Progressive Memory Adjustment with Performance Guarantee in Virtualized Systems

Lulu Yao†, Yongkun Li†, Jiawei Li†, Weijie Wu*, Yinlong Xu†

†University of Science and Technology of China

*Independent Researcher

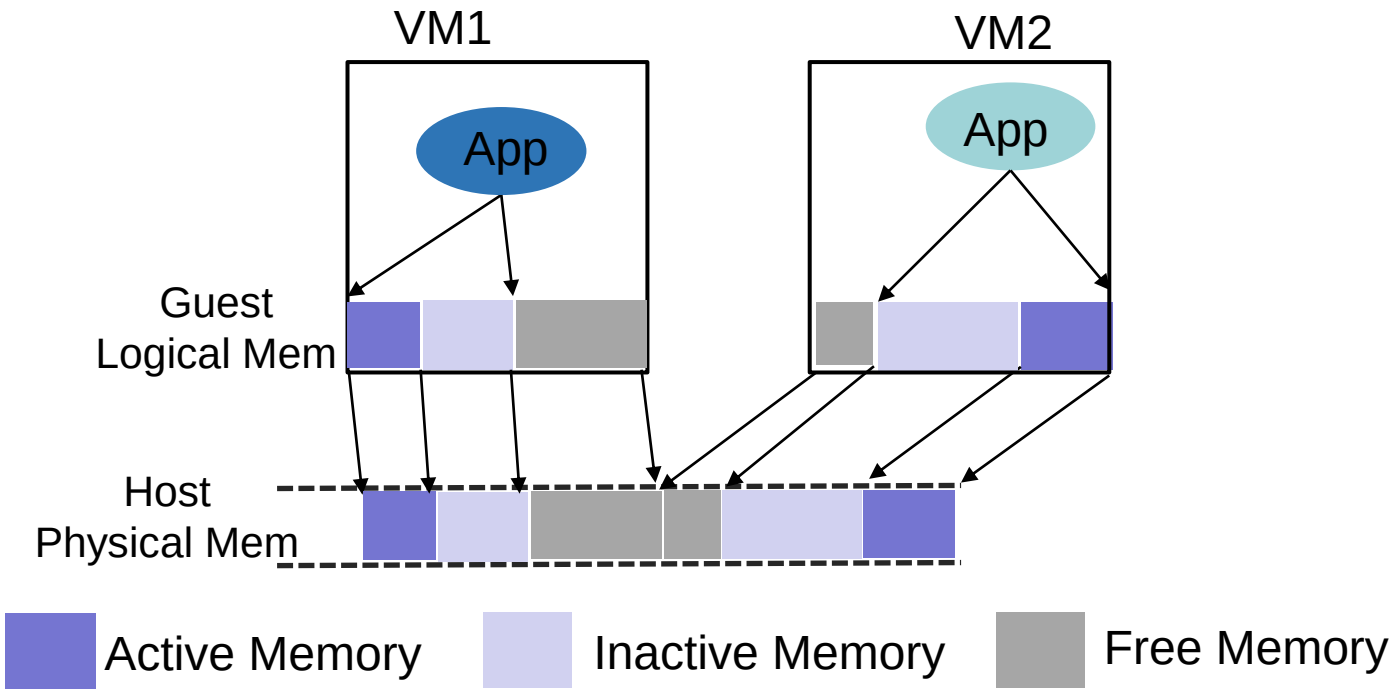
Background

- Cloud computing has been widely adopted in recent years
  - Virtualization is the basic technology of cloud computing
    - Provides virtual machine (VM) to tenants

- Memory is very important in virtualized systems
  - A comparatively scarce resource
  - Rather critical to the performance of VMs

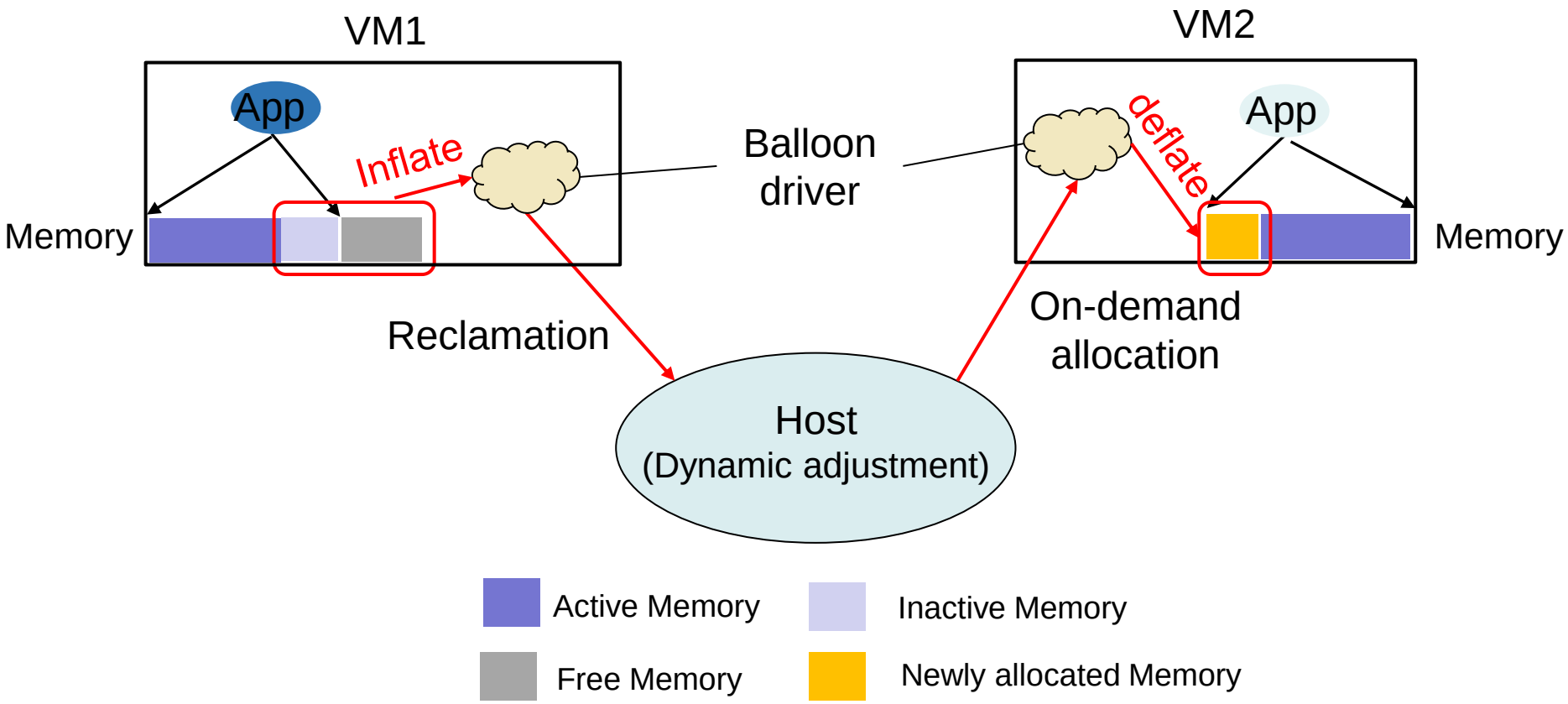
Background

- Inefficient memory usage due to static allocation
  - VMs are configured for peak memory demand
  - After execution, free memory is still occupied by VMs



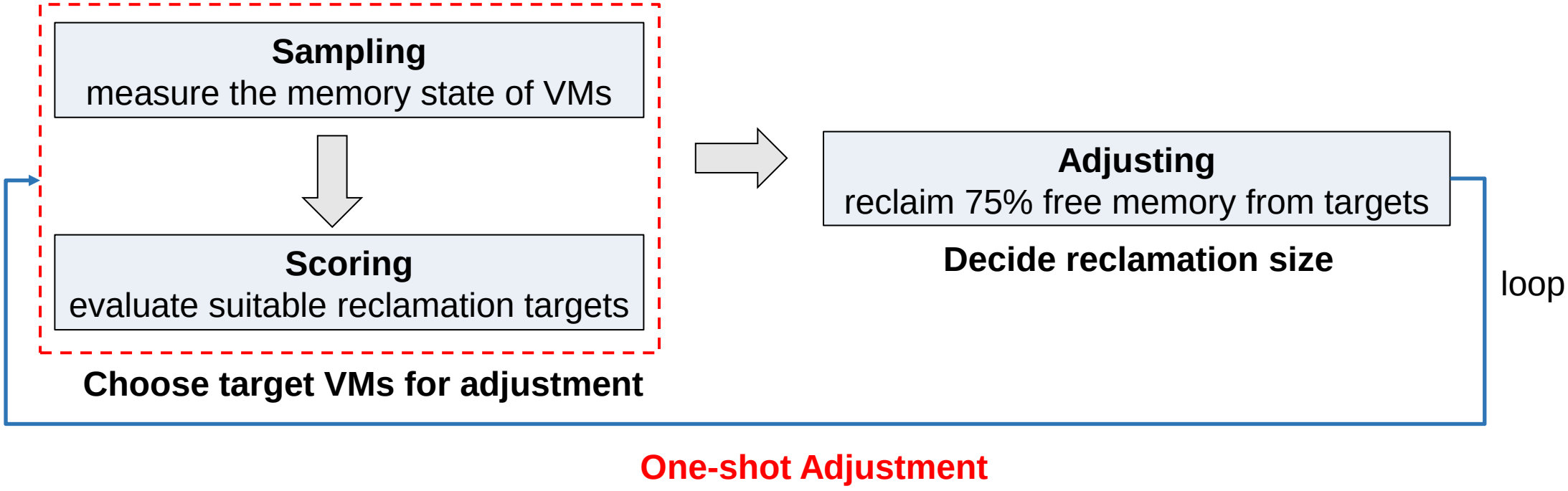
Background

- Dynamic memory adjustment
  - Reclaim inactive/free memory of some VMs and give it to other VMs



Motivation

- State-of-the-art method : One-shot Adjustment
  - **Strong isolation** : guarantee the isolation between host and VMs
  - **Good compatibility** : compatible with different virtualization platforms

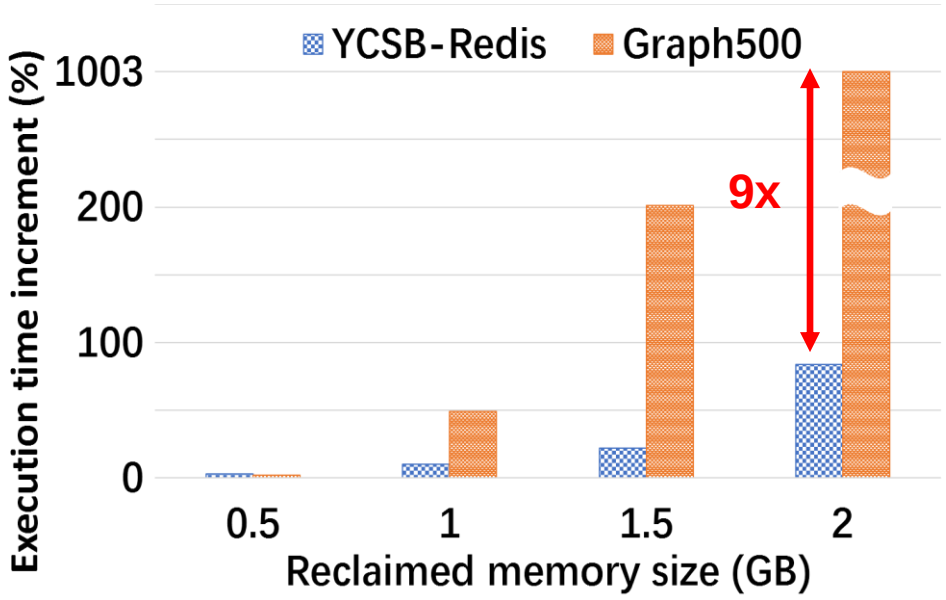


One-shot Adjustment

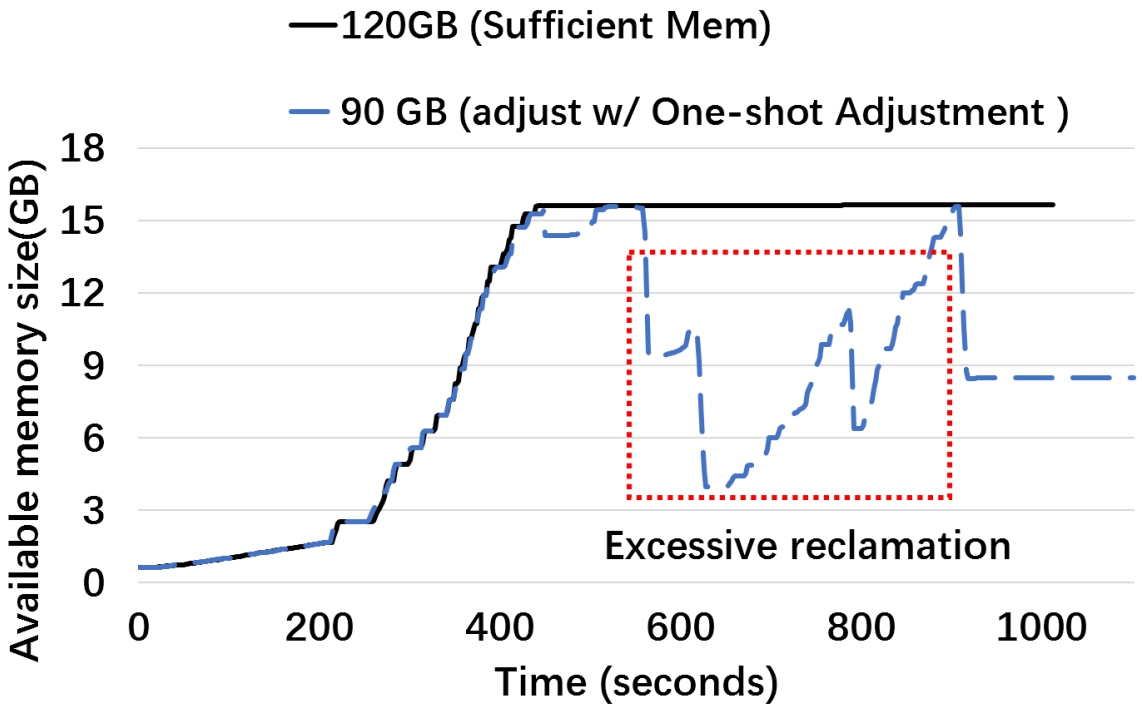
Motivation

➢ Limitation 1: Unware of memory sensitivity of different VMs

- Reclaim fixed percentage (75%) of free memory



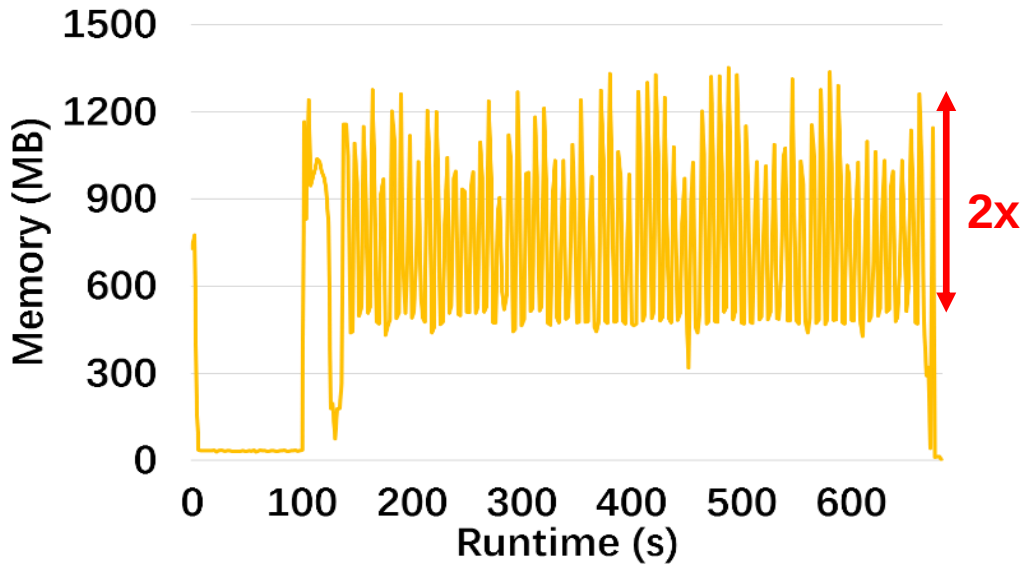
memory sensitivity varies significantly



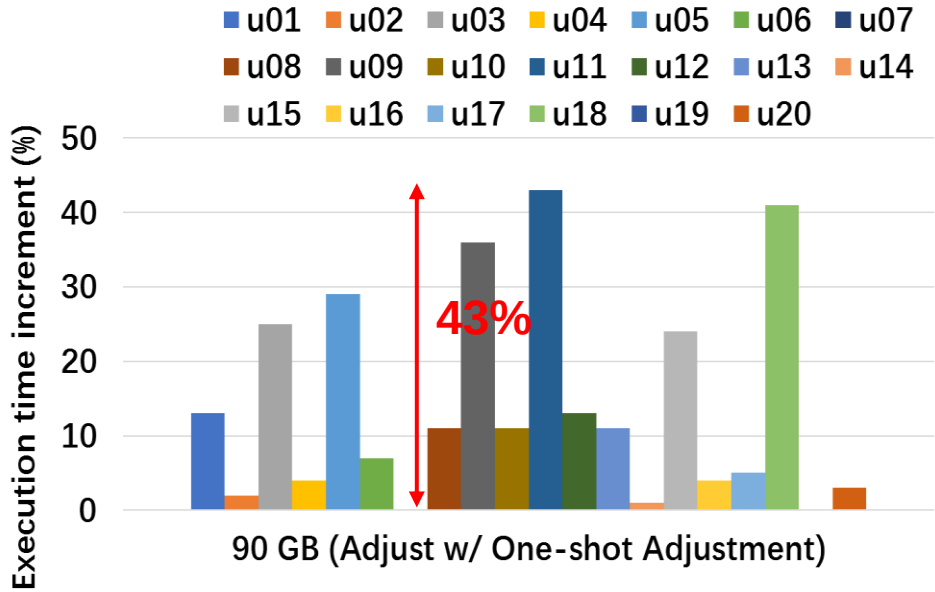
Excessive reclamation results in 41% performance degradation

Motivation

- Limitation 2: Unware of memory access dynamics
  - Result in large differences in performance (execution time increment) loss



Memory requirements of Apps change dynamically



Large variation in performance

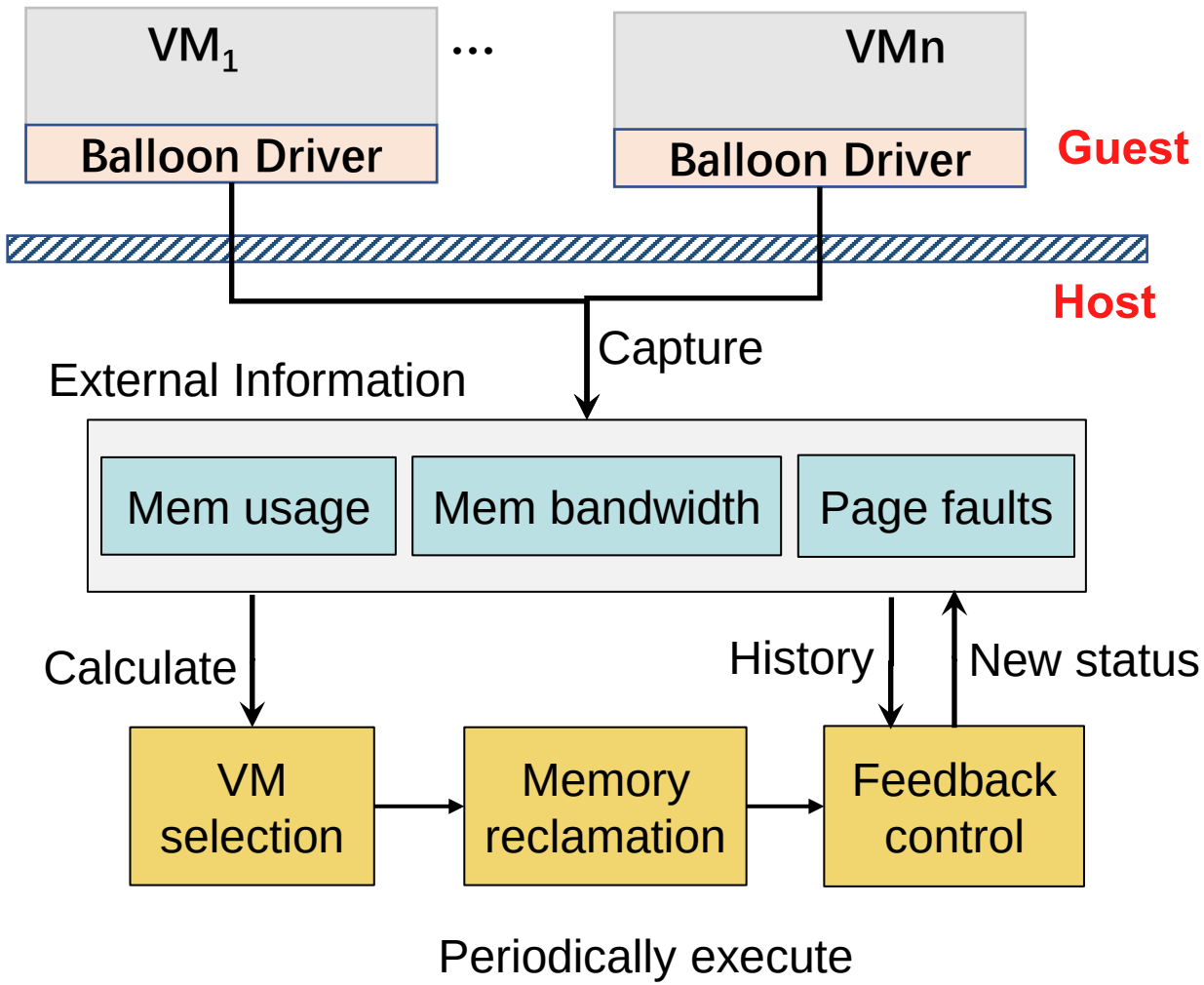
Problems & Challenges

- **Problems with One-shot Adjustment**
  - Unware of memory sensitivity
  - Unware of memory access dynamics
- Challenges
  - VMs are black box
    - Hard to know internal memory state
    - Hard to perceive performance impact

PMA : Progressive memory adjustment with feedback control

Design Overview

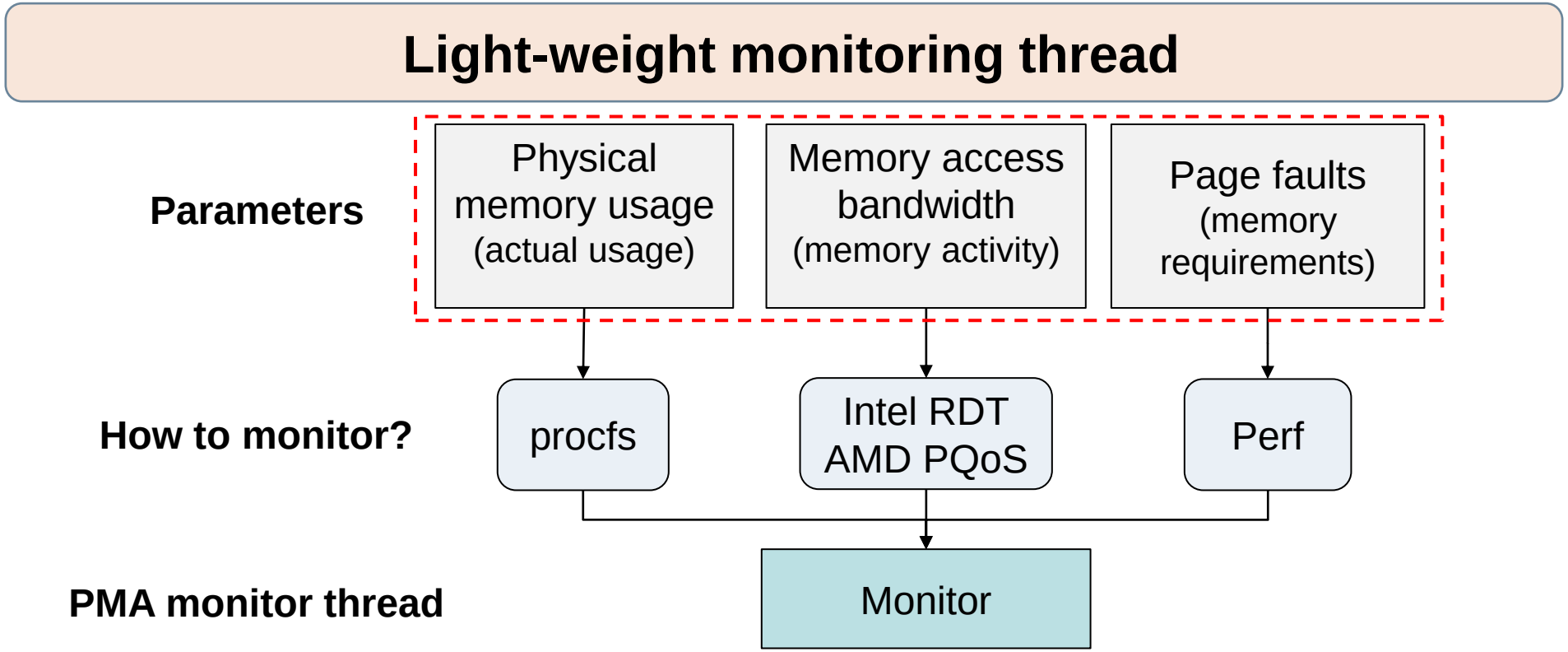
- **Step 1 : Target VM selection**
  - Choose low memory sensitivity VMs
- **Step 2 : Memory reclamation**
  - Progressively increase reclamation size
- **Step 3 : Feedback control**
  - Continue or stop reclamation depending on performance degradation



PMA Design

➢ **Question 1**: Which VM(s) to choose for memory adjustment?

- Which parameters reflect memory access patterns?



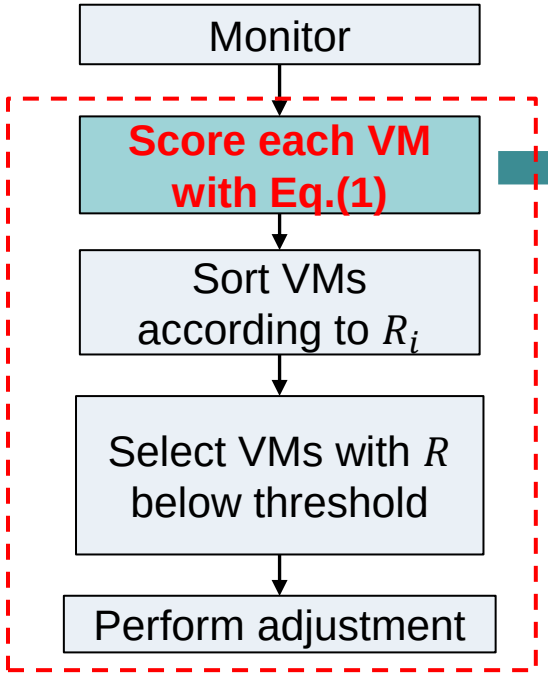
PMA Design

➢ **Question 1**: Which VM(s) to choose for memory adjustment?

- How to evaluate the memory state of VMs?
  - Memory demand is positively correlated to memory access bandwidth

Simple and effective evaluation model

VM Memory State Evaluator



Eq.(1)

$$R_i = \frac{B_i}{\sum_{j=1}^{N} B_j} - \frac{S_i}{\sum_{j=1}^{N} S_j} + f_i \times \frac{P_i}{\sum_{j=1}^{N} P_j}$$

R_i represents memory demand of VM_i

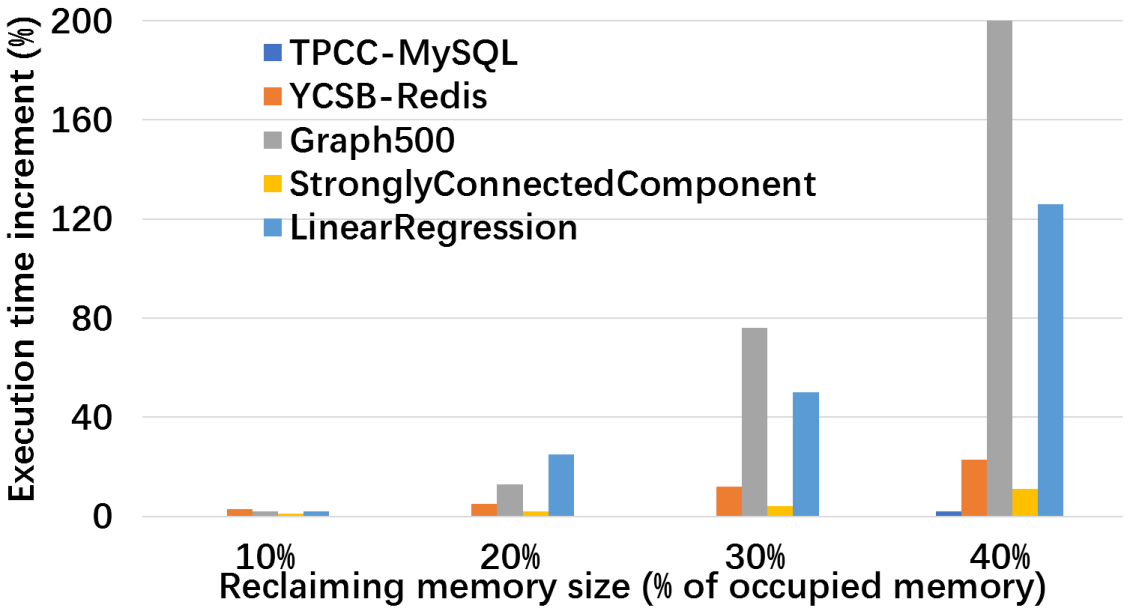
Percentage of memory access bandwidth (B), memory occupied (S) and page faults (P)

(e.g., Small B, large S and small P implies VM_i has a large amount of inactive memory)

PMA Design

➢ **Question 2**: How much memory to reclaim each time?

Progressive memory adjustment



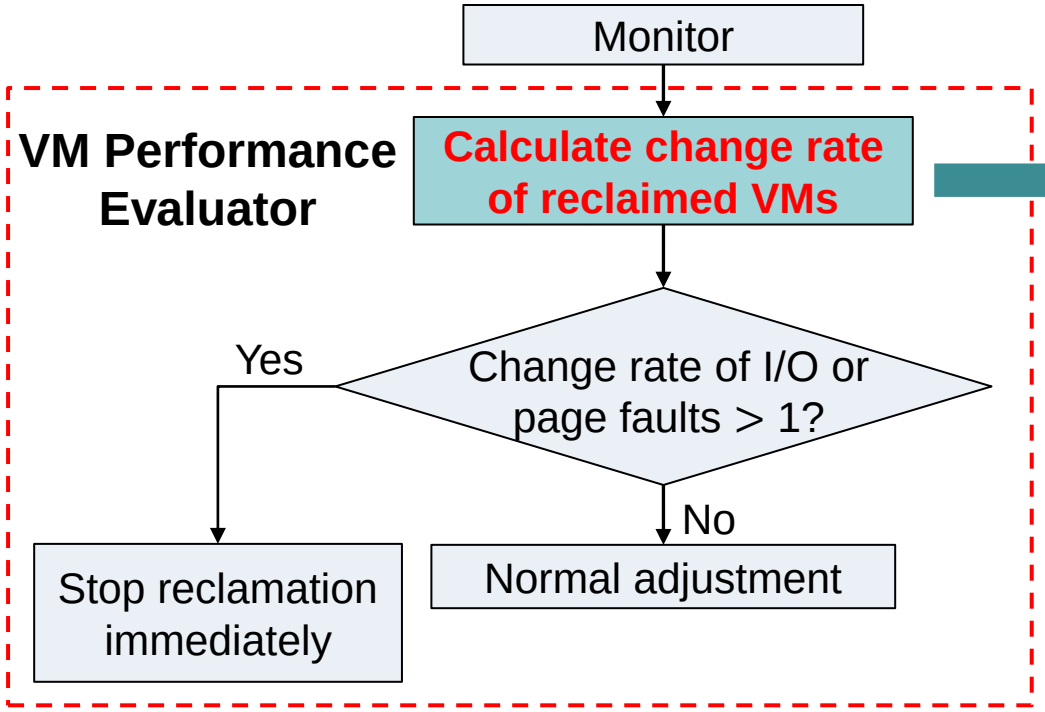
Empirical initial memory reclamation size (10%)

- **How to set the initial reclamation size**
  - Consider the difference in memory sensitivity
    - Negligible impact on performance
    - Empirical value is 10%
- **How to adjust the reclamation size**
  - Consider the reclamation process
    - Double the amount of previous reclamation

PMA Design

➢ **Question 3**: How to avoid excessive reclamation?

Memory adjustment with performance feedback



Performance degradation detection

- **How to judge performance degradation**
  - Consider I/O and memory allocation
    - Change rate of I/O size & page faults after reclamation
- **How to balance performance impact**
  - The amplification factor of R_i to avoid reclaiming the same VM consecutively

$$R_i = \frac{B_i}{\sum_{j=1}^{N} B_j} - \frac{S_i}{\sum_{j=1}^{N} S_j} + \boldsymbol{f_i} \times \frac{P_i}{\sum_{j=1}^{N} P_j}$$

Evaluation

- Server configuration

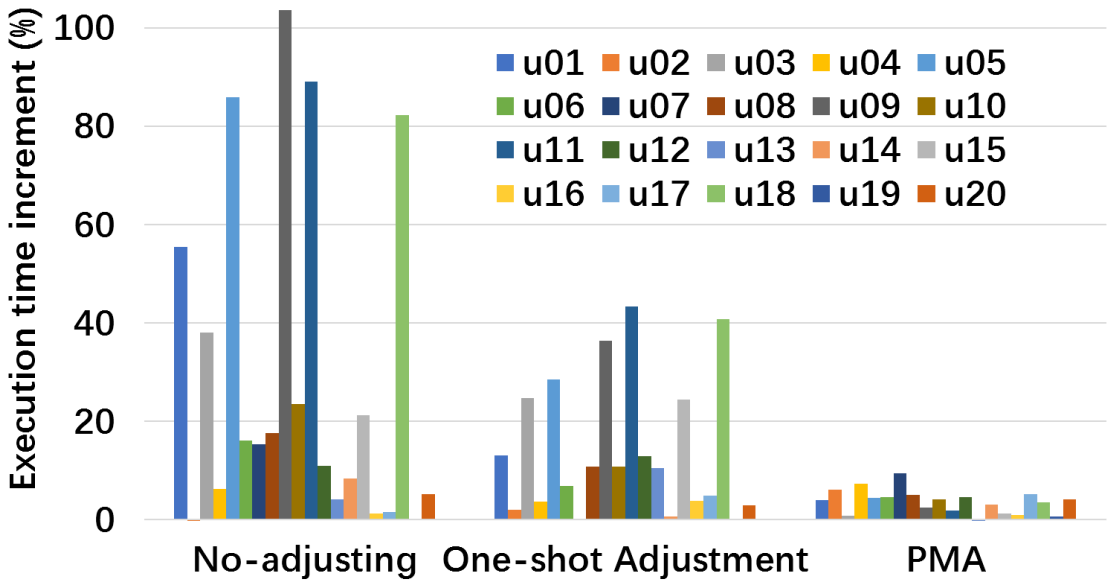
CPU	Memory	OS	Virtualization
2* (Intel(R) Xeon(R) E5-2650 v4 @ 2.20GHz 12-cores)	128GB RAM	16.04.1-Ubuntu 64-bit Linux 4.19.24	QEMU + KVM

- Experimental configuration
  - Workloads: 20VMs with a total of 120GB memory occupy
  - VM configuration
    - With a unique ID , run one application from one benchmark suite (total 20 Apps)
  - Benchmark Suite

VM ID	u01, u02, u05, u06, u08, u09, u10, u11	u12, u13, u14, u15, u16	u17, u18, u19, u20	u03	u04	u07
Suit Name	Spark-Bench	PARSEC	NPB	TPC-C	Graph500	YCSB, Redis

Exp 1: Performance Results

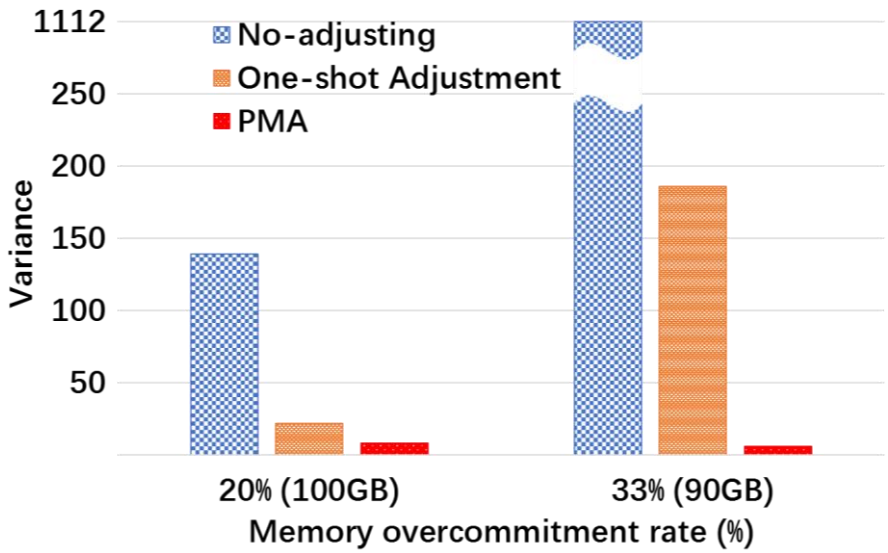
- 20 VMs with 33% memory overcommitment rate
  - 20 VMs require 120GB memory, while the host memory is limited to 90GB
    - Memory overcommitment rate=(120-90)/90=33%



PMA **limits the performance loss of all VMs within 10%**

Exp 2: Performance variance

- 20 VMs with 20% & 33% memory overcommitment rate
  - 20% rate : limits the host memory size to 100GB
  - Variance represents the degree of dispersion of performance

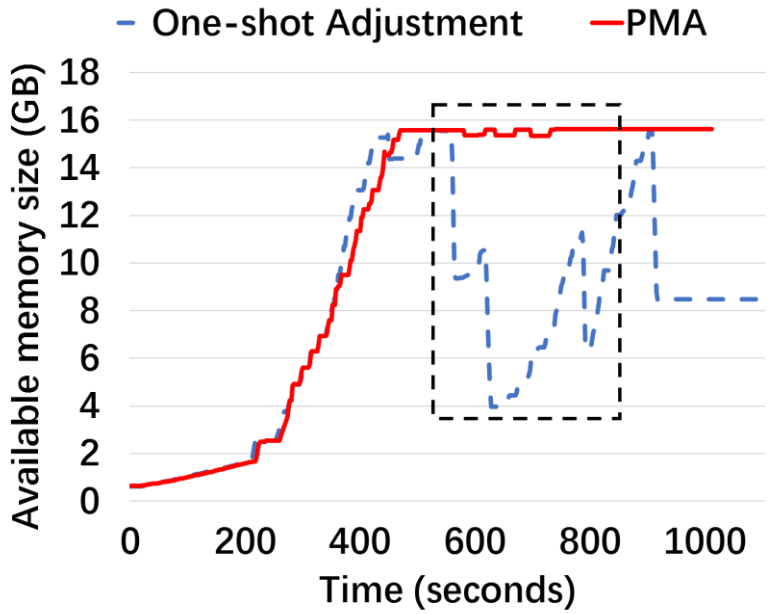


	Technology	20%	33%
Max-Min (Range)	No-adjusting	41	104
	One-shot Adjustment	23	43
	PMA	**15**	**10**

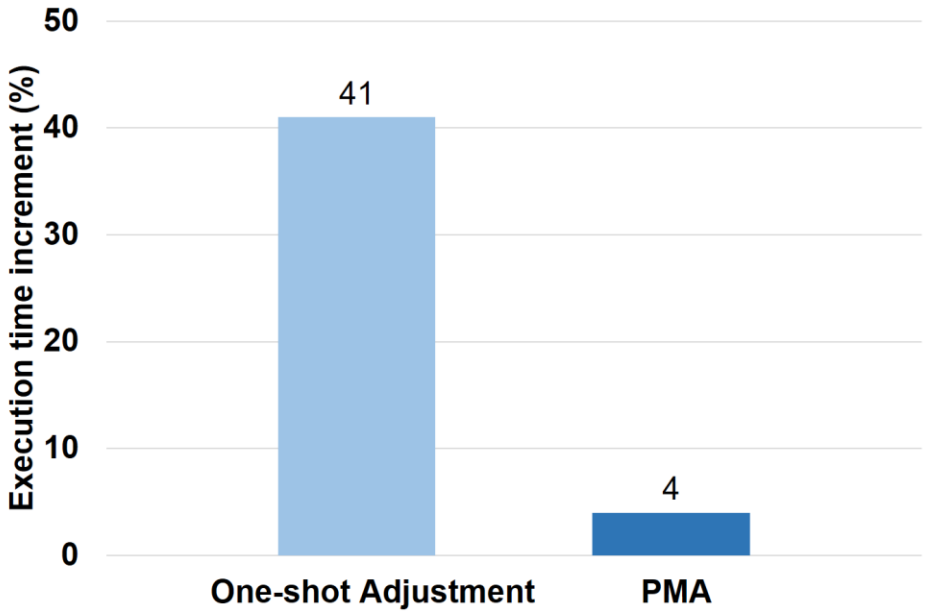
PMA makes virtual machine **performance more balanced**

Exp 3: Study of excessive reclamation

- 20 VMs with 33% memory overcommitment rate



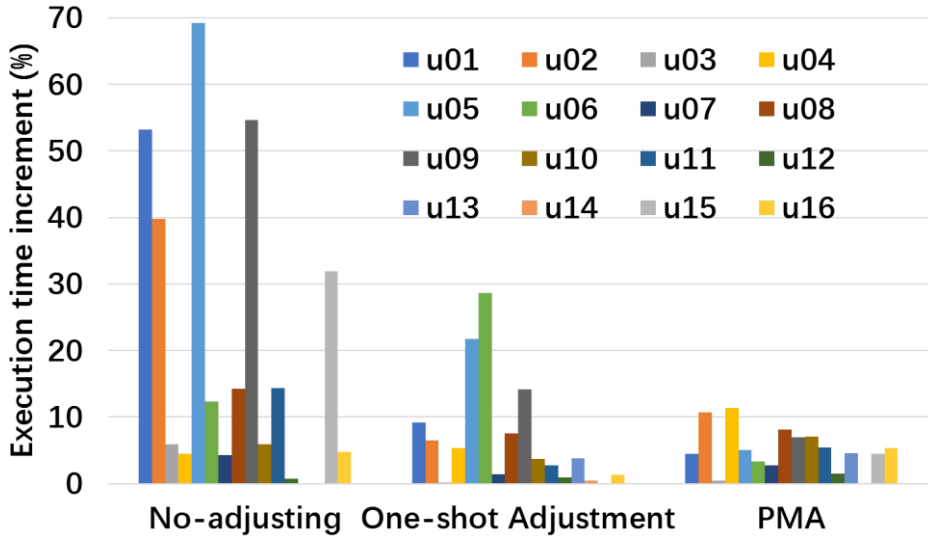
VM u18



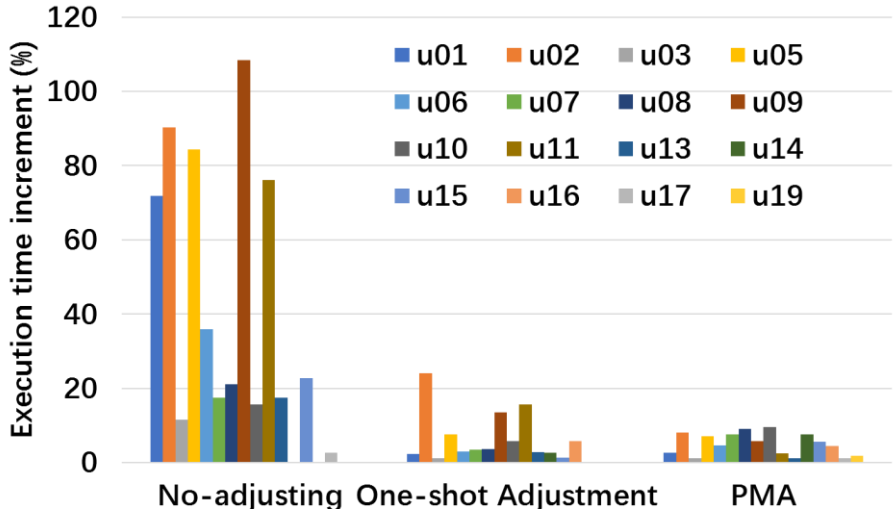
PMA effectively avoid performance loss caused by excessive reclamation

Exp 4: Performance results under 16 VMs

- 16 VMs with 33% memory overcommitment rate



Fixed selection (i.e.,u01~u16)



Randomly selected 16 VMs

PMA achieves more balanced performance & less performance degradation

Conclusions

- PMA: a progressive memory adjustment with feedback control
  - Consider memory access dynamics and memory sensitivity of VMs
  - Progressive adjustment with feedback control to minimize the performance degradation of VMs
  - Retain strong isolation between the host and VMs

- More design details and evaluation results are in the paper

Thanks for your attention!

Q&A