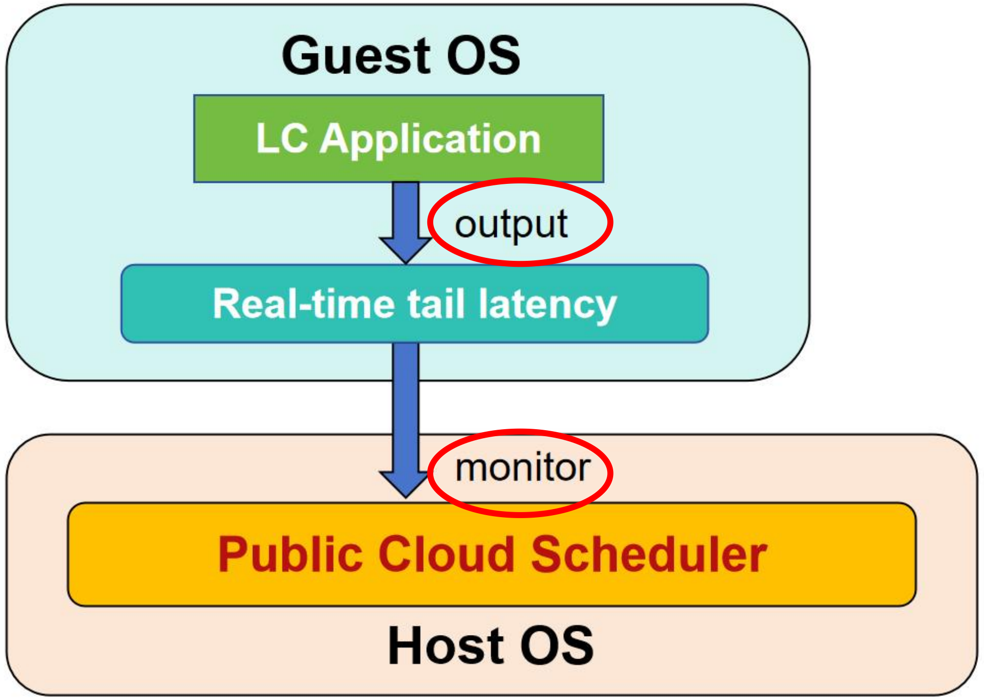
```
function ARQ
    isAdjust ←False, E_S ← 1
    while True do
        Monitor the tail latency values of the LC applications and the IPC values of BE applications periodically
        E'_S ← E_S
        E_S ← computeEntropy()
        // ReT is an array, the elements of which are the remaining tolerance of each LC application.
        ReT ← computeRemainingTolerance()
```

Ah-q (HPCA' 23)

Algorithm 1: PARTIES' main function.

```
// Start from fair allocation of all resources
initialization();
while TRUE do
    monitor tail latency and resource utilization for 500ms;
    adjust_network_bandwidth_partition();
    for each application A do
        slack[A] ← (target[A] - latency[A]) / target[A];
    end
```

PARTIES (ASPLOS' 19)



1 Background and Motivation

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES

Challenges:

Public Cloud

Virtualization | **Oversubscription** | **QoS-Aware**

1) LC performs 10x worse than BE applications due to double scheduling problem.

2) LC applications are subject to internal resource contention within a VM.

3) No application-level performance metrics inside VMs to help manage resources.

2 Characterization

Challenges：

1) Coordination between Host OS and Guest OS

LC performs 10x worse than BE applications due to double scheduling problem.

2) Coordination between vCPU Threads and Emulator Threads

LC applications are subject to internal resource contention within a VM.

3) Coordination between Host Core Manager and Guest Applications

No application level performance metrics inside VMs to help manage resources.

2 Characterization

Challenges：

1) Coordination between Host OS and Guest OS

LC performs 10x worse than BE applications due to double scheduling problem.

2) Coordination between vCPU Threads and Emulator Threads

LC applications are subject to internal resource contention within a VM.

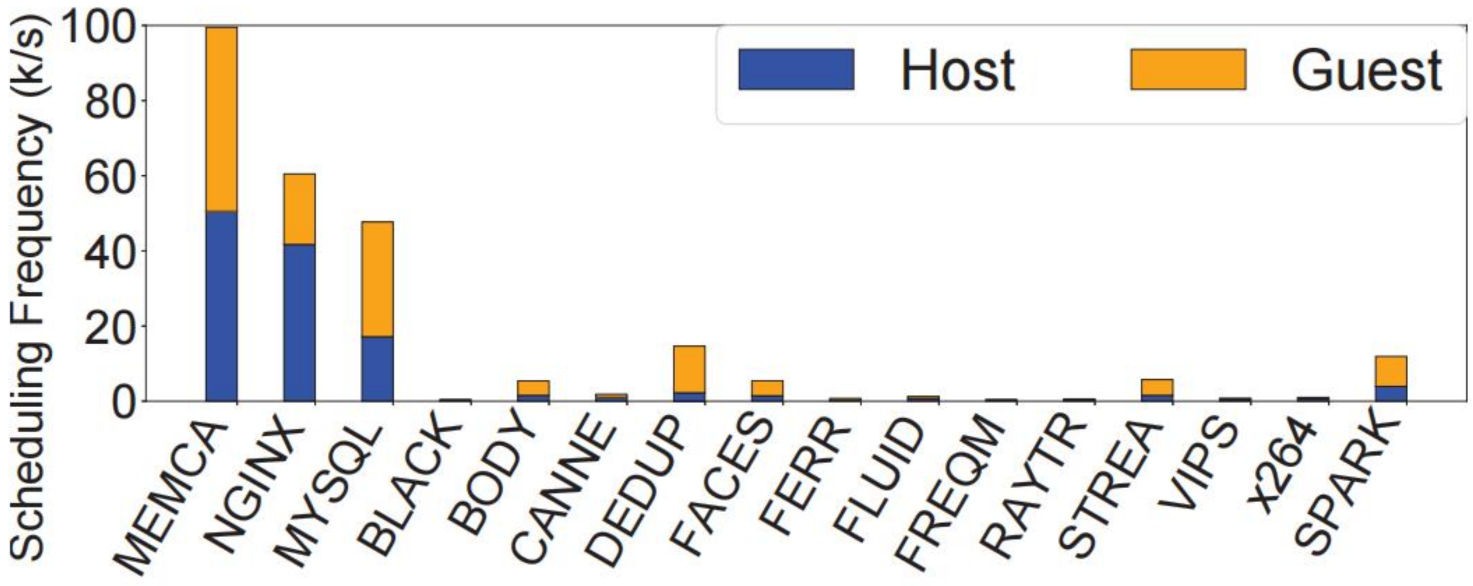
3) Coordination between Host Core Manager and Guest Applications

No application level performance metrics inside VMs to help manage resources.

LC applications: more context switch overhead

LC applications consists of **numerous sub-millisecond** tasks.

BE applications: **fewer** and **longer** tasks.



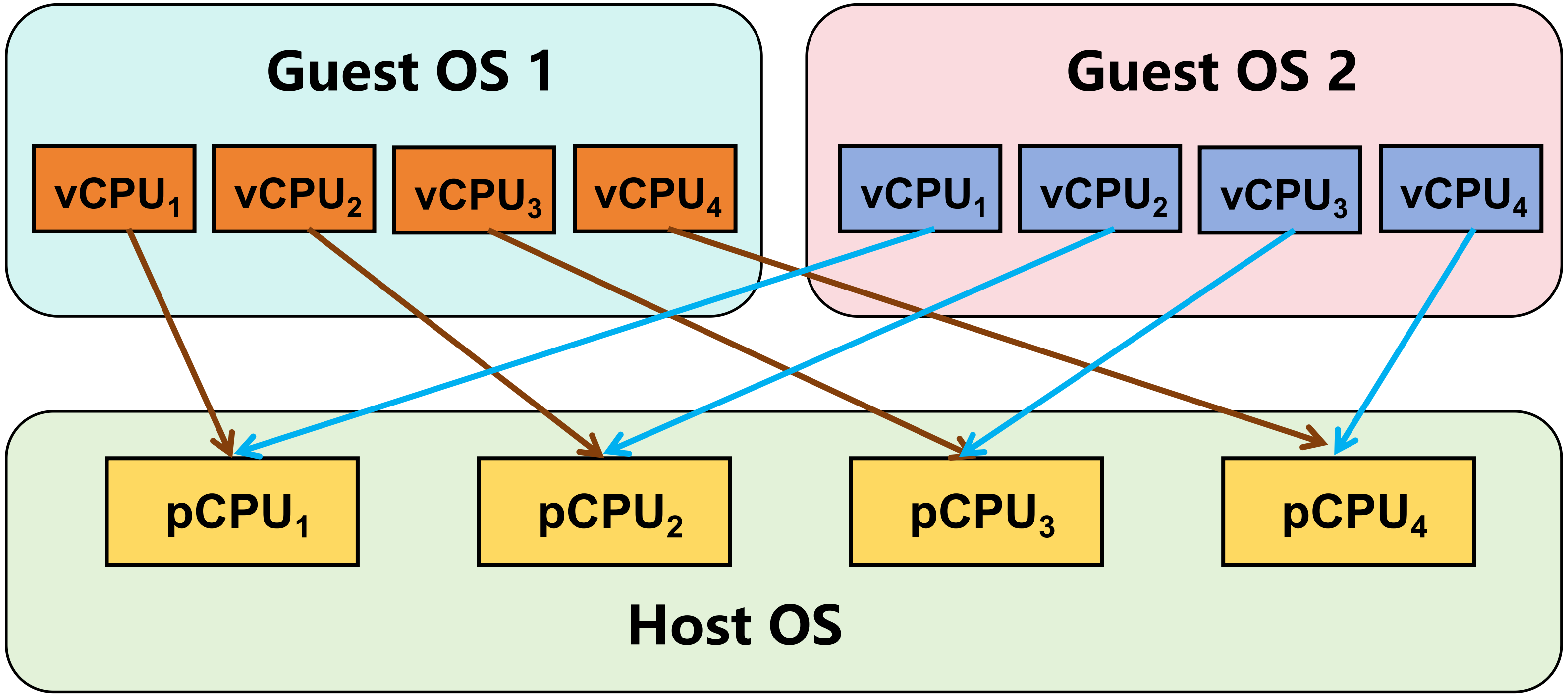
How to fix it?

② Characterization

Default: Rely on the scheduling policy of OS to schedule VMs. All the two VMs share the same 4 pCPUs.

Isolation: Isolate the two VMs, each assigned two pCPUs on the host.

Host-Aware Isolation: On top of **Isolation**, the Guest OS is aware that the VM is allocated with only two pCPUs, and schedules only two vCPUs. (Hot-plug)



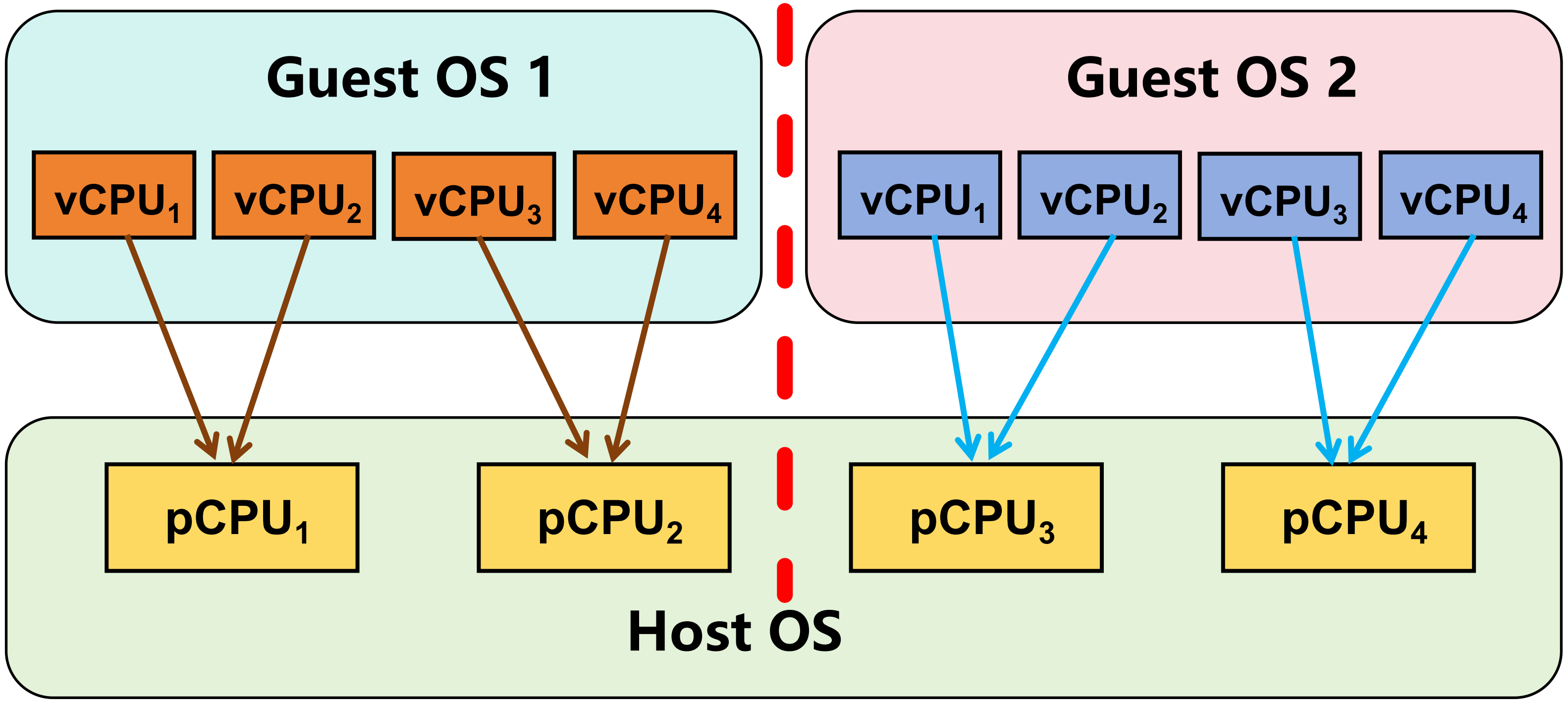
Default

② Characterization

Default: Rely on the scheduling policy of OS to schedule VMs. All the two VMs share the same 4 pCPUs.

Isolation: Isolate the two VMs, each assigned two pCPUs on the host.

Host-Aware Isolation: On top of **Isolation**, the Guest OS is aware that the VM is allocated with only two pCPUs, and schedules only two vCPUs. (Hot-plug)



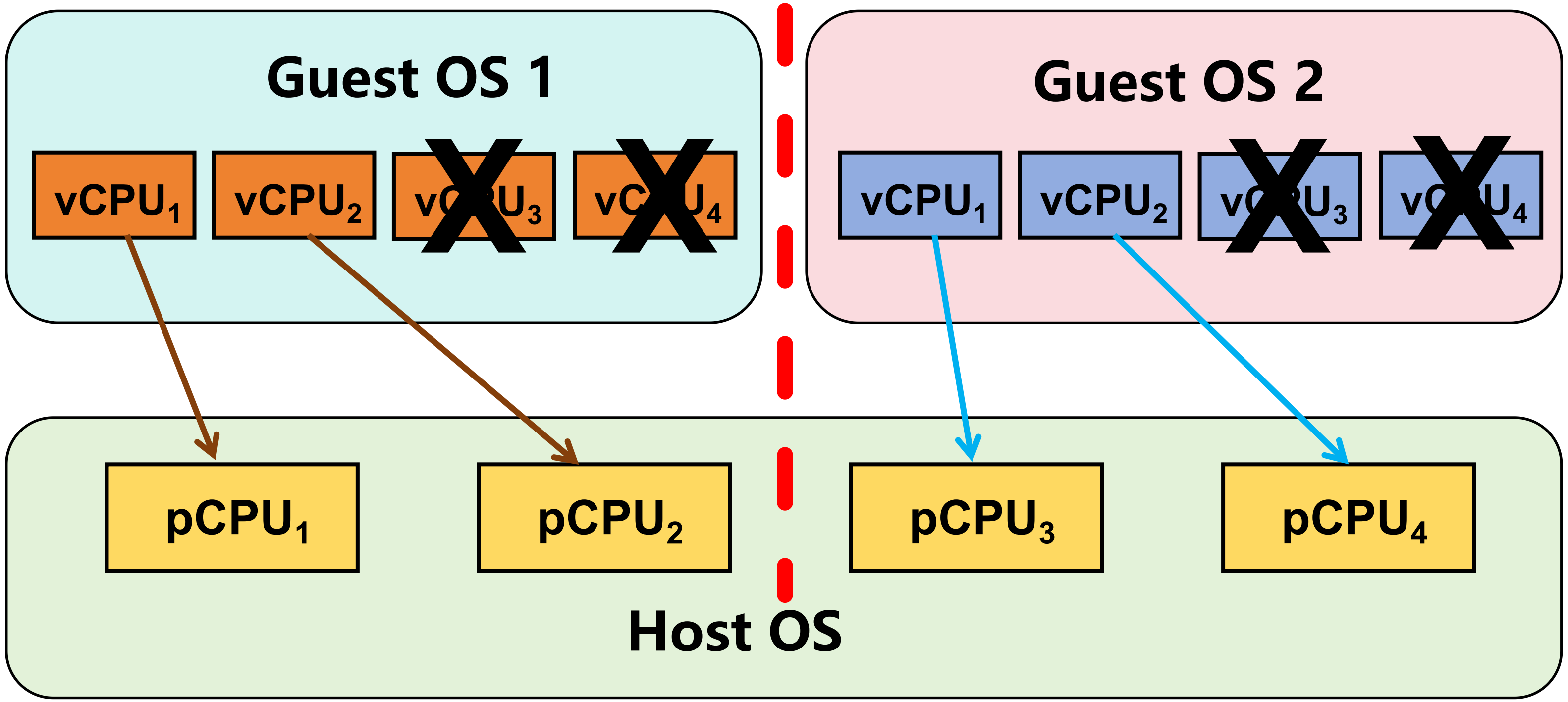
Isolation

② Characterization

Default: Rely on the scheduling policy of OS to schedule VMs. All the two VMs share the same 4 pCPUs.

Isolation: Isolate the two VMs, each assigned two pCPUs on the host.

Host-Aware Isolation: On top of **Isolation**, the Guest OS is aware that the VM is allocated with only two pCPUs, and schedules only two vCPUs. (Hot-plug)

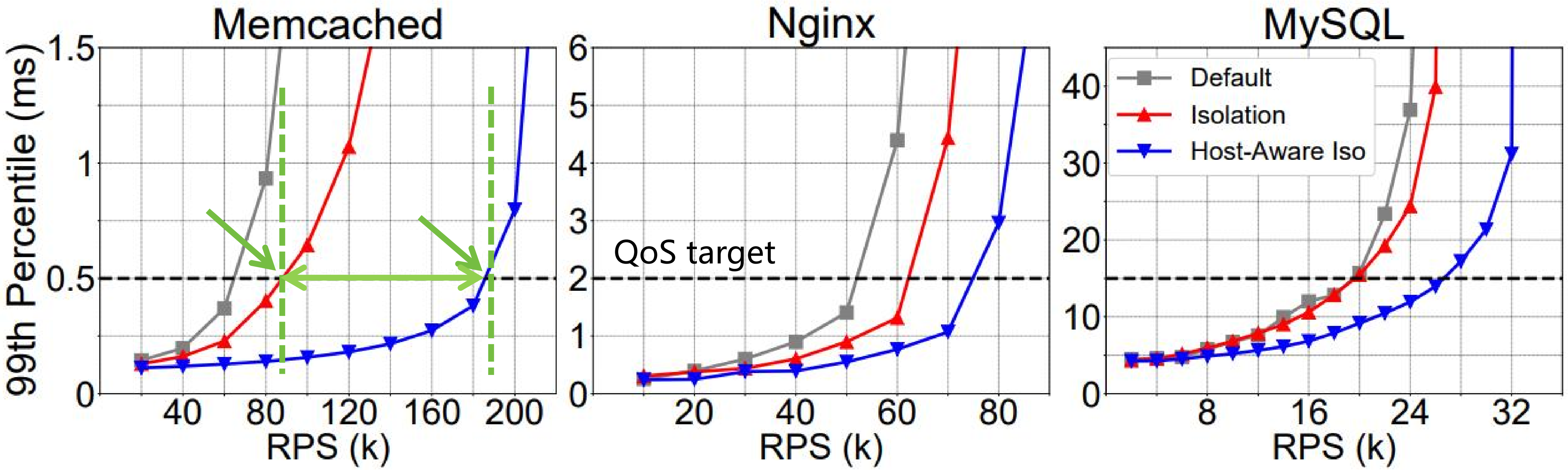


Host-Aware Isolation

② Characterization

Host Aware-Isolation: reduce overhead from double scheduling problem

Keep the number of vCPU same as pCPU ⇨ Host-Guest coordination

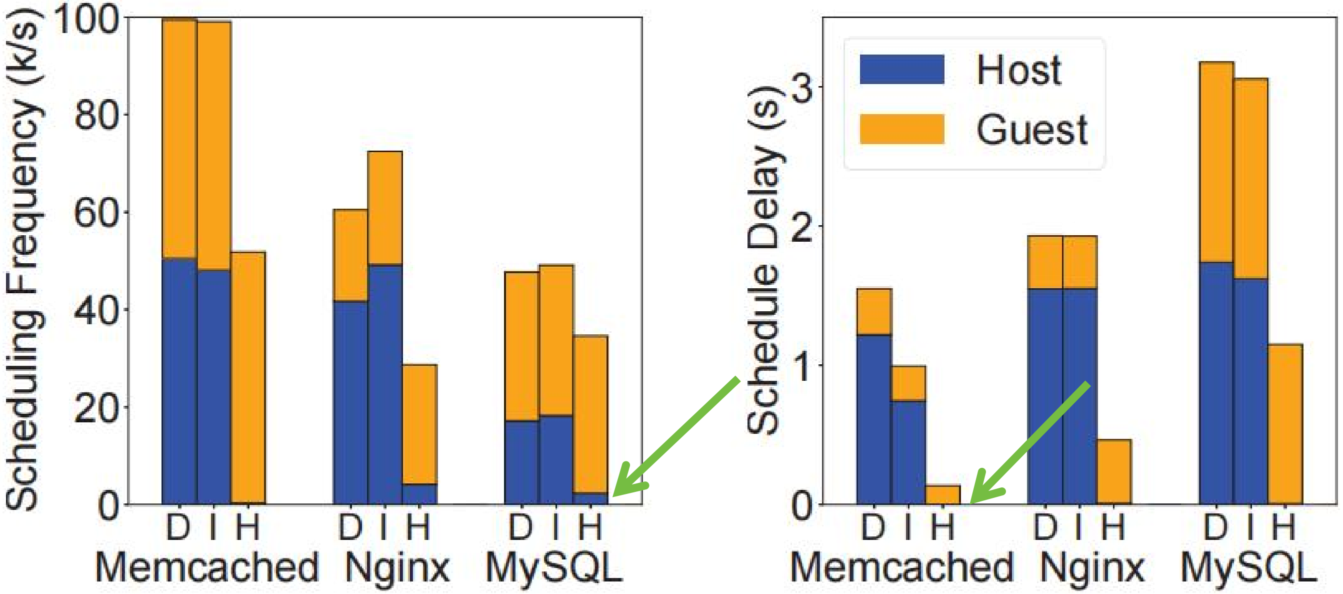


Isolation achieves up to **33%**(average 18%) higher load than **Default**, **Host-Aware Iso** further increases the maximum load under QoS by up to **25% - 125%** than **Isolation**.

② Characterization

Host Aware-Isolation: reduce overhead from double scheduling problem

Keep the number of vCPU same as pCPU ⇨ Host-Guest coordination



(a) Scheduling Frequency **(b) Scheduling Delay**

D : Default
I : Isolation
H : Host-Aware Isolation

1) Schedule Frequency, Schedule Delay,
2) VM Exits, VM Exit Handling Time, Cache Miss.

2 Characterization

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY CHINESE ACADEMY OF SCIENCES

Challenges:

1) Coordination between Host OS and Guest OS

LC performs 10x worse than BE applications due to double scheduling problem.

2) Coordination between vCPU Threads and Emulator Threads

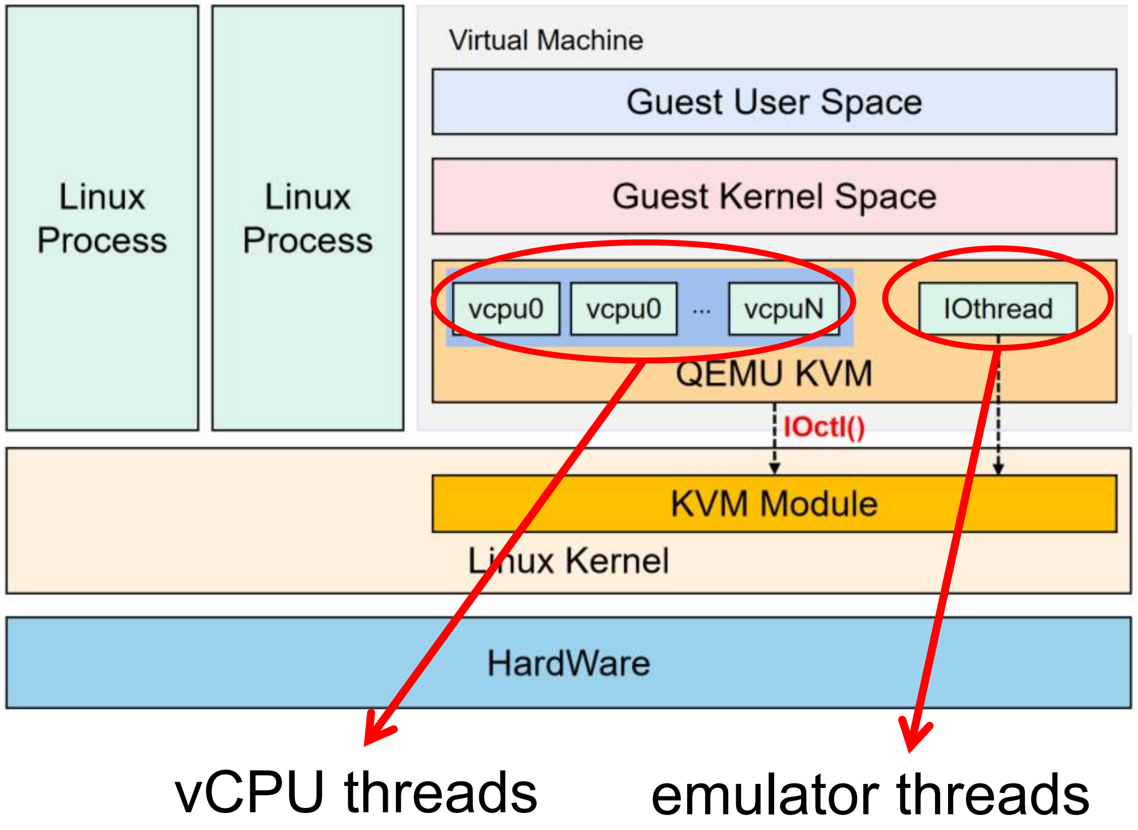
LC applications are subject to internal resource contention within a VM.

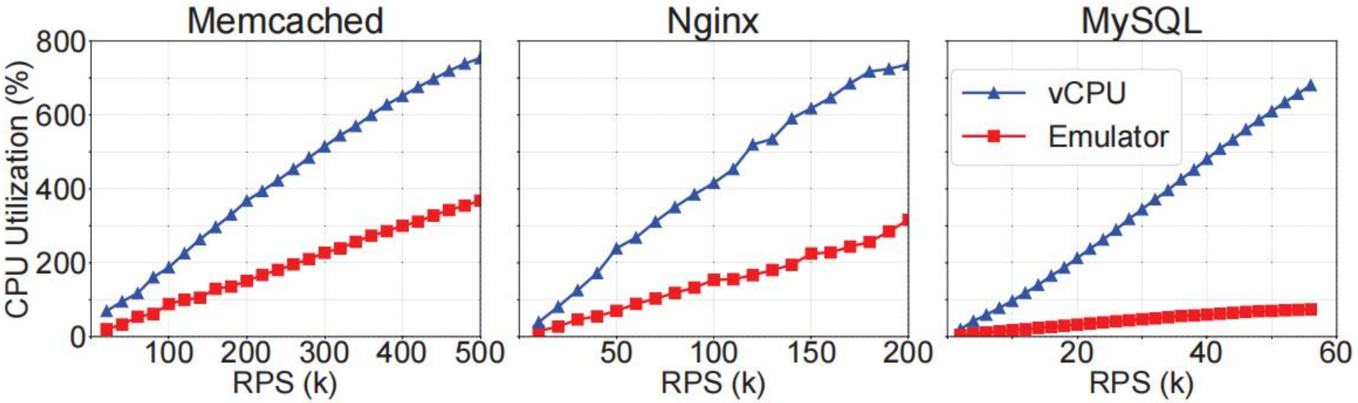
3) Coordination between Host Core Manager and Guest Applications

No application level performance metrics inside VMs to help manage resources.

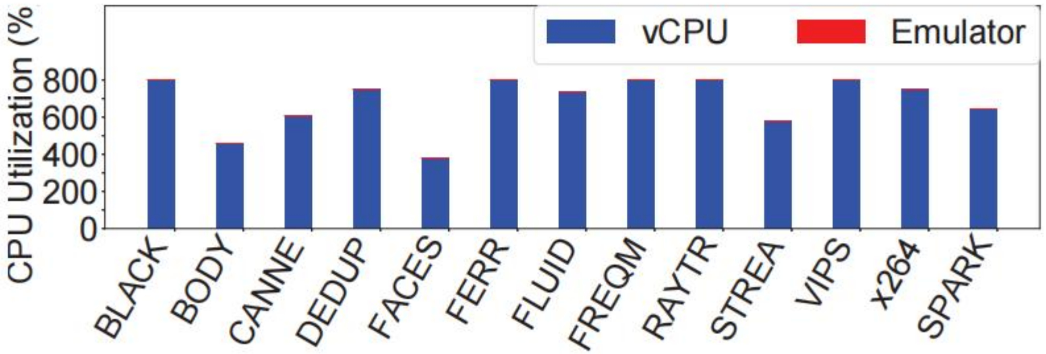
② Characterization

Emulator threads cause resource contention within a VM.





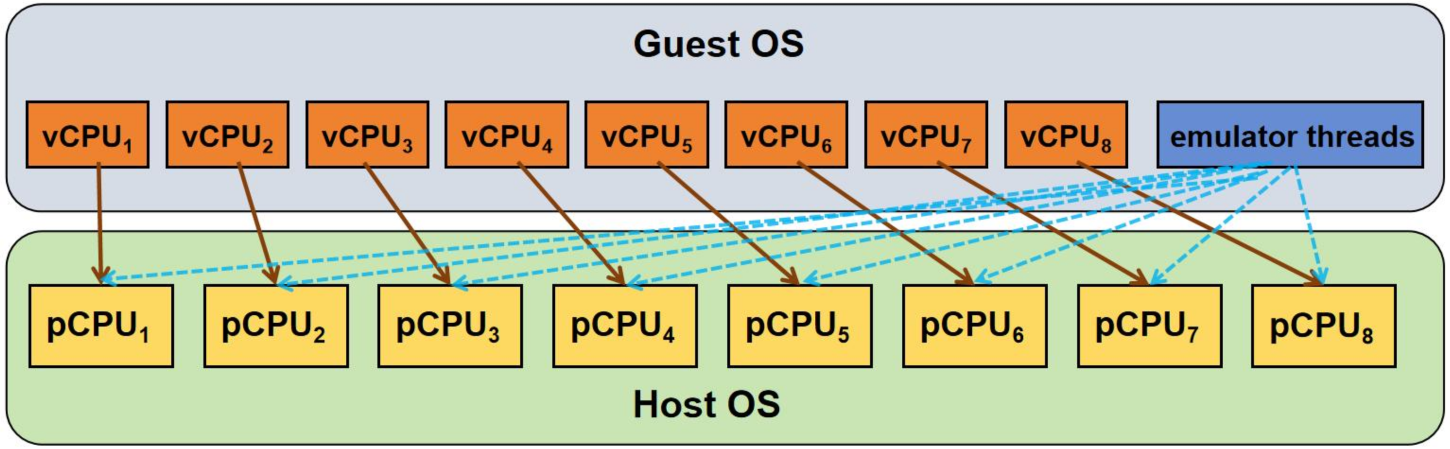
LC: active, related to input load



BE: almost have no usage

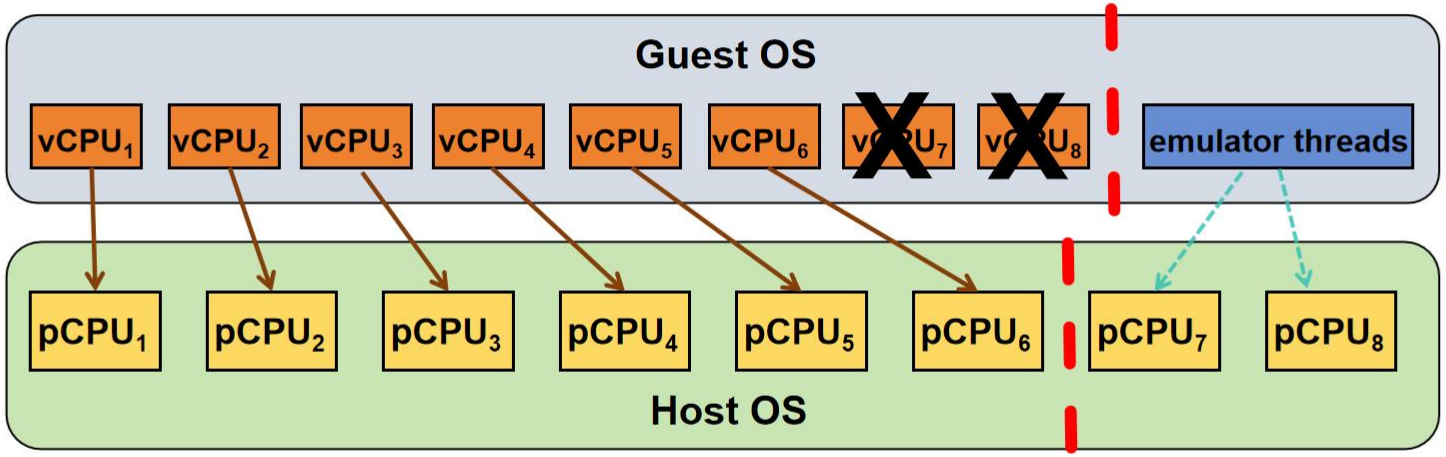
2 Characterization

vCPU thread group and emulator thread group have different core demands, and interfere with each other when sharing cores.



Shared:

vCPU threads share 8 pCPUs with emulator threads.
(Default set)



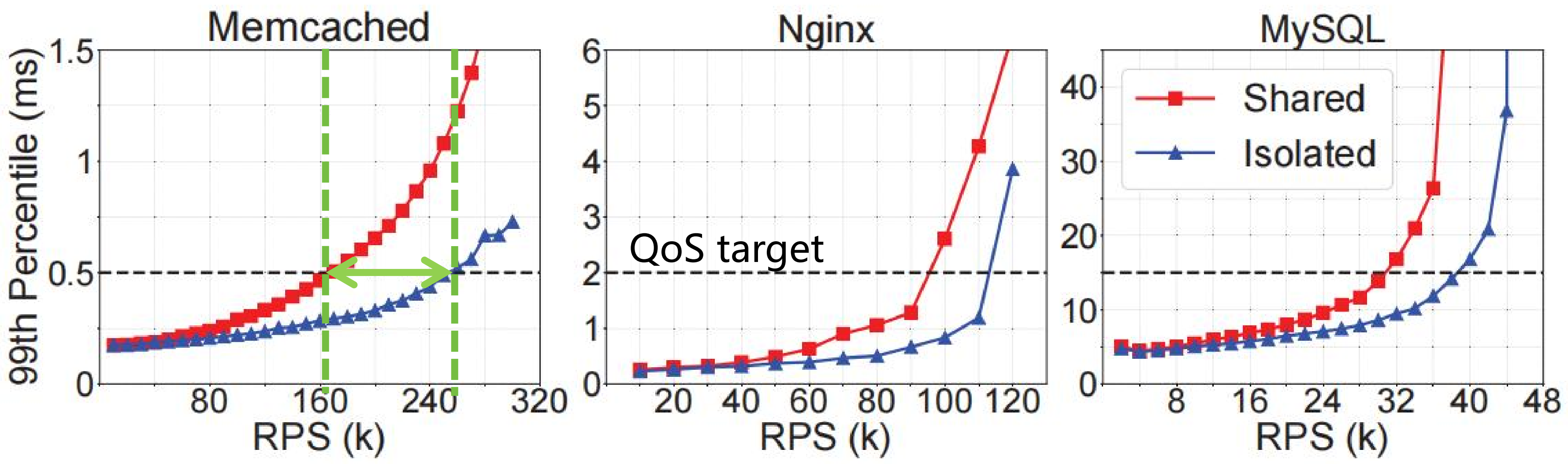
Isolated:

Partition 8 pCPUs into 6 and 2 cores, and adopt **Host-Aware Isolation** in the vCPU core group.

② Characterization

Isolation inner VM

⇨ Coordination between vCPUs Threads and Emulator Threads



Compared with **Shared**, **Isolated** achieves **15% - 50%** higher input load.

- CPU utilization is a great indicator of application's input load;
- Core allocation of both vCPU and emulator threads should be dynamically adjusted based on input load.

2 Characterization

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES

Challenges:

1) Coordination between Host OS and Guest OS

LC performs 10x worse than BE applications due to double scheduling problem.

2) Coordination between vCPU Threads and Emulator Threads

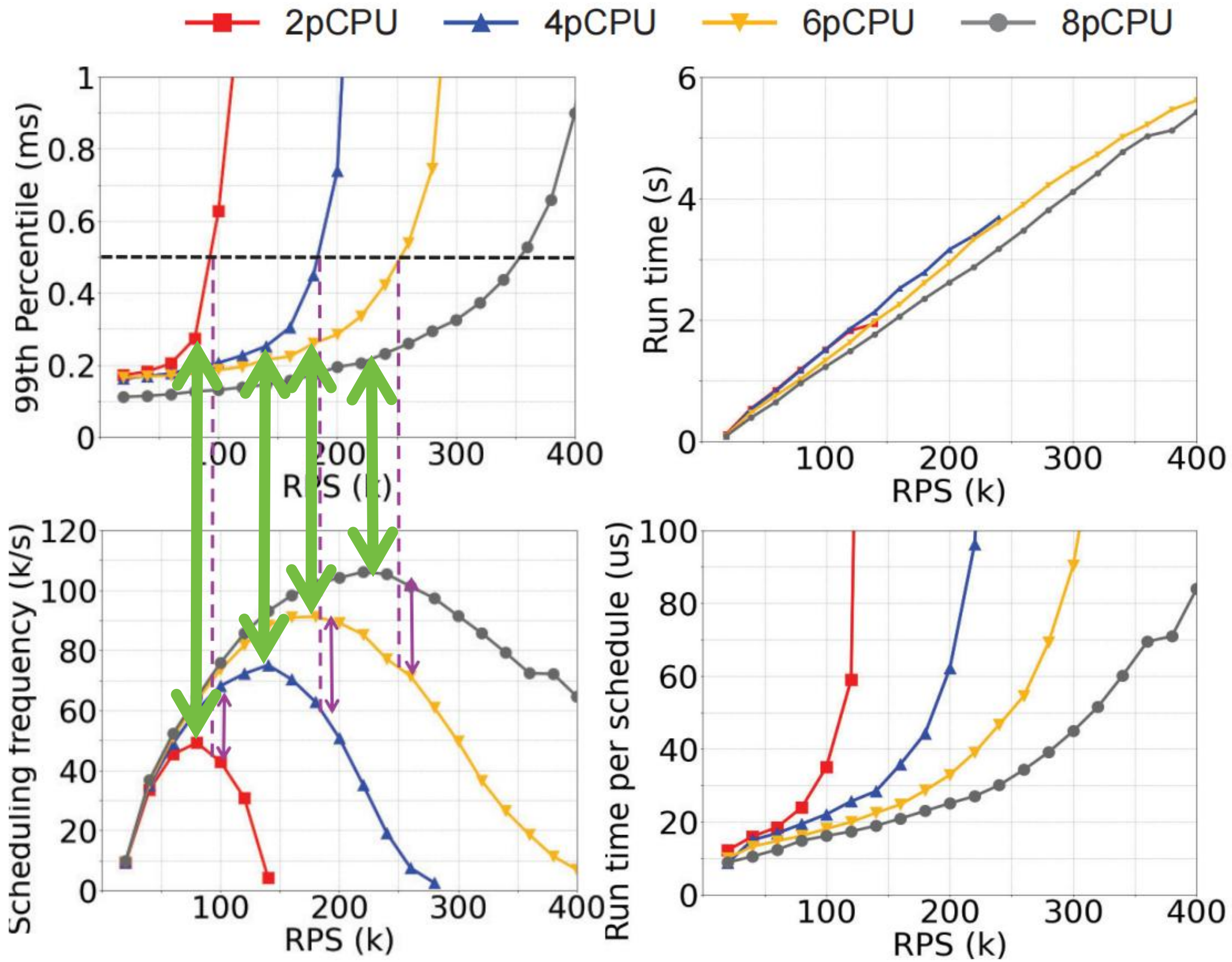
LC applications are subject to internal resource contention within a VM.

3) Coordination between Host Core Manager and Guest Applications

No application level performance metrics inside VMs to help manage resources.

② Characterization

Scheduling Frequency in Guest OS represents p99 ⇨
Coordination between Host Core Manager and Guest Applications



1) SF curve: quadratic function

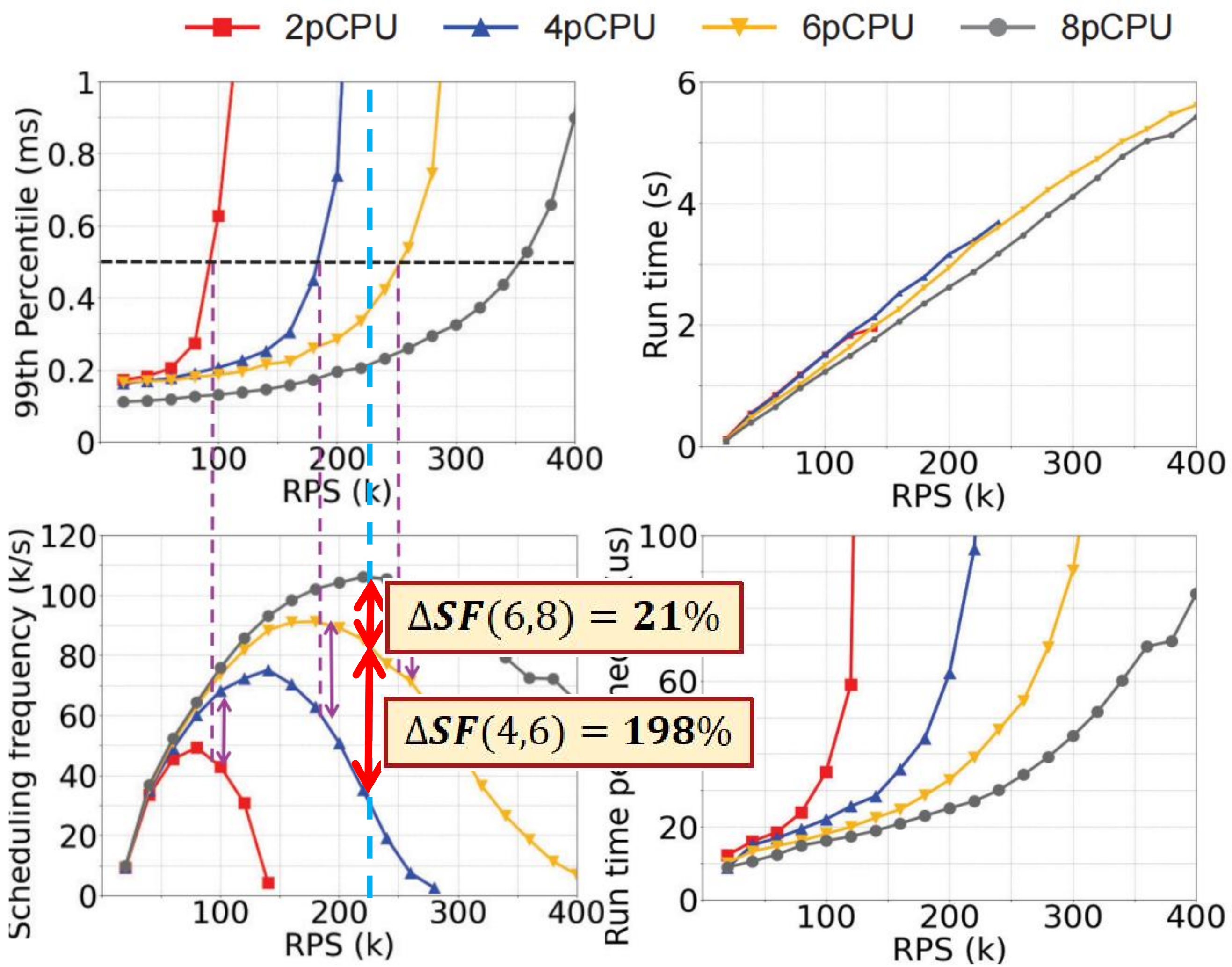
$$y = ax^2 + bx + c$$

8U: $y = -1.837x^2 + 860.2x + 4713$

2) SF curve's peak point

② Characterization

Scheduling Frequency in Guest OS represents p99 ⇨ Coordination between Host Core Manager and Guest Applications



1) SF curve: quadratic function

$$y = ax^2 + bx + c$$

2) SF curve's peak point

3) Threshold [x]

$$\Delta SF = \frac{SF[c+2] - SF[c]}{SF[c+2]} < x$$

$$\begin{cases} \Delta SF(4,6) = 198\% > x \\ \Delta SF(6,8) = 21\% < x \end{cases}$$

➡ output: 6 cpus

③ UFO Design

UFO: The Ultimate QoS-Aware CPU Core Management for Virtualized and Oversubscribed Public Clouds

- **Prioritize for LC applications**

UFO's goal is to meet QoS for LC applications through modeling of SF in Guest OS.

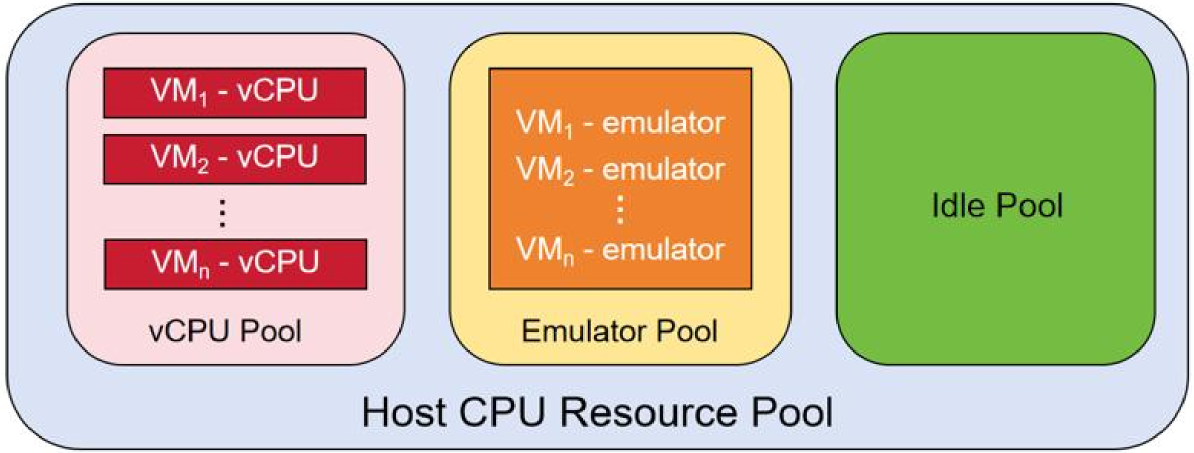
- **Optimize for virtualized and oversubscribed public clouds**

Fix double scheduling through Guest-Host coordination and vCPU-emulator isolation.

- **Focus on core management**

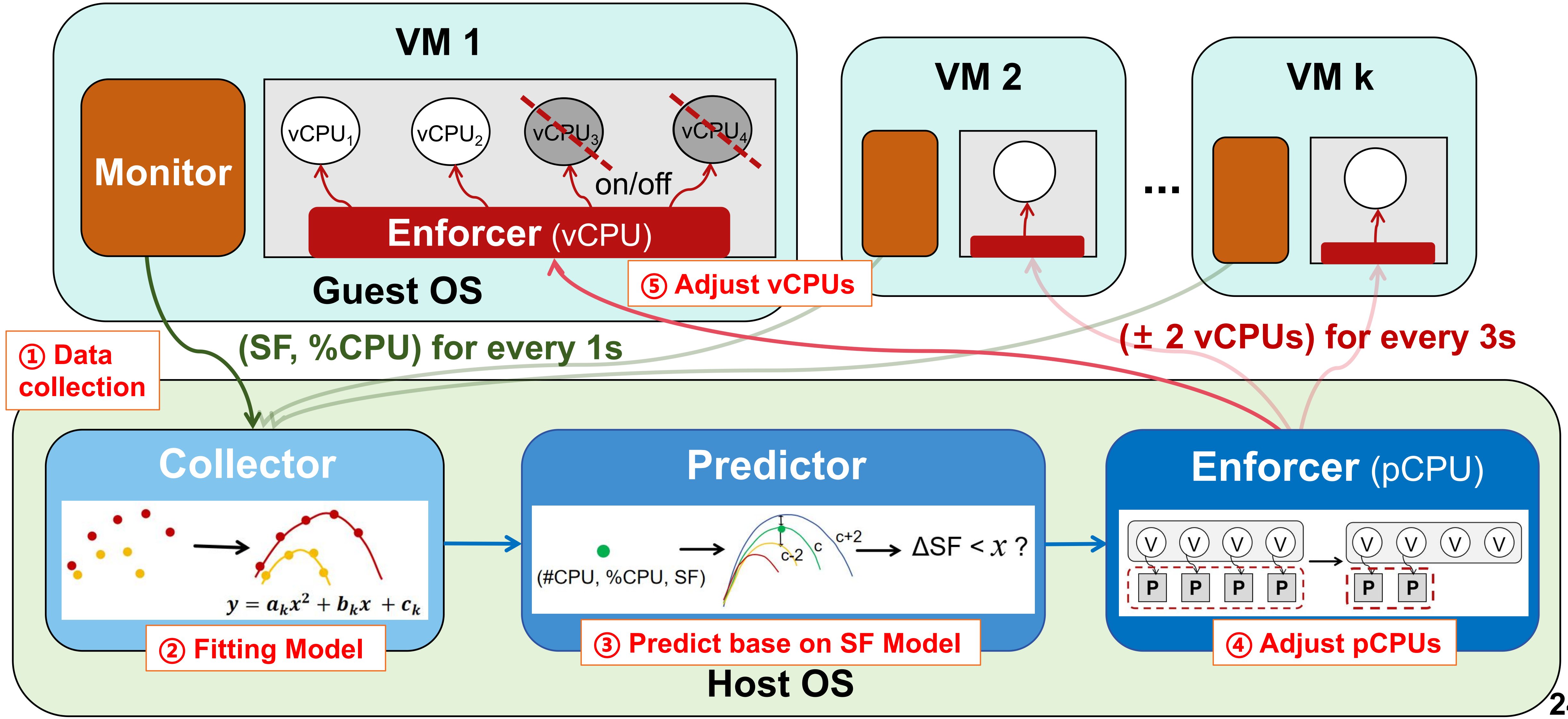
Higher performance with fewer resources.

- **Accommodate more VMs under QoS on a single host.**



③ UFO Design

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES



4 Evaluation

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES

Table 2: Latency-critical applications.

Application	**Memcached**	**Nginx**	**MySQL**
Domain	Key-value store	Web server	Database
QoS Target	0.5ms	2ms	15ms
Max Load under QoS	350k	120k	50k
Load Generator	Mutated	wrk2	sysbench
Dataset	One million <key,value> pairs	10,000 html files of 4KB each	20 tables, each with one million entries
Request Type	100% GET requests	Get file content	OLTP transactions, each with 18 select and 2 update queries

VM Size: 8 vCPU, 16 GB memory

Hyperthreading: Enabled

Baselines: Default and DynIso

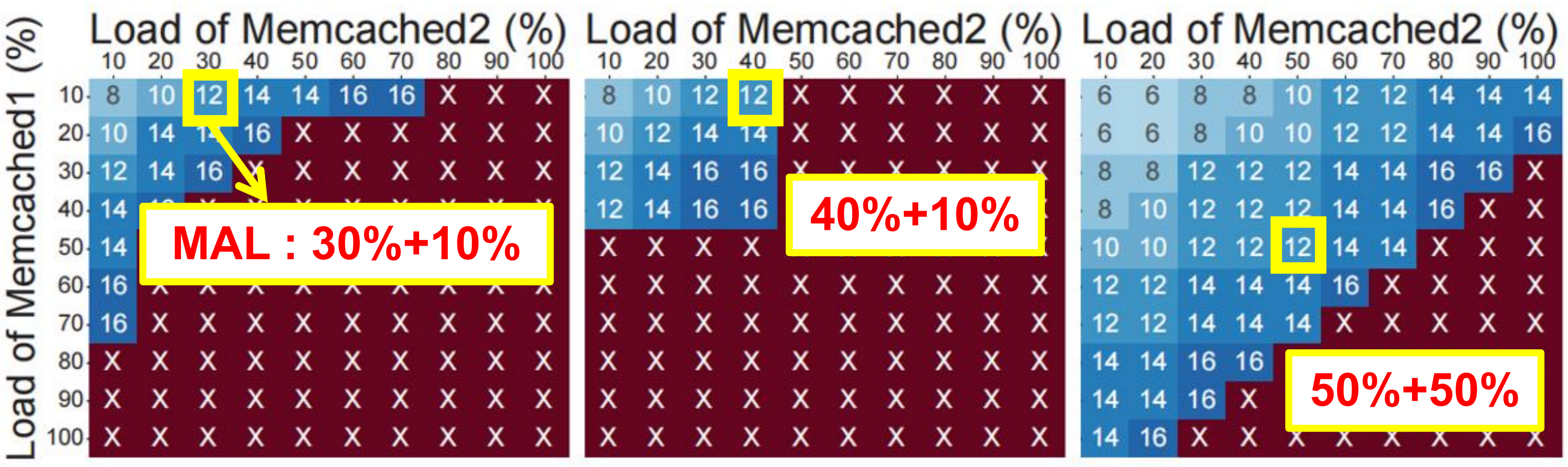
④ Evaluation: Resource Efficiency

Constant Load: Colocation of 2 VMs, evaluate resource efficiency.

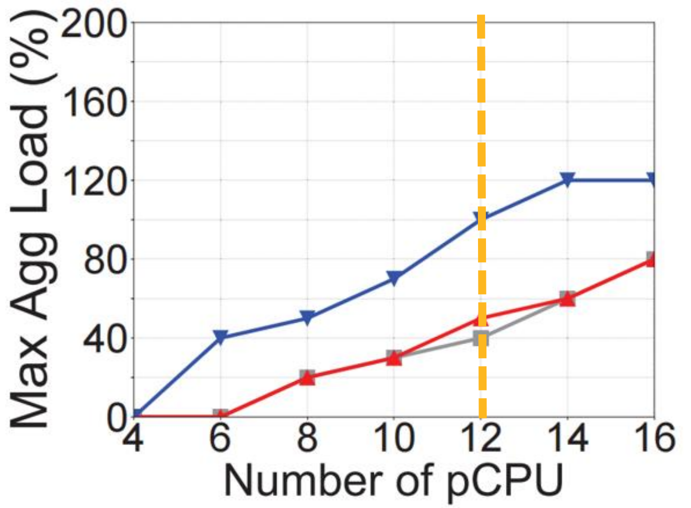
(a) Default : Default Linux set, VMs share all the pCPU cores

(b) DynIso : Dynamic isolation between VMs based on VM's input load [1]

(c) UFO : QoS-Aware CPU core management [2]



(a) Default (b) DynIso (c) UFO



Memcached + Memcached

Max Agg Load： maximum aggregated load (MAL)

[1] PARTIES (Asplos'19) [2] UFO (NSDI'24)

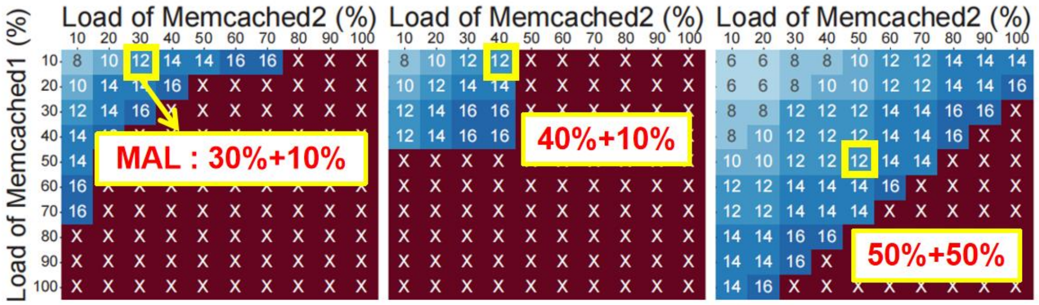
4 Evaluation: Resource Efficiency

Constant Load: Colocation of 2 VMs, evaluate resource efficiency.

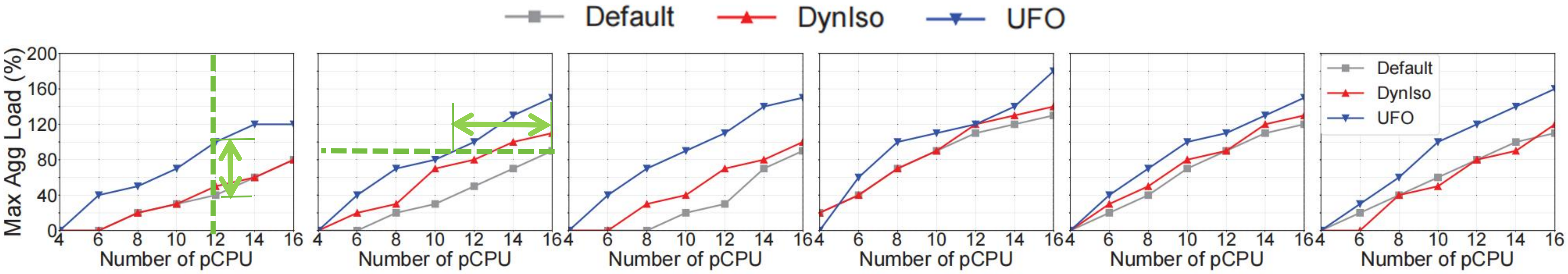
Compared with DynIso, UFO：

achieves up to 60% higher MAL under same #pCPUs.

saves up to 50% cores under the same input load.



(a) Default (b) DynIso (c) UFO



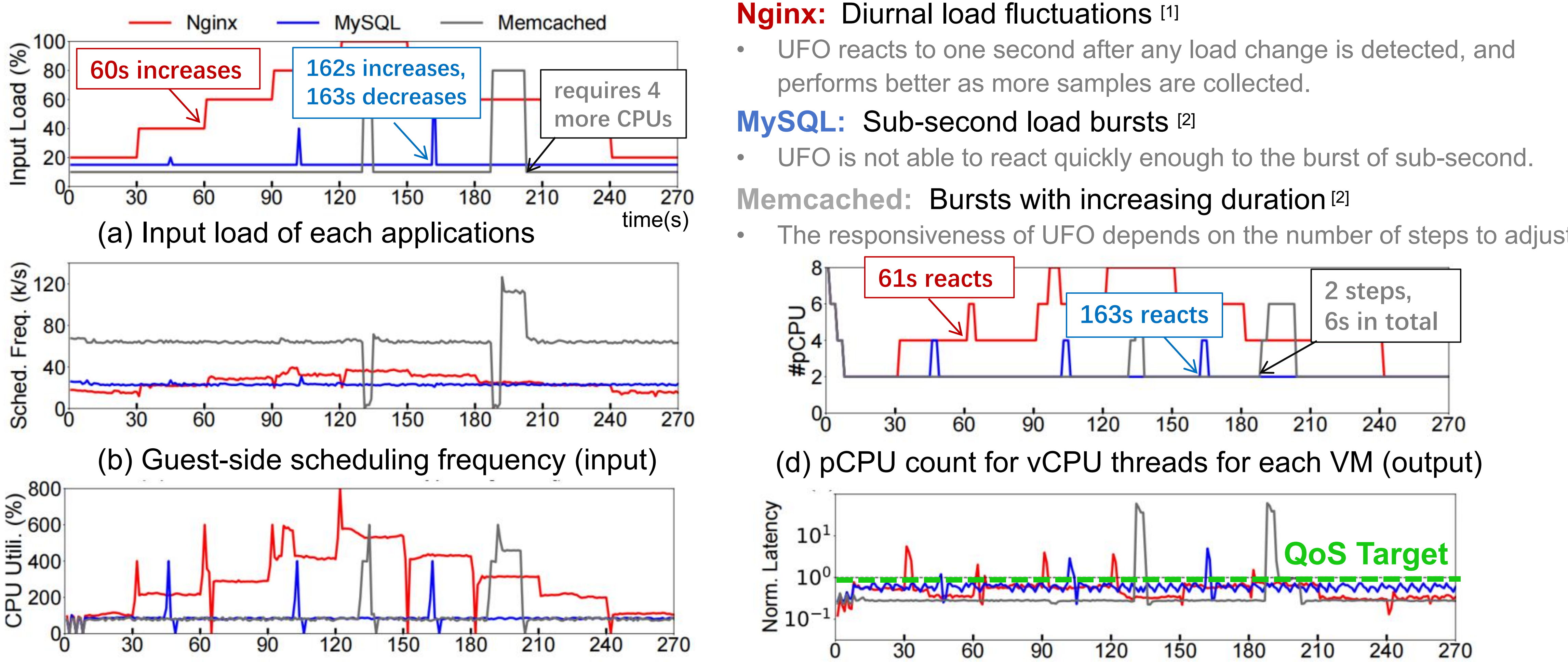
(a) Mem$+Mem$ (b) Mem$+Nginx (c) Mem$+MySQL (d) Nginx+Nginx (e) Nginx+MySQL (f) MySQL+MySQL

Max Agg Load： maximum aggregated load (MAL)

④ Evaluation: Reactions on Dynamic Load

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES

Dynamic Load: Colocation of 3 VMs, evaluate fluctuating load.



(a) Input load of each applications

(b) Guest-side scheduling frequency (input)

(c) Guest-side CPU utilization (input)

(d) pCPU count for vCPU threads for each VM (output)

(e) Tail latency normalized to QoS target under UFO (verify)

Nginx: Diurnal load fluctuations [1]

- UFO reacts to one second after any load change is detected, and performs better as more samples are collected.

MySQL: Sub-second load bursts [2]

- UFO is not able to react quickly enough to the burst of sub-second.

Memcached: Bursts with increasing duration [2]

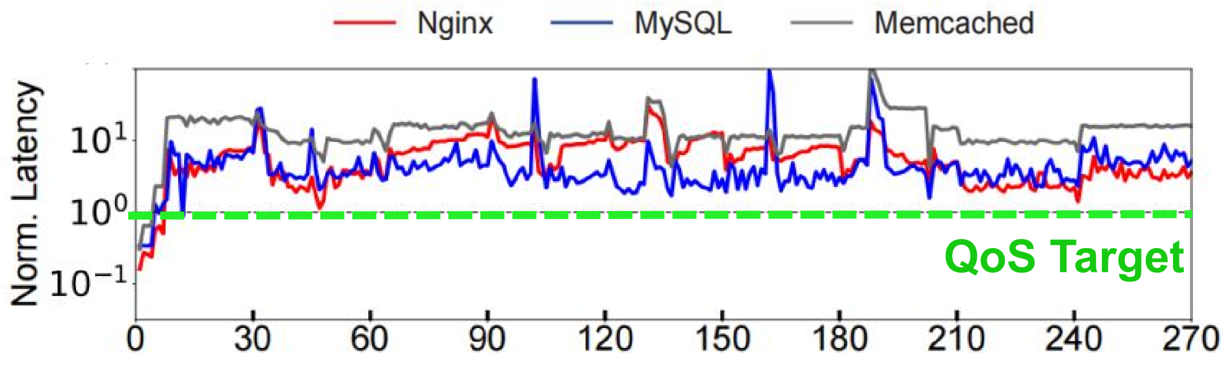
- The responsiveness of UFO depends on the number of steps to adjust.

[1] Applied machine learning at Facebook (HPCA' 18) [2] Shenango (NSDI' 19)

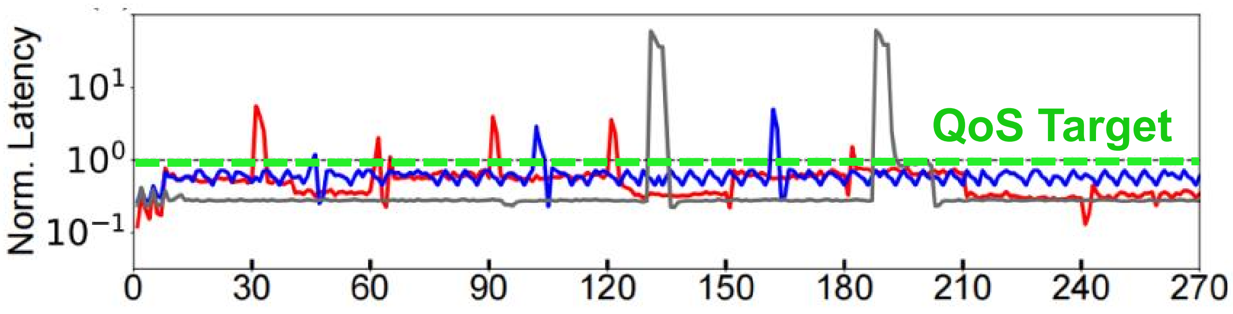
④ Evaluation: Reactions on Dynamic Load

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES

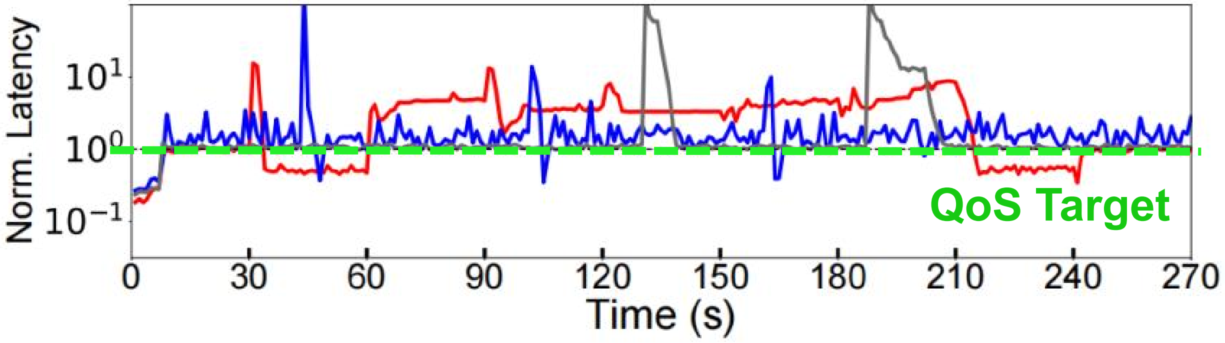
Dynamic Load: Colocation of 3 VMs, evaluate fluctuating load.



Tail latency normalized to QoS target under Default



Tail latency normalized to QoS target under UFO



Tail latency normalized to QoS target under DynIso

Nginx: Diurnal load fluctuations [1]

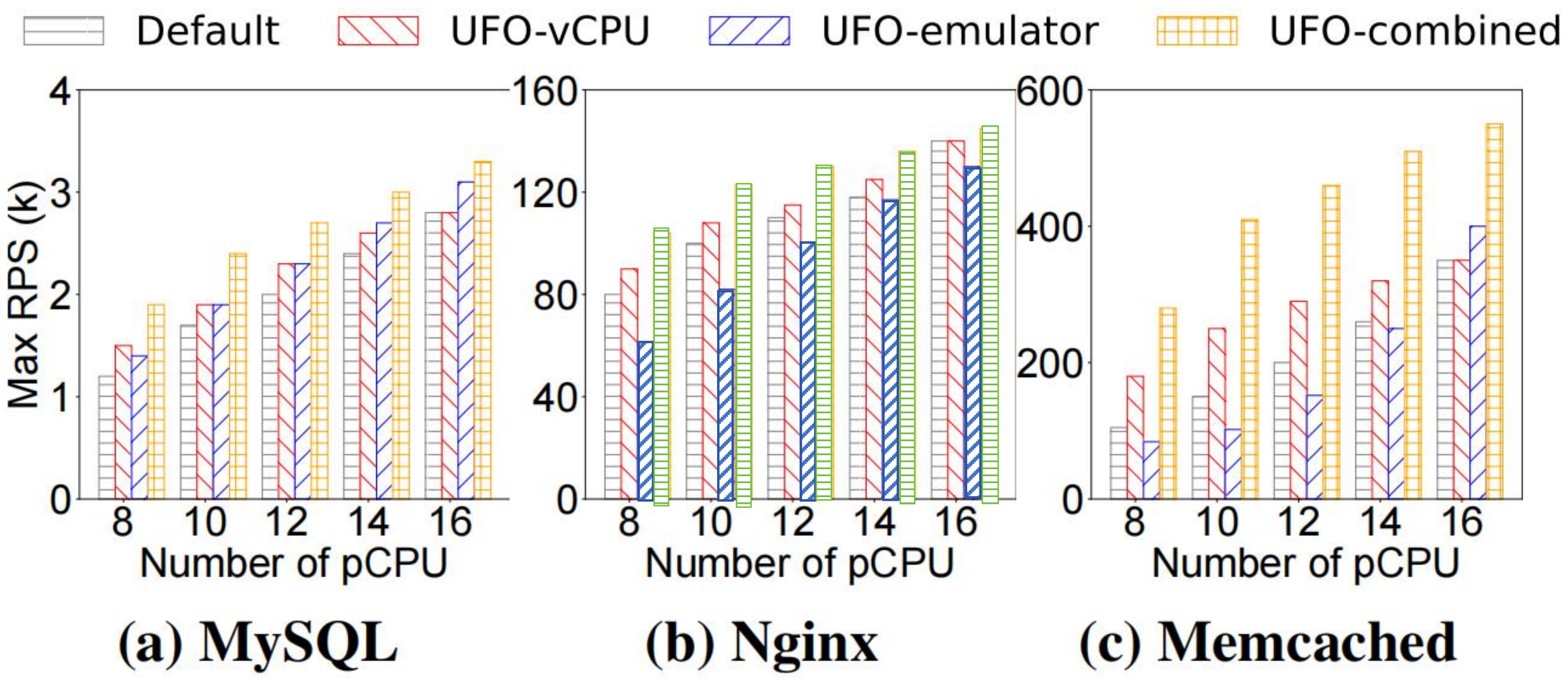
MySQL: Sub-second load bursts [2]

Memcached: Bursts with increasing duration [2]

[1] Applied machine learning at Facebook (HPCA' 18) [2] Shenango (NSDI' 19)

4 Evaluation: Decomposition of UFO

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES



UFO-vCPU achieves 19.8% higher load on average under QoS than Default.

UFO-combined: 69.8% higher load

UFO-emulator: cause vcpu staking

5 Conclusion

中国科学院深圳先进技术研究院 SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY CHINESE ACADEMY OF SCIENCES

UFO: The Ultimate QoS-Aware CPU Core Management for Virtualized and Oversubscribed Public Clouds

Yajuan Peng*, Shuang Chen*, Yi Zhao, and Zhibin Yu. （**USENIX NSDI 2024**）

- Three levels CPU coordination
  - Host OS & Guest OS
  - Inner VM: vCPU threads & emulator threads
  - Host scheduler & Guest Applications
- Dynamic management based on QoS
- Higher resource efficiency

Save up to 50% (average of 22%) cores under the same colocation scenario

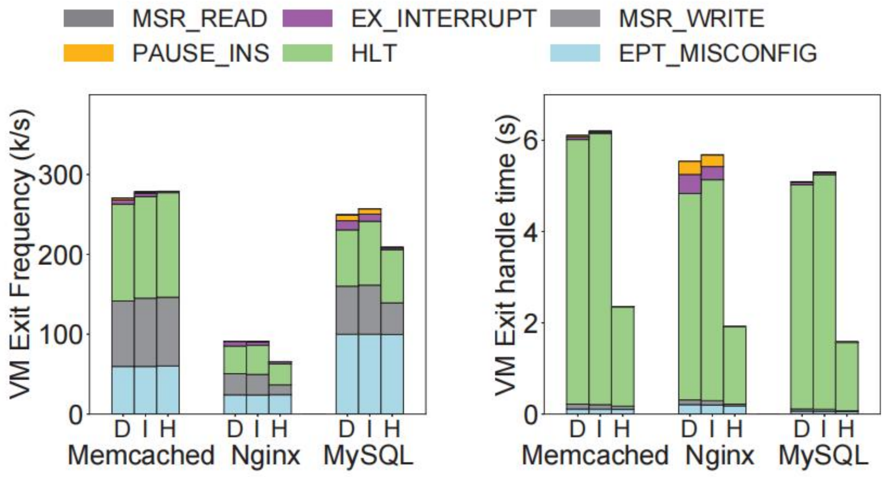
Thank You

彭雅娟 yj.peng2@siat.ac.cn

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES

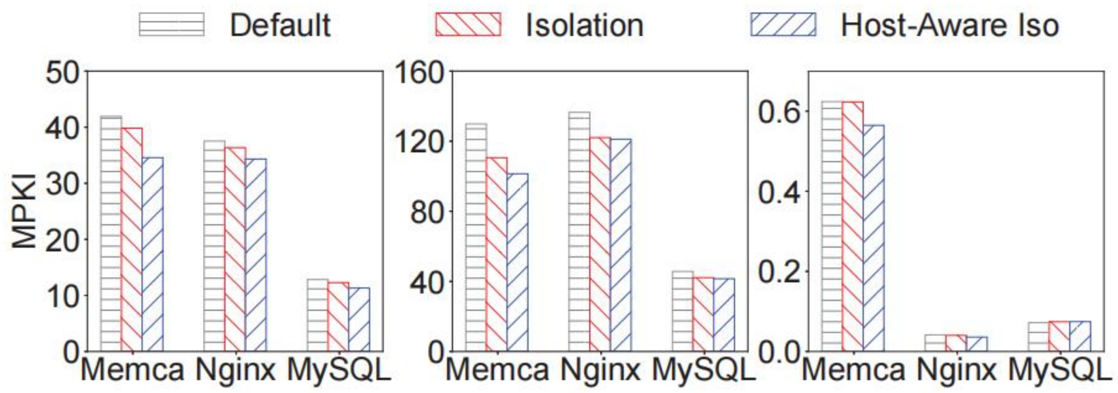
Appendix A

Impact on VM Exits and Caches under Host-Aware Iso



(a) #VM Exits (b) VM Exit Handled Time

Figure 16: VM exit frequency and VM exit handled time under default (D), isolation (I), and host-aware isolation (H), decomposed by VM exit reason.



(a) L1 dcache (b) L1 icache (c) Last-level cache

Figure 17: Cache misses-per-kilo-instructions (MPKI) under three core managers.

- VM exits are handled 2x faster on the host under host-aware isolation.
- Compared with Default, Isolation reduces L1D and L1I MPKI by up to 5% and 15% (average of 4.1% and 11%), respectively.

Appendix B

Comparison with related work

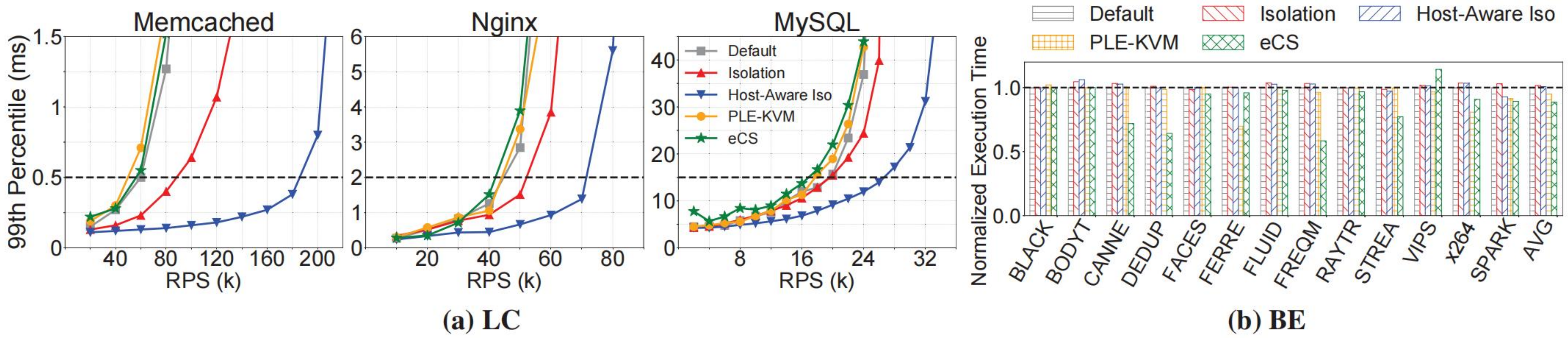


Figure 2: Performance under five core allocation mechanisms. For LC applications, we show the 99^{th} percentile tail latency with increasing input load (RPS). Horizontal dotted lines represent applications' QoS targets. For BE applications, we show the execution time of each benchmark normalized to that under the *Default* manager. Lower is better.

[1] PLE-KVM: Mitigating excessive vcpu spinning in vm-agnostic kvm. (VEE'21)

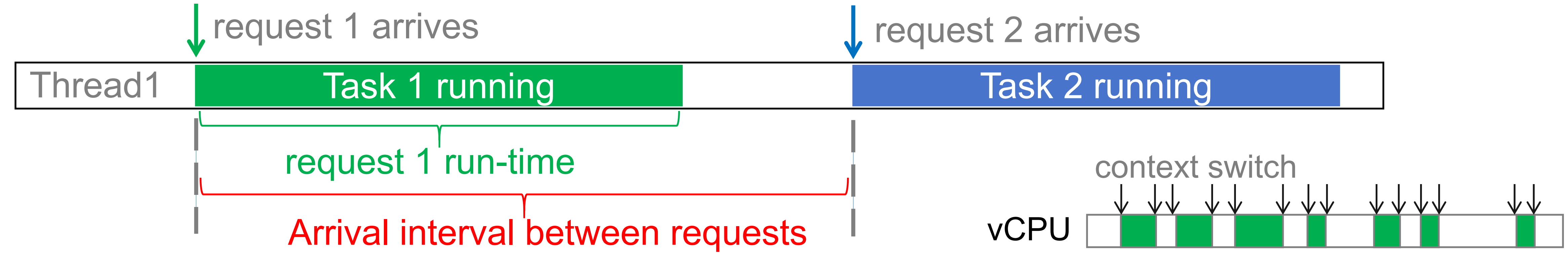
[2] eCS: Scaling guest {OS} critical sections with ecs. (USENIX ATC'18)

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES

Appendix C

High input load cause scheduling frequency decrease

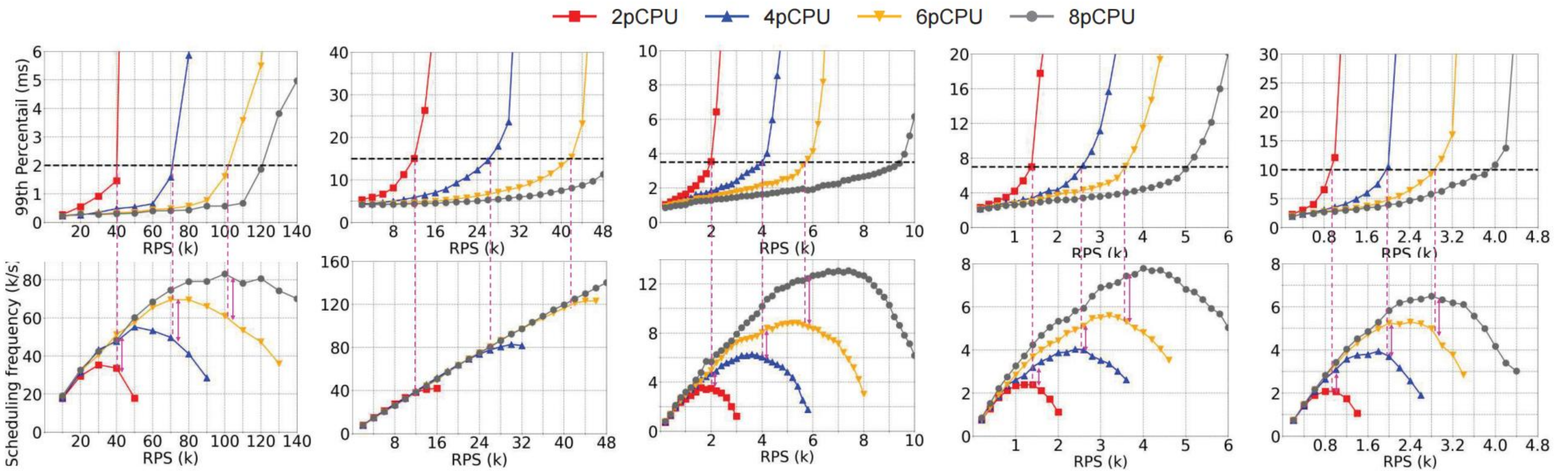
Low input load： request inter-arrival time > request processing time



High input load： request inter-arrival time ≤ request processing time

Appendix D

Indications of Guest-side Scheduling Frequency



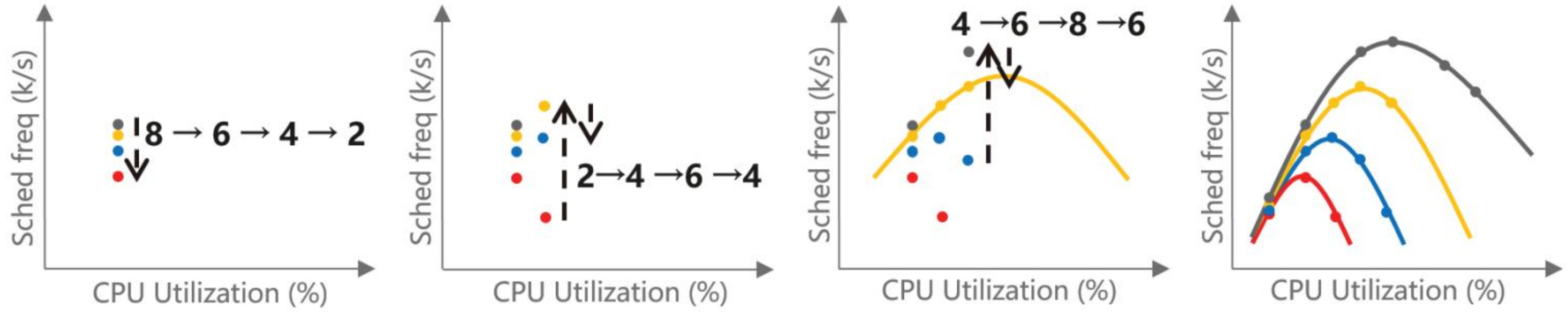
(a) Nginx (b) MySQL (c) ImgDNN (d) Xapian (e) Moses

$$\Delta SF = \frac{SF[c+2] - SF[c]}{SF[c+2]} < x$$

- For MySQL, x = 5%.
- For other 5 LC applications, x = 30%.

Appendix E

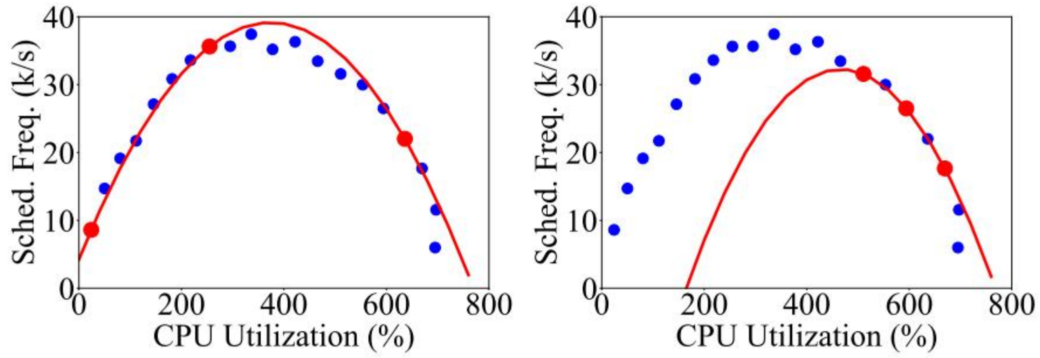
UFO: Four stages in the core predictor



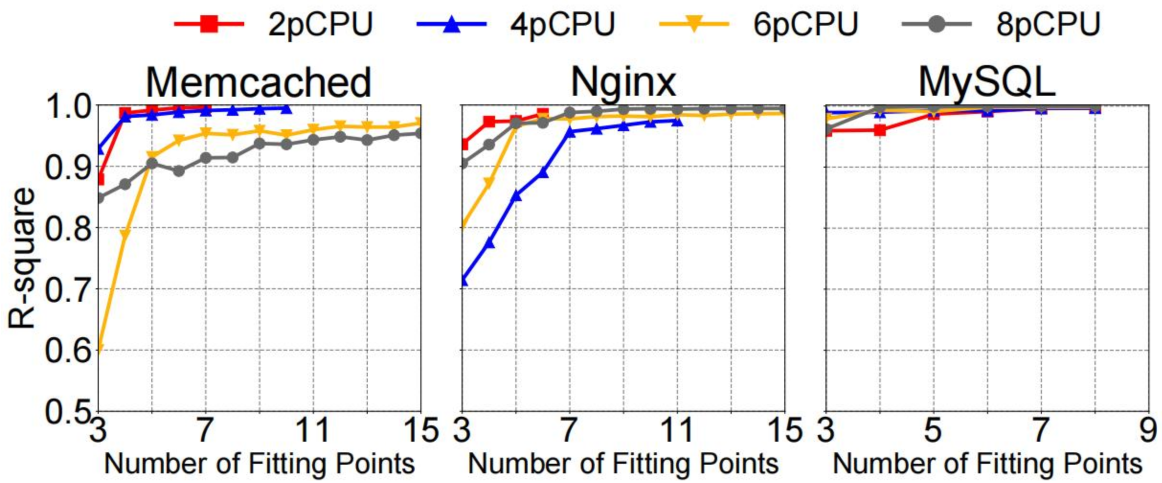
(a) Initial stage. (b) Collection stage. (c) Fitting stage. (d) Full profile.

For a #pCPU, UFO starts fitting the model when samples > 3.

Model accuracy > 95% when samples > 7.



(a) Good samples: $R^2 = 0.962$ (b) Bad samples: $R^2 = -0.029$



Appendix F

Heatmaps of 2-VM Colocation Mixes

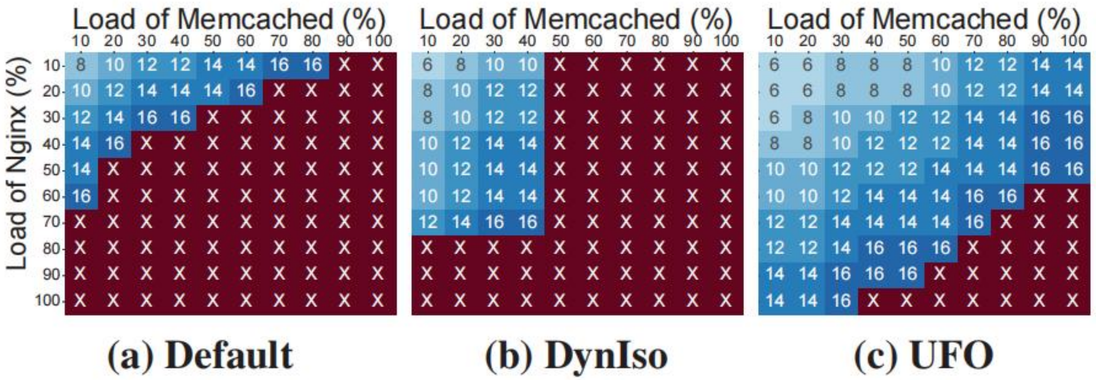


Figure 21: Colocation of Memcached and Nginx.

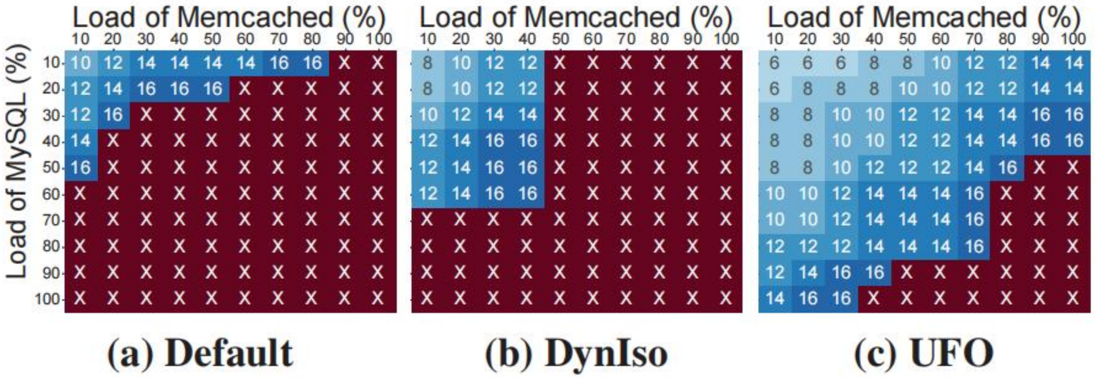


Figure 22: Colocation of Memcached and MySQL.

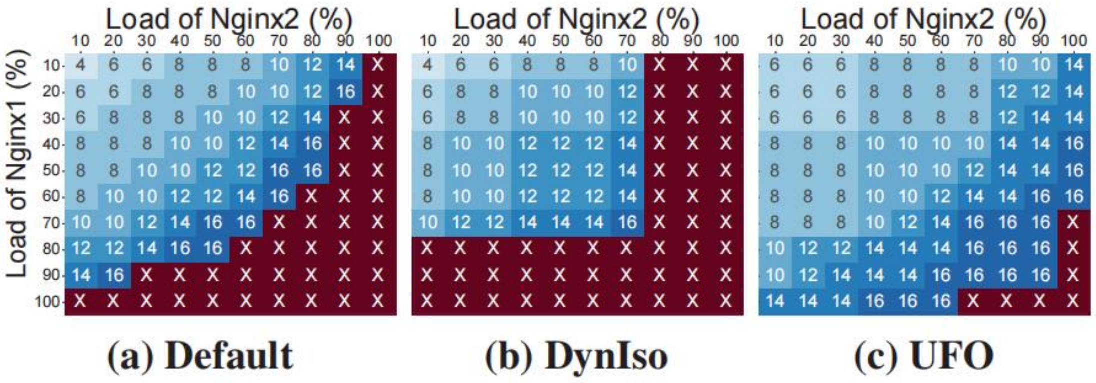


Figure 23: Colocation of Nginx and Nginx.

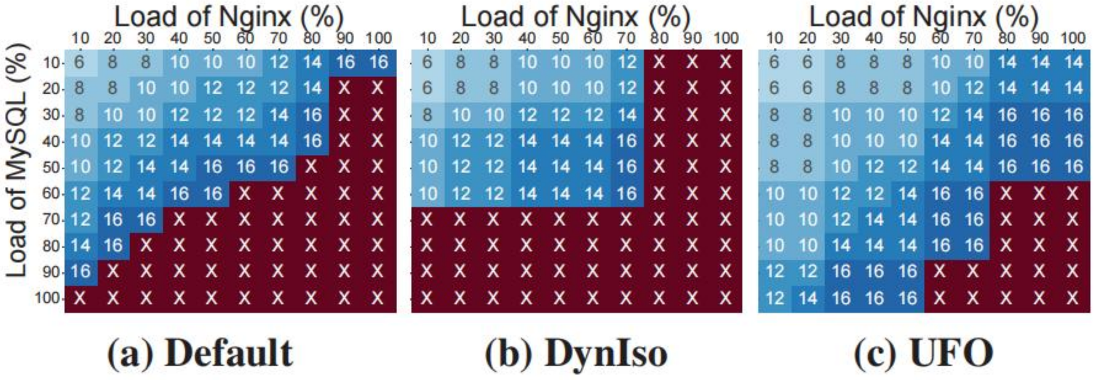


Figure 24: Colocation of Nginx and MySQL.

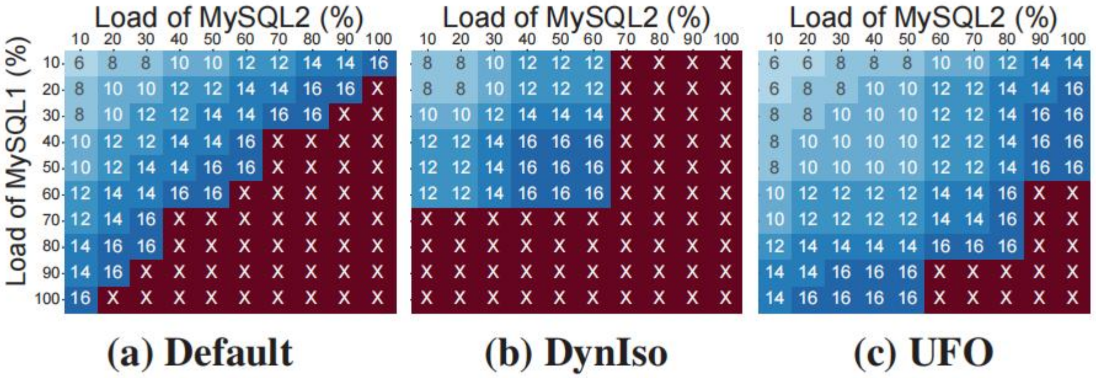


Figure 25: Colocation of MySQL and MySQL.