OSML

Intelligent Resource Scheduling for Co-located Latency-critical Services: A Multi-Model Collaborative Learning Approach

Lei Liu[1], Xinglei Dou[2] (Presenter), Yuetao Chen[2]

[1]Beihang University; [2]ICT, CAS; Sys-Inventor Lab

FAST-2023

Session: AI and Storage

Executive Summary

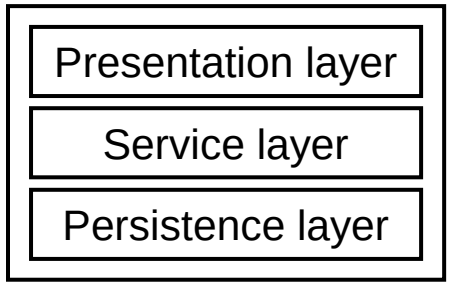
- Runtime resource scheduling becomes the pivot for Quality of Service (QoS) control in complicated co-location cases.
- **Challenges:**
  - Co-located services exhibit **diverse behaviors** across the storage hierarchy.
  - **Enlarging scheduling exploration space** and **multiple interactive resources** make it hard for schedulers to provide ideal solutions quickly and efficiently.
  - “Resource cliffs” (**RCliff**) reduces the exploration efficiency and lead to severe QoS fluctuations.
- **Solutions:**
  - Data-driven approach based on extensive traces
  - Collaborative ML models for intelligent scheduling
- **Results:**
  - higher loads and shorter convergence time than prior work

Outline

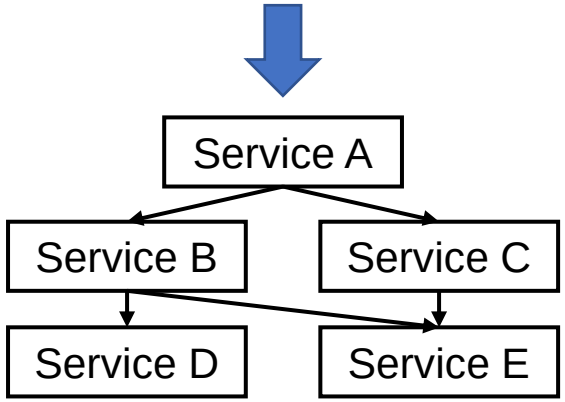
- **Motivation**
- Study in Resource Scheduling for LC services
- Leveraging ML for scheduling
- Evaluations
- Conclusion

Monolithic Architecture → Loosely-coupled Design

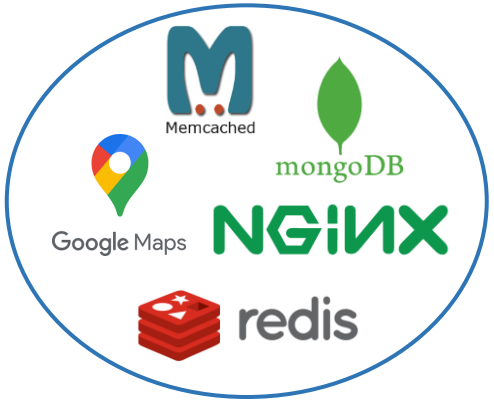
- Cloud applications are shifting from monolithic architectures to loosely-coupled designs, including many latency-critical (LC) services with strict quality of service (QoS) requirements.



Monolithic architecture



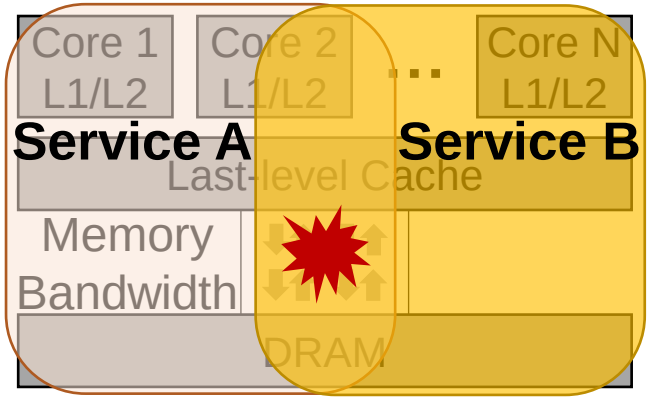
Loosely-coupled design



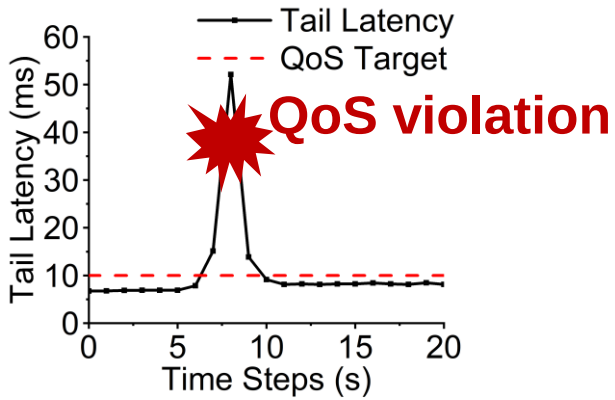
Latency-critical services

Co-location → Contention for Shared Resources

- Co-located services exhibit diverse behaviors across the **storage hierarchy**, including multiple **interactive resources** such as CPU cores, last level cache (LLC), memory/IO bandwidth, and main memory banks.
- Co-located services contend for shared resources, leading to Quality of Service (QoS) violations.



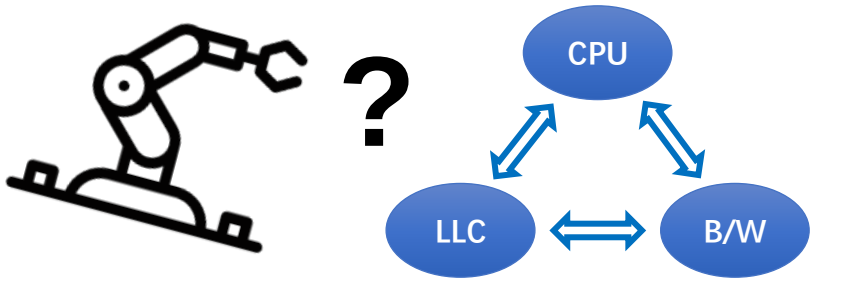
Contention in shared resources



QoS violations of Latency-critical (LC) services

New Scheduling Approaches are Expected

- Existing schedulers still have room for improvement in **scheduling convergence time**, **intelligence**, and how to schedule complicated **interactive resources** in a timely fashion.



Interactive resources

- Existing schedulers cannot easily avoid "**resource cliffs**", i.e., decreasing a resource only slightly during scheduling leads to a significant QoS slow-down.

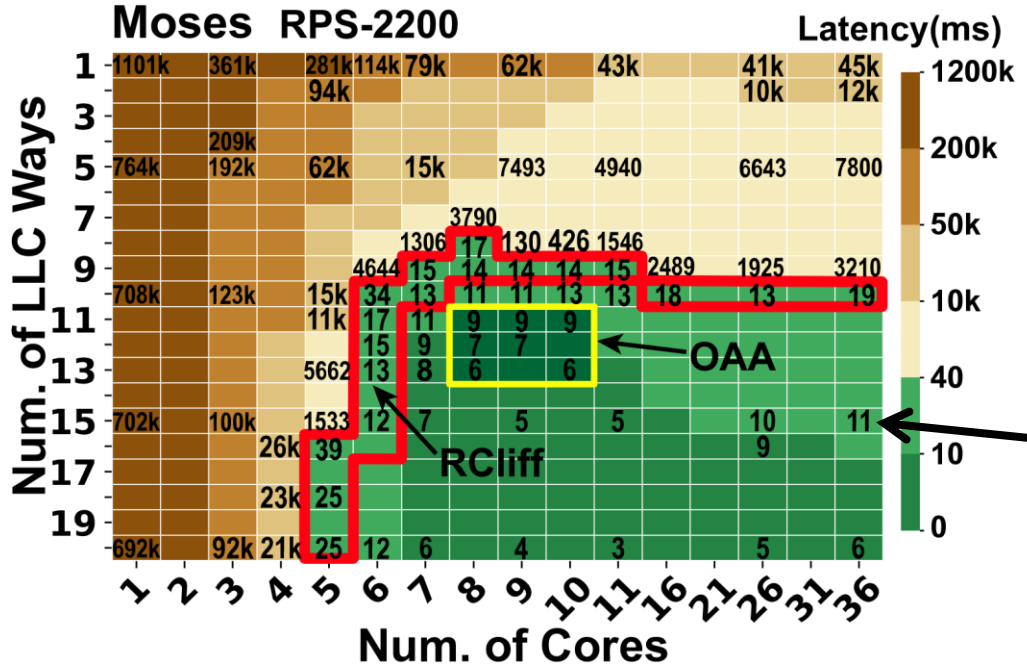
QoS violations

Outline

- Motivation
- **Study in Resource Scheduling for LC services**
- Leveraging ML for scheduling
- Evaluations
- Conclusion

Key Observations: RCliff and OAA

- **Resource Cliff (RCliff):** the resource allocation cases that could incur the most significant performance slowdown if resources (e.g., core, cache) are deprived of via a fine-grain way in the scheduling exploration space
- **Optimal Allocation Area (OAA):** the ideal number of allocated cores and LLC ways to bring an acceptable QoS

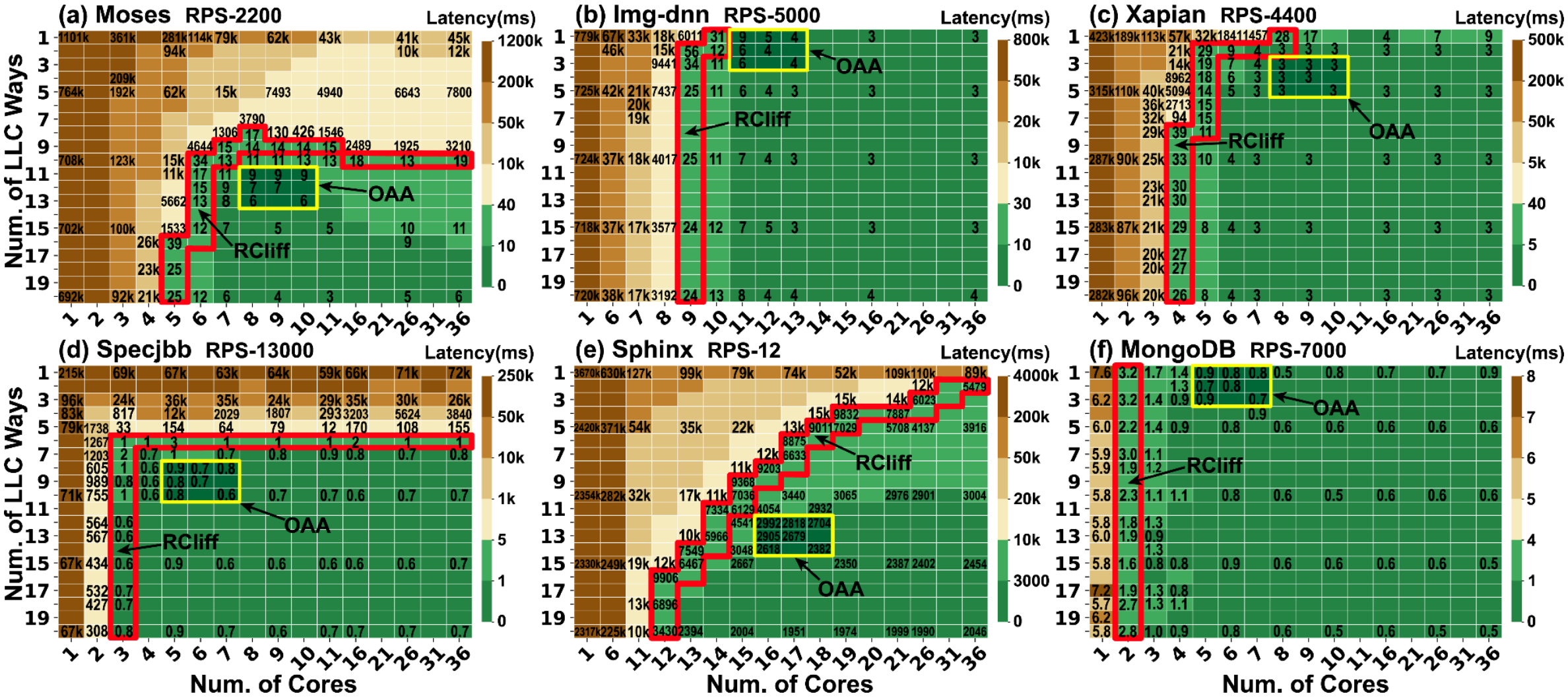


A resource scheduling exploration space containing 36 (#cores) * 20 (#LLC ways) possible allocations

Each cell denotes the LC service' s response latency under the given number of cores and LLC ways

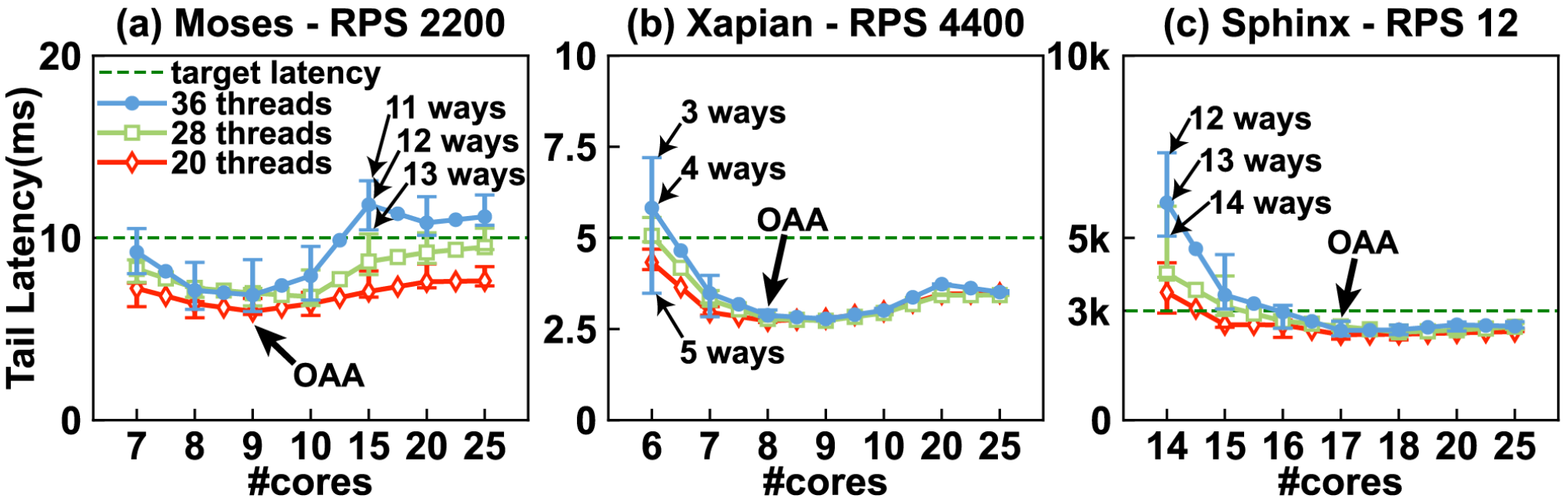
Key Observations: RCliff and OAA

- **RCliff and OAA commonly exist for many LC services.**



Is OAA Sensitive to the Number of Threads?

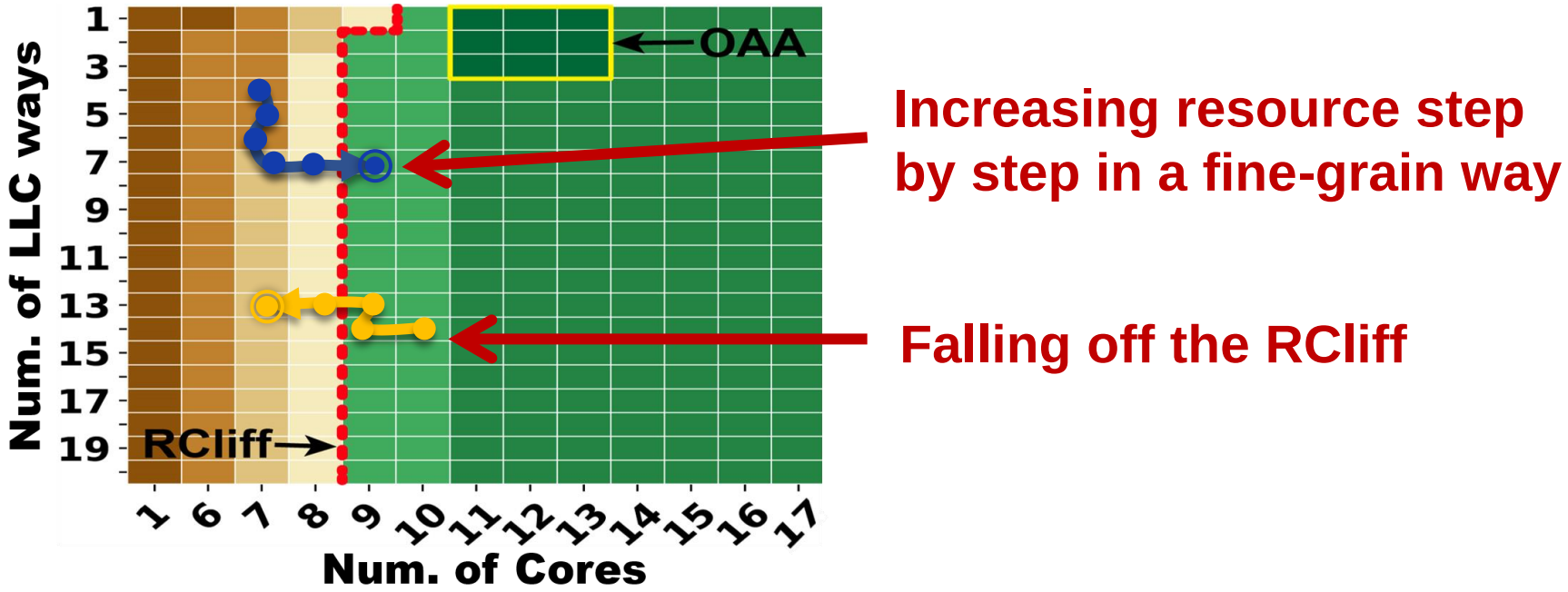
- More threads do not necessarily bring more benefits.
- The OAA is not sensitive to the number of concurrent threads.



OAA exists regardless of the number of concurrent threads

Issues the Existing Schedulers May Meet

- Entangling with RCliffs
- Difficulty in accurately and simultaneously scheduling a combination of multiple interactive resources (e.g., cores, LLC ways) to achieve OAAs in low overheads
- Difficulty in providing accurate QoS predictions



Example: Existing schedulers are entangling with RCliffs

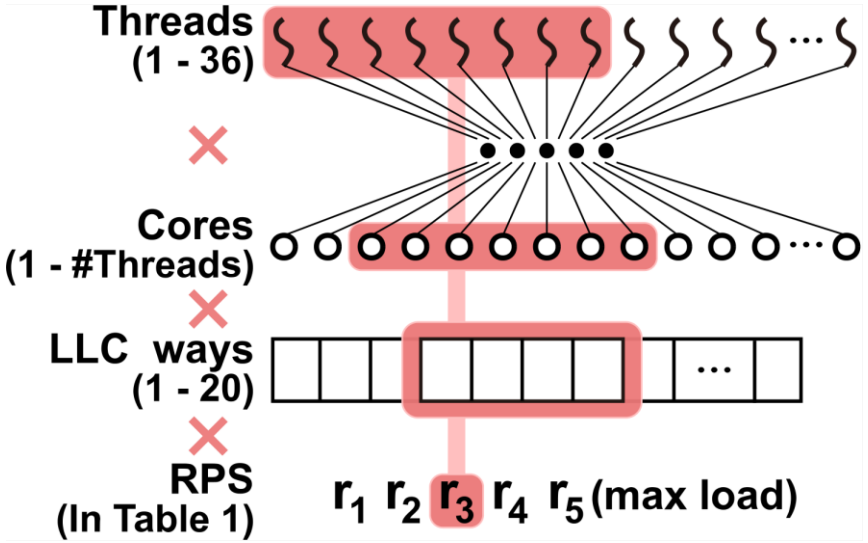
Outline

- Motivation
- Study in Resource Scheduling for LC services
- **Leveraging ML for scheduling**
- Evaluations
- Conclusion

OSML - a Data-driven Approach

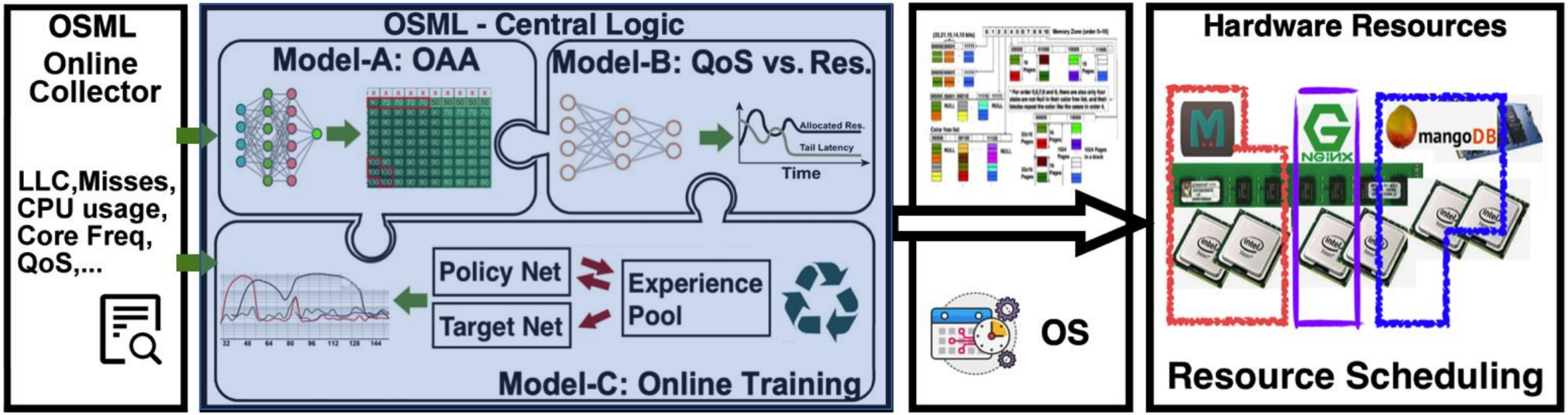
- **Data collection**
  - 11 representative LC services
  - Common RPS demands
  - 1-36 threads
  - Map on 1 - #Threads cores
  - 1-20 LLC ways
- **Data set from 11 typical LC services**
  - 62,720,264 resource allocation cases, containing around 2-billion samples

LC service	Domain	RPS (Requests Per Second)
Img-dnn [62]	Image recognition	2000,3000,4000,5000,6000 (Max)
Masstree [62]	Key-value store	3000,3400,3800,4200,4600
Memcached [65]	Key-value store	256k,512k,768k,1024k,1280k
MongoDB [64]	Persistent database	1000,3000,5000,7000,9000
Moses [62]	RT translation	2200,2400,2600,2800,3000
Nginx [66]	Web server	60k,120k,180k,240k,300k
Specjbb [62]	Java middleware	7000,9000,11000,13000,15000
Sphinx [62]	Speech recognition	1,4,8,12,16
Xapian [62]	Online search	3600,4400,5200,6000,6800
Login [68]	Login	300,600,900,1200,1500
Ads [68,52]	Online renting ads	10,100,1000



OSML – Using ML for Resource Scheduling

- **Multi-model collaborative learning approach**
  - Model-A: Aiming Optimal Allocation Area
  - Model-B: Balancing QoS and Resources
  - Model-C: Handling the Changes On the Fly
- **OSML is designed as a co-worker of the OS scheduler located between the OS kernel and the user layer**



Summary of the ML Models

Inputs

Feature	Description	Models
IPC	Instructions per clock	A/A'/B/B'/C
Cache Misses	LLC misses per second	A/A'/B/B'/C
MBL	Local memory bandwidth	A/A'/B/B'/C
CPU Usage	The sum of each core's utilization	A/A'/B/B'/C
Virt. Memory	Virtual memory in use by an app	A/A'/B/B'
Res. Memory	Resident memory in use by an app	A/A'/B/B'
Allocated Cores	The number of allocated cores	A/A'/B/B'/C
Allocated Cache	The capacity of allocated cache	A/A'/B/B'/C
Core Frequency	Core Frequency during run time	A/A'/B/B'/C
QoS Slowdown	Percentage of QoS slowdown	B
Expected Cores	Expected cores after deprivation	B'
Expected Cache	Expected cache after deprivation	B'
Cores used by N.	Cores used by Neighbors	A'/B/B'
Cache used by N.	Cache capacity used by Neighbors	A'/B/B'
MBL used by N.	Memory BW used by Neighbors	A'/B/B'
Resp. Latency	Average latency of a LC service	C

Details of the ML models

ML	Model	Features	Model Size	Loss Function	Gradient Descent	Activation Function
A	MLP	9	144 KB	Mean Square Error (MSE)	Adam Optimizer	ReLU
A'	MLP	12	155 KB			
B	MLP	13	110 KB	Modified MSE		
B'	MLP	14	106 KB	MSE		
C	DQN	8	141 KB	Modified MSE	RMSProp	

Models → outputs

Model-A (MLP) → **Outputs:** RCliff, OAA, OAA bandwidth (for **a single service**)
Format: <RCliff cores, RCliff LLC ways, OAA cores, OAA LLC ways, [OAA bandwidth]>

Model-A' (MLP) → **Outputs:** OAA, RCliff, OAA bandwidth (for **co-location cases**)
Format: <RCliff cores, RCliff LLC ways, OAA cores, OAA LLC ways, [OAA bandwidth]>

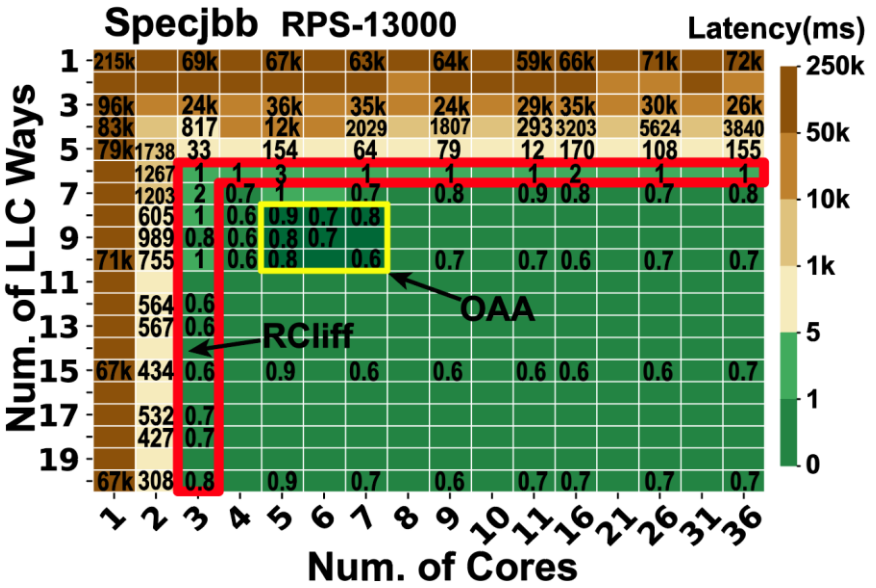
Model-B (MLP) → **Outputs:** The resources that a service **can be deprived of** under allowable QoS slowdown
Format: <Cores, LLC ways>, <Cores (dominated), LLC ways>, <Cores, LLC ways (dominated)>

Model-B' (MLP) → **Outputs:** How much **QoS slowdown** will suffer if a certain amount of resources is deprived of a specific service
Format: <QoS slowdown>

Model-C (DQN) → **Outputs:** The **scheduling actions** (reducing/increasing a specific number of cores/LLC ways)
Format: <Core step, LLC way step>

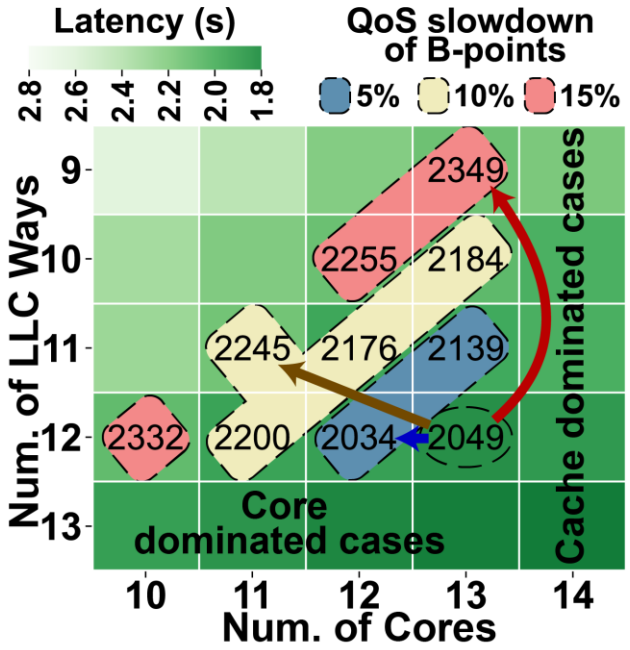
Model-A: Aiming OAA

- **Model-A predicts RCliff, OAA and OAA bandwidth.**
- **Model-A inputs:**
  - <IPC, Cache Misses, MBL, CPU Usage, Virt. Memory, Res. Memory, Allocated Cores, Allocated Cache, Core Frequency>
- **Model-A outputs:**
  - <RCliff cores, RCliff LLC ways, OAA cores, OAA LLC ways, OAA bandwidth>
- **Model-A' is used in co-location cases.**
- **Model-A' inputs:**
  - Input features of Model-A
  - <Cores used by neighbors, Cache used by neighbors, Memory BW used by neighbors>
- **Model-A' outputs:**
  - Same as Model-A



Model-B: Balancing QoS and Resources

- **Model-B balances the QoS and resource allocations among co-located LC services.**
- **Model-B inputs:**
  - Input features of Model-A'
  - QoS Slowdown
- **Model-B outputs** 3 policies as the computing units and memory resource can be **fungible**:
  - <Cores, LLC ways>, <Cores (dominated), LLC ways>, <Cores, LLC ways (dominated)>



- **Model-B' predicts QoS slowdown if a certain amount of resources is deprived of a specific service.**
- **Model-B' inputs:**
  - Input features of Model-A'
  - <Cores after deprivation, Cache after deprivation>
- **Model-B' outputs:**
  - QoS slowdown

Model-C: Handling the Changes On the Fly

- **Model-C shepherds the allocations and recovers from the QoS violation and resource over-provision cases.**
- **Model-C inputs:**
  - <IPC, Cache Misses, MBL, CPU Usage, Allocated Cores, Allocated Cache, Core Frequency, Response Latency>
- **Action:**
  - {<m,n> | m ∈ [−3,3], n ∈ [−3,3]}
  - m indicates the action on cores; n indicates the action on LLC ways.
- **Reward:**

Mitigate QoS violations — **Minimize resource usage**

If $Latency_{t-1} > Latency_t$:

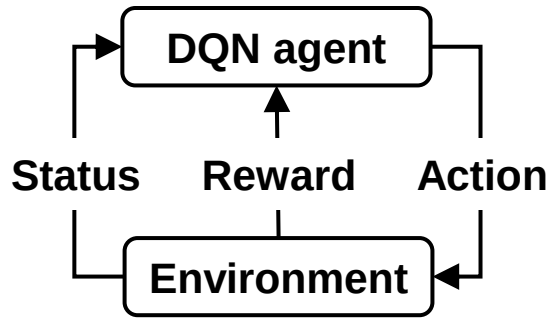
$R_t = \log(1+Latency_{t-1}-Latency_t) + (\Delta CoreNum+\Delta CacheWay)$

If $Latency_{t-1} < Latency_t$:

$R_t = -\log(1+Latency_t-Latency_{t-1}) + (\Delta CoreNum+\Delta CacheWay)$

If $Latency_{t-1} = Latency_t$:

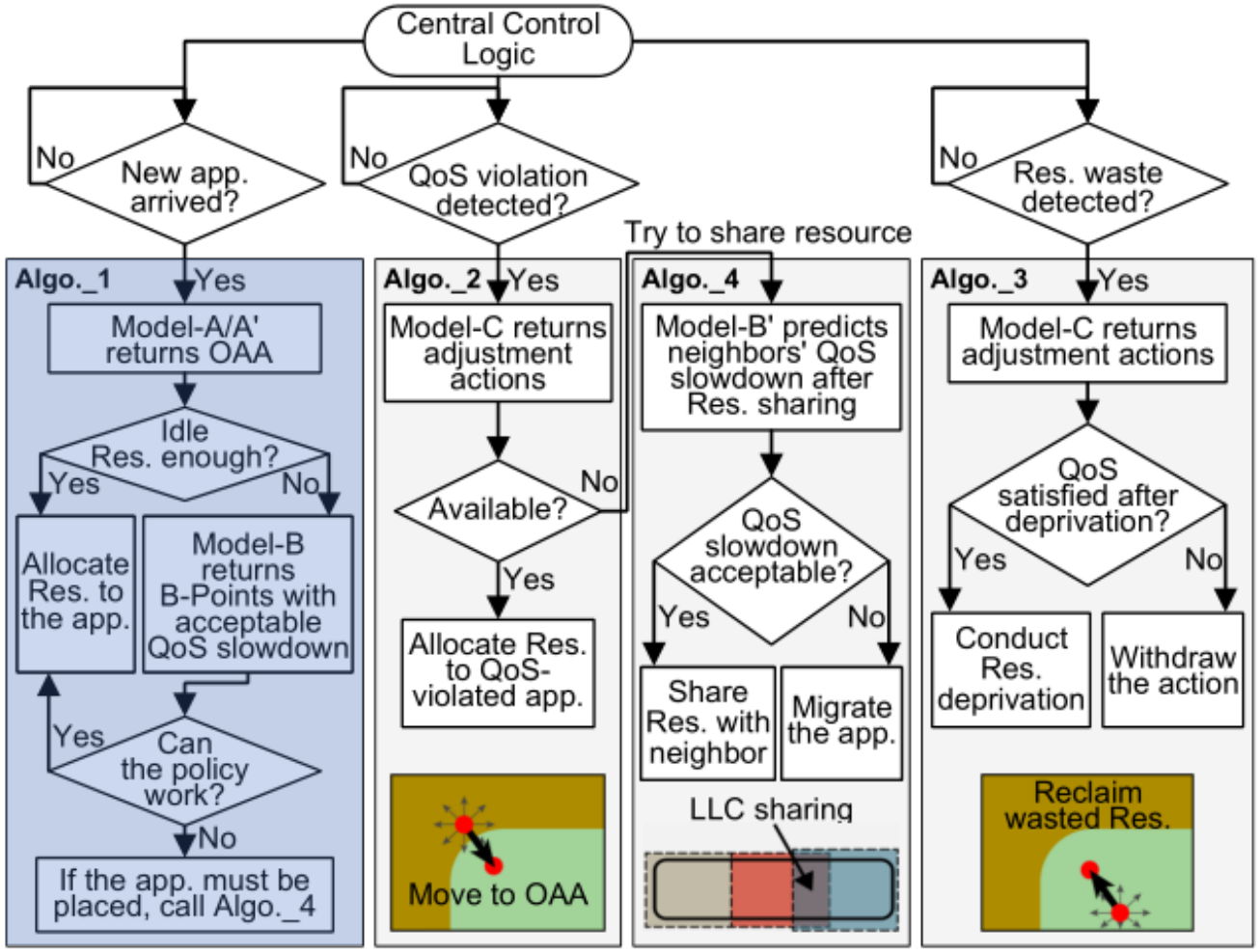
$R_t = -(\Delta CoreNum+\Delta CacheWay)$



Central Logic - Using ML Models in a Pipelined Way

- **Algorithm 1: Allocating resources for a coming LC service**

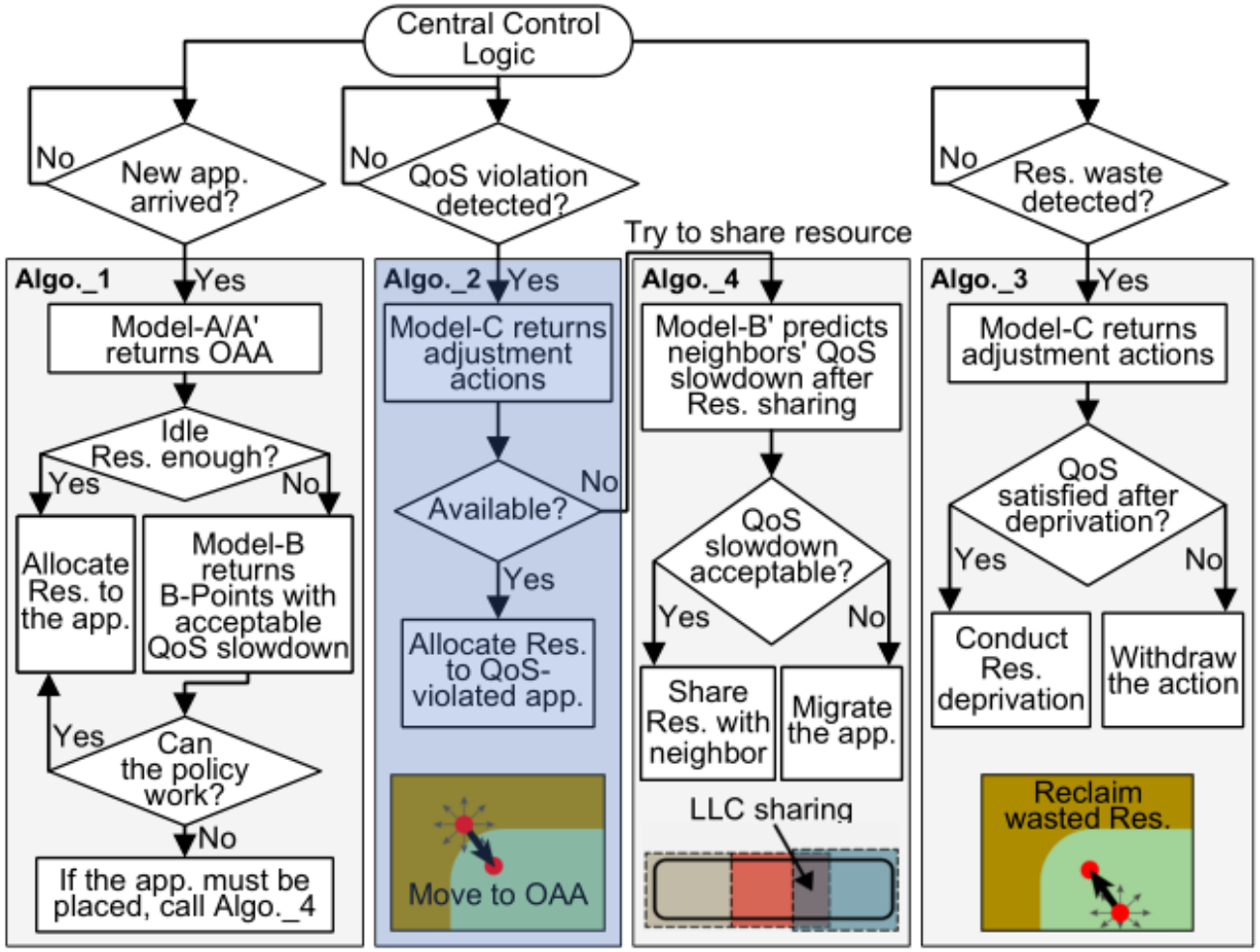
- Enable Model-A/A’ to get the OAA and RCliff
- If idle resources are sufficient to satisfy the new LC service, then allocate resources to the service
- If not, enable Model-B to deprive resources from neighbors and allocate them to the new one
- Enable resource sharing if necessary



Central Logic - Using ML Models in a Pipelined Way

- **Algorithm 2: Handling resource under-provision cases**

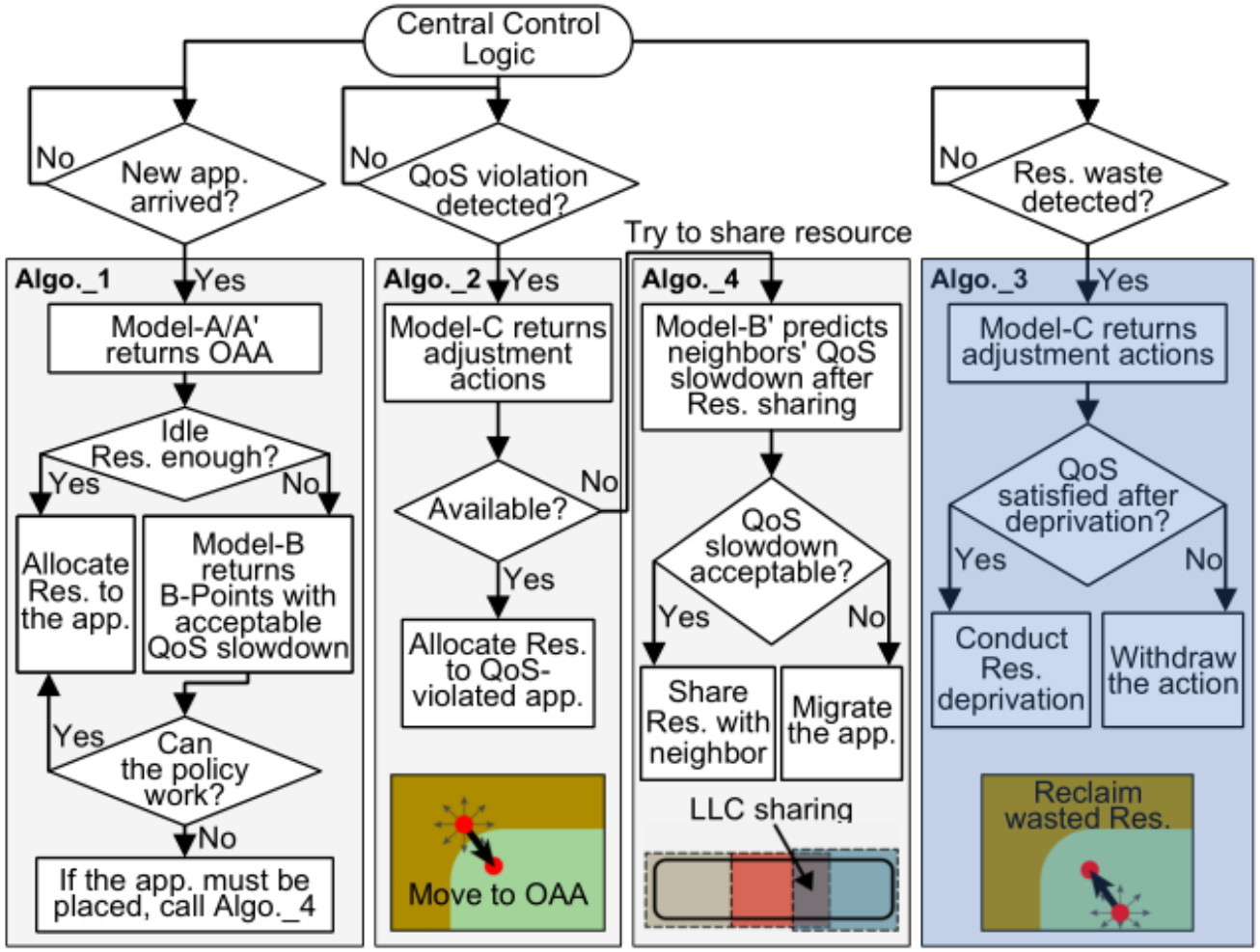
- Call Model-C to allocate more resources to achieve ideal QoS
- Enable resource sharing if necessary



Central Logic - Using ML Models in a Pipelined Way

- **Algorithm 3: Handling resource over-provision cases**

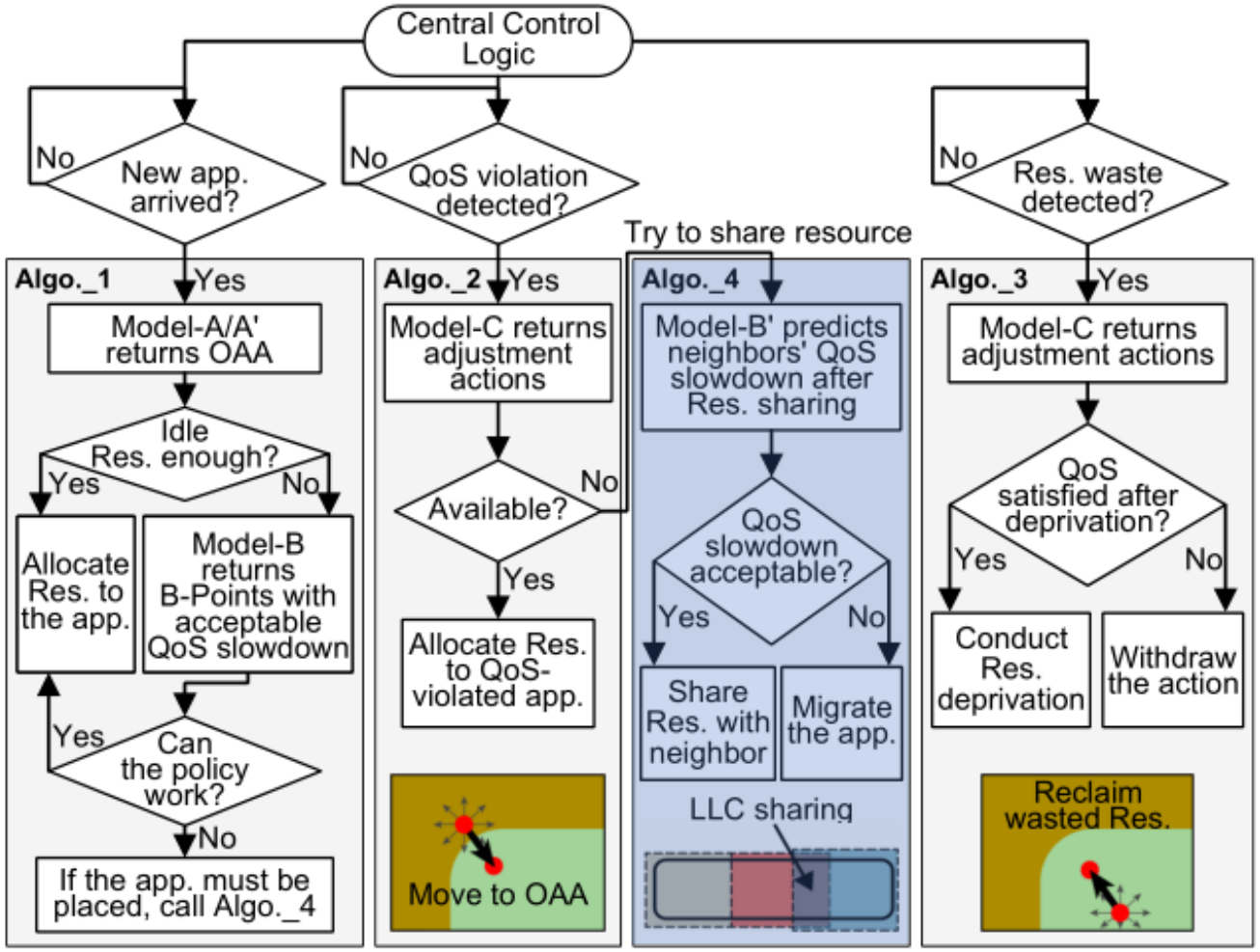
- Call Model-C to reclaim over-provisioned resources
- Withdraw the action if QoS is not satisfied after the deprivation



Central Logic - Using ML Models in a Pipelined Way

- **Algorithm 4: Handling resource sharing among apps**

- Enable Model-B' to predict QoS slowdown after resource sharing
- If the slowdown is acceptable, share resource with neighbor
- If not, migrate the application



Outline

- Motivation
- Study in Resource Scheduling for LC services
- Leveraging ML for scheduling
- **Evaluations**
- Conclusion

Methodology

- **Platform:**
  - Intel Xeon E5-2697 v4, 36 logical cores (18 physical cores), 45MB LLC, 20 LLC ways.

- **Capture traces using:**
  - pqos tool [1]
  - PMU

- **Isolation mechanisms:**
  - CPU cores - Linux taskset
  - LLC ways - Intel Cache Allocation Technology (CAT) [1]
  - Memory bandwidth - Intel Memory Bandwidth Allocation (MBA) [2]

[1] “Improving real-time performance by utilizing cache allocation technology,” https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf, Intel Corporation, April, 2015.
[2] “Intel 64 and IA-32 Architectures Software Developer’s Manual,” https://software.intel.com/en-us/articles/intel-sdm, Intel Corporation, October, 2016.

Evaluations

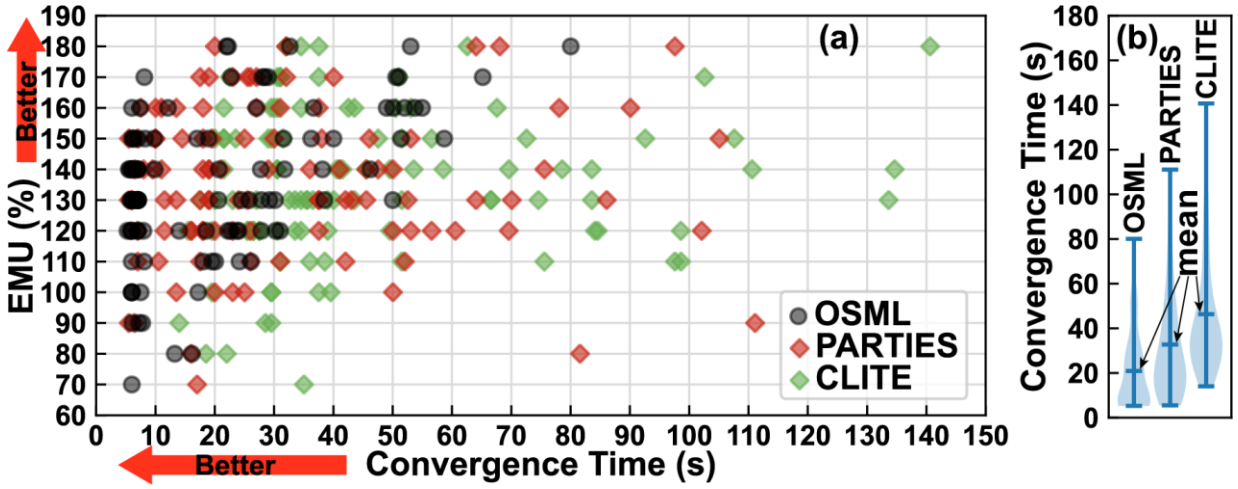
- **Metrics:**
  - Quality of Service (**QoS**, QoS target is the 99th percentile latency of the knee of the latency-RPS curve)
  - Effective Machine Utilization (**EMU**, the max aggregated load of all co-located LC services)

- **Competing approaches:**
  - **PARTIES** [1], a heuristic scheduling approach
  - **CLITE** [2], based on Bayesian optimizer
  - **Unmanaged Allocation**
  - **ORACLE**, the best allocation policy obtained by exhaustive offline sampling

[1] Shuang Chen, Christina Delimitrou, José F. Martínez, “PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services,” in ASPLOS, 2019.
[2] Tirthak Patel, Devesh Tiwari, “CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers,” in HPCA, 2020.

Performance Distribution

- **OSML can achieve the same EMU with a shorter convergence time for a specific load.**
- **OSML converges faster mainly because the start point provided by Model-A is close to OAA.**



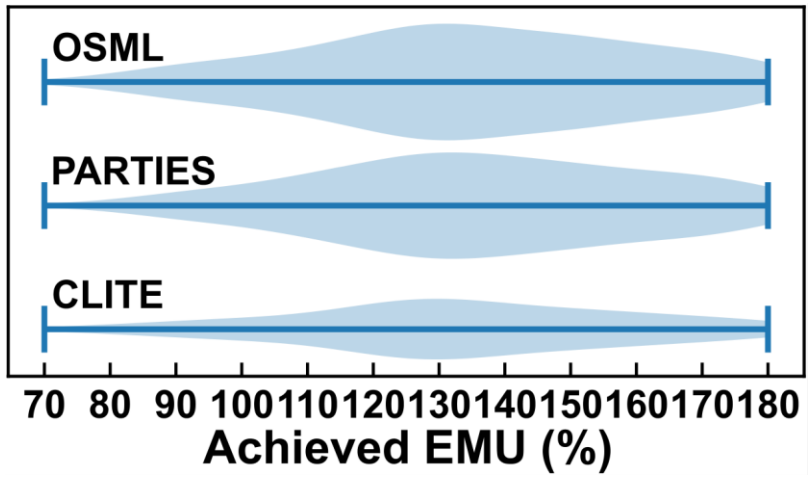
Scheduler	Average Convergence Time
OSML	20.9s
PARTIES	32.7s (1.56×OSML)
CLITE	46.3s (2.22×OSML)

(a) The performance distribution for 104 loads that OSML and other baselines can all converge.
(b) Violin plots of convergence time for loads in (a).

OSML converges 1.56× and 2.22× faster than the baseline approaches.

EMU Distribution

- **OSML works for more loads across different EMUs, particularly in cases where the EMU is high (e.g., 130%~ 180%).**



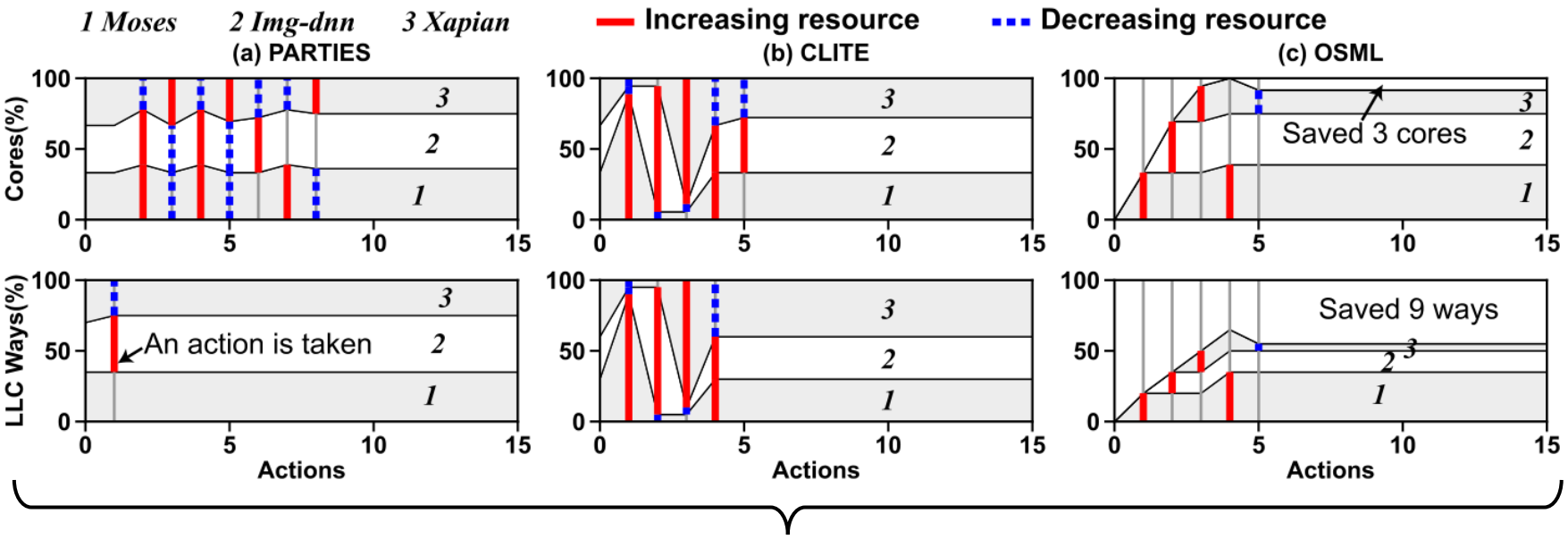
EMU distribution for converged loads among **302 loads.**

Scheduler	Number of Converged loads
OSML	285 (94.4%)
PARTIES	260 (86.1%)
CLITE	148 (49.0%)

OSML, PARTIES, and CLITE works for 94.4%, 86.1%, and 49.0% loads, respectively.

- **Each resource scheduler has its own pros and cons, suited for different cases.**

Resource Usage Comparisons

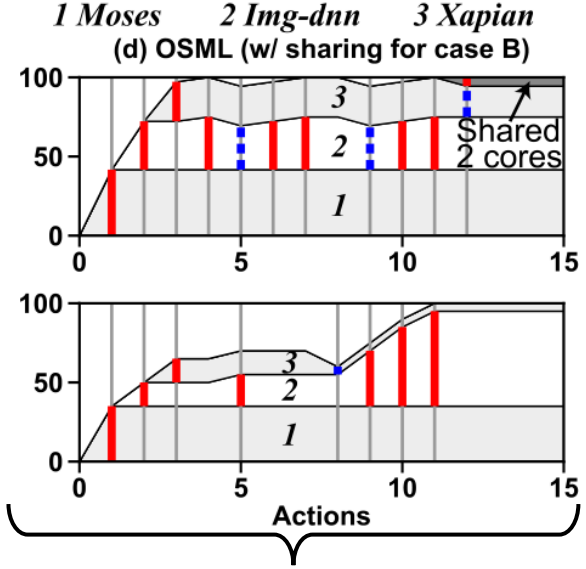


Case A: Moses, Img-dnn, and Xapian with 40%, 60%, and 50% of their maximum loads.

Less Resource Usage: OSML **saves 3 cores and 9 LLC ways**, other baselines use **all resources** on the platform.

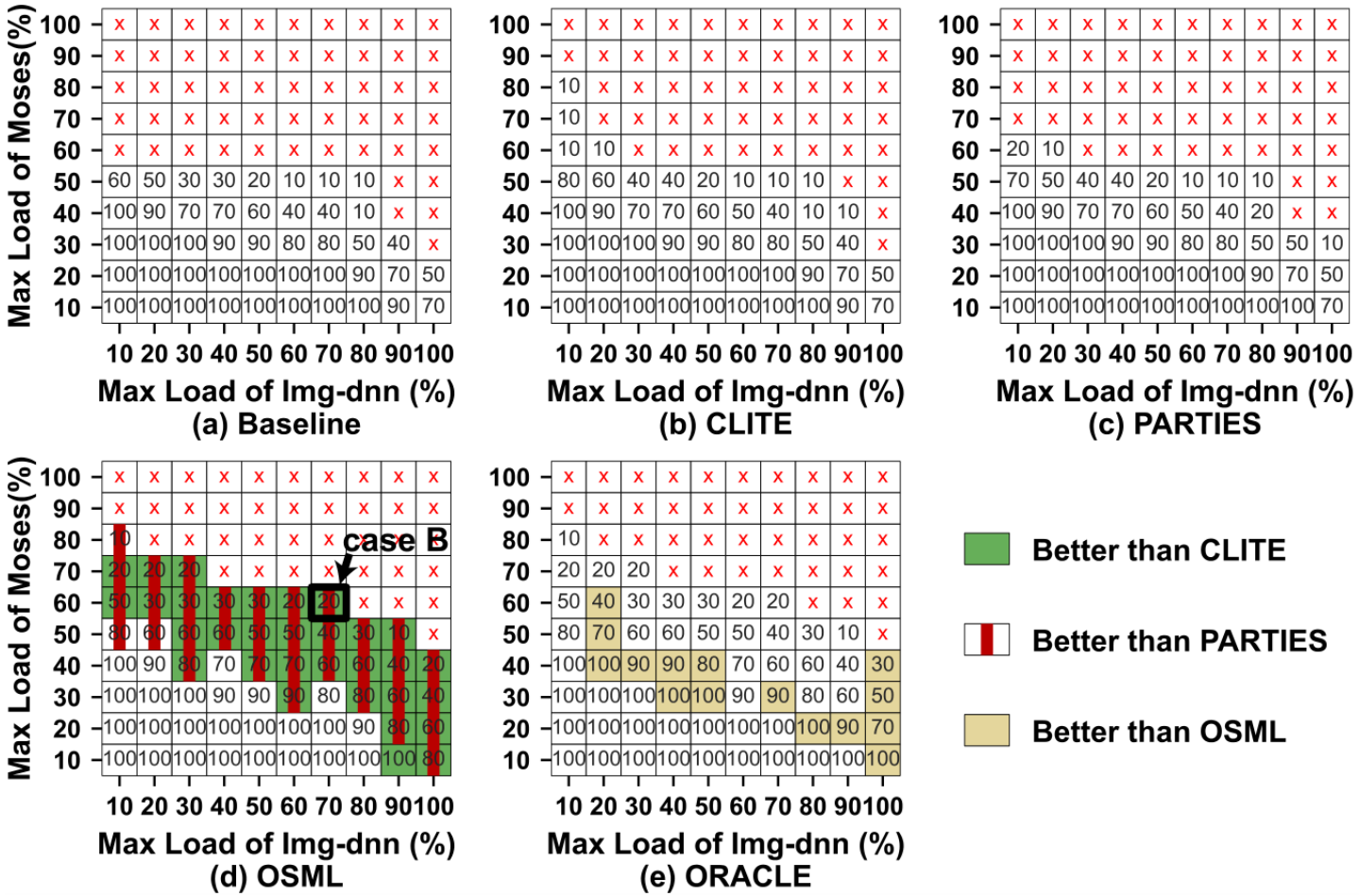
Achieved Loads

- **OSML provides solutions for sharing some cores and LLC ways among LC services, therefore supporting higher loads.**



Case B: Moses, Img-dnn, and Xapian with 60%, 70%, and 20% of their maximum loads.

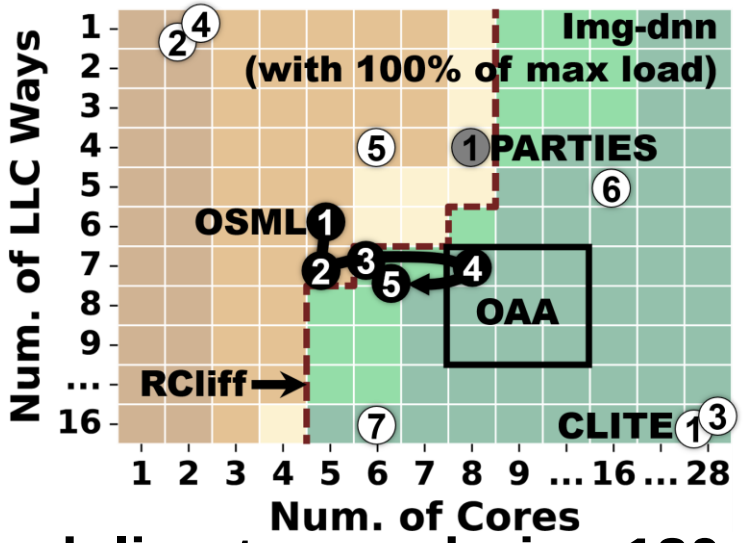
Higher Loads: OSML converges by **sharing resources**. The baselines can not converge for this load.



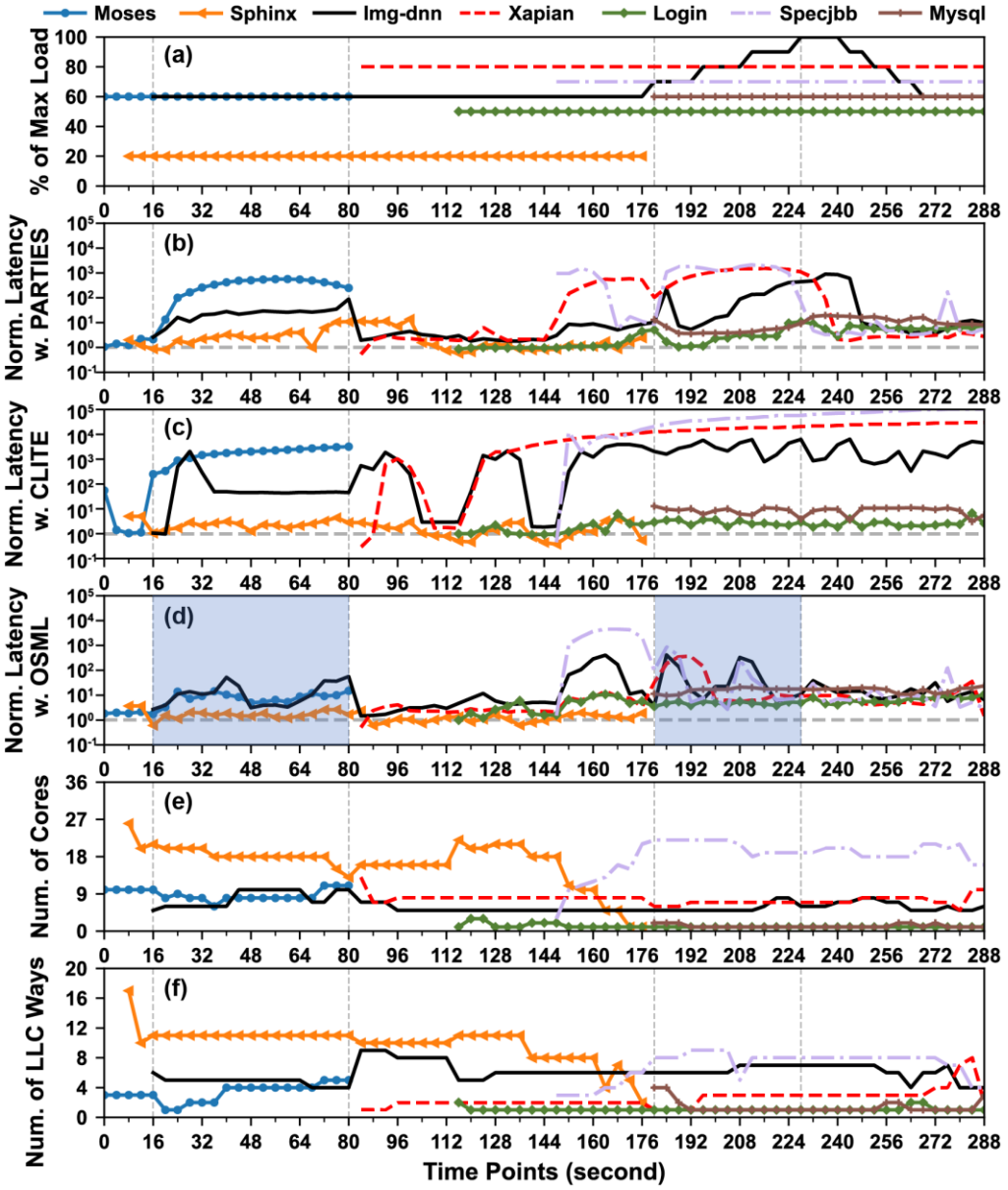
Workload: Moses, Img-dnn, and Xapian

Performance for Workload Churn

- **16s~80s: New service arrives**
  OSML provides better scheduling solutions at time point 48 for all three services.
- **180s~228s: Img-dnn's load increases**
  OSML meets Img-dnn's changing demand with Model-C.



Scheduling traces during 180s~228s



Generalization

- **Performance for all unseen applications**
  - OSML converges **1.38×~1.50×** faster than baselines.

- **Model errors**
  - Errors are not significant.
  - Model-A/B outputs approximate OAA, Model-C schedules and learns online.

ML	Outputs	Error		Errors for unseen LC services		Err on new platforms (TL)		MSE	Over-heads
		Core	LLC	Core	LLC	Core	LLC		
A	RCliff	0.589	0.288	1.266	0.198	2.142	0.542	0.0025	0.20s
	OAA	0.580	0.360	1.276	0.191	2.004	0.865		
A'	RCliff	1.072	0.815	3.403	1.835	0.772	0.411	0.0135	0.20s
	OAA	1.072	0.814	3.404	1.835	0.790	0.413		
B	B-Points	0.612	0.053	4.012	0.167	2.320	0.969	0.0012	0.18s
	B-Points,Core dominated	0.314	0.048	3.434	0.937	2.250	0.815		
	B-Points,Cache dominated	0.093	0.462	0.789	0.783	1.868	1.519		
B'	QoS reduction	7.87%		8.33%		11.28%		0.0035	0.19s
C	Scheduling actions	0.908	0.782	0.844	0.841	1.390	1.801	0.7051	0.20s

- **OSML is a long-term project open to the community; we continue adding new traces collected from new applications and new servers to the data set for enhancing models' performance for new cases.**

Conclusion

- **OSML is an ML-based resource scheduler for co-located LC services.**
- **OSML uses multiple ML models cooperatively in a pipe-lined way.**
- **Leveraging ML could have an immense potential for OS design.**
- **In a world where co-location and sharing are a fundamental reality, our solution should grow in importance and benefits our community.**

Thank You!

Lei Liu[1], Xinglei Dou[2], Yuetao Chen[2]

[1]Beihang University; [2]ICT,CAS; Sys-Inventor Lab

https://liulei-sys-inventor.github.io/