The Elasticity and Plasticity in Semi-Containerized Co-locating Cloud Workload: a view from Alibaba Trace

Qixiao Liu* and Zhibin Yu

Shenzhen Institute of Advanced Technology

Chinese Academy of Science

@SoCC 2018, Carlsbad, CA, U.S.A

Introduction

- Challenges in the cloud computing
  - – Low resource utilization
  - – Tail latency
  - – IRU-QoS dilemma
  - – Task scheduling, resource management, programming diagram, etc.
- Traces from industrial production environment
  - – The Google trace released in **2011**
    - 12.7m machines, 670k jobs (mixed workload), 29 days.
  - – Alibaba released in **2017**
    - 1.3k machines, 23k jobs (also mixed workload), in 1 day.

Fraction External

Google trace vs. Alibaba trace

- Google trace
  1. Server heterogeneity
  2. Priority information
  3. Server failure
  4. Mixed workload (production and non-production), but are 'equal' as jobs

- Alibaba trace
  1. All servers are equipped with 64 CPUs, >99% of servers: same memory and disk capacities.
  2. No priority information
  3. Negligible server failures
  4. Online services and batch jobs are traced separately

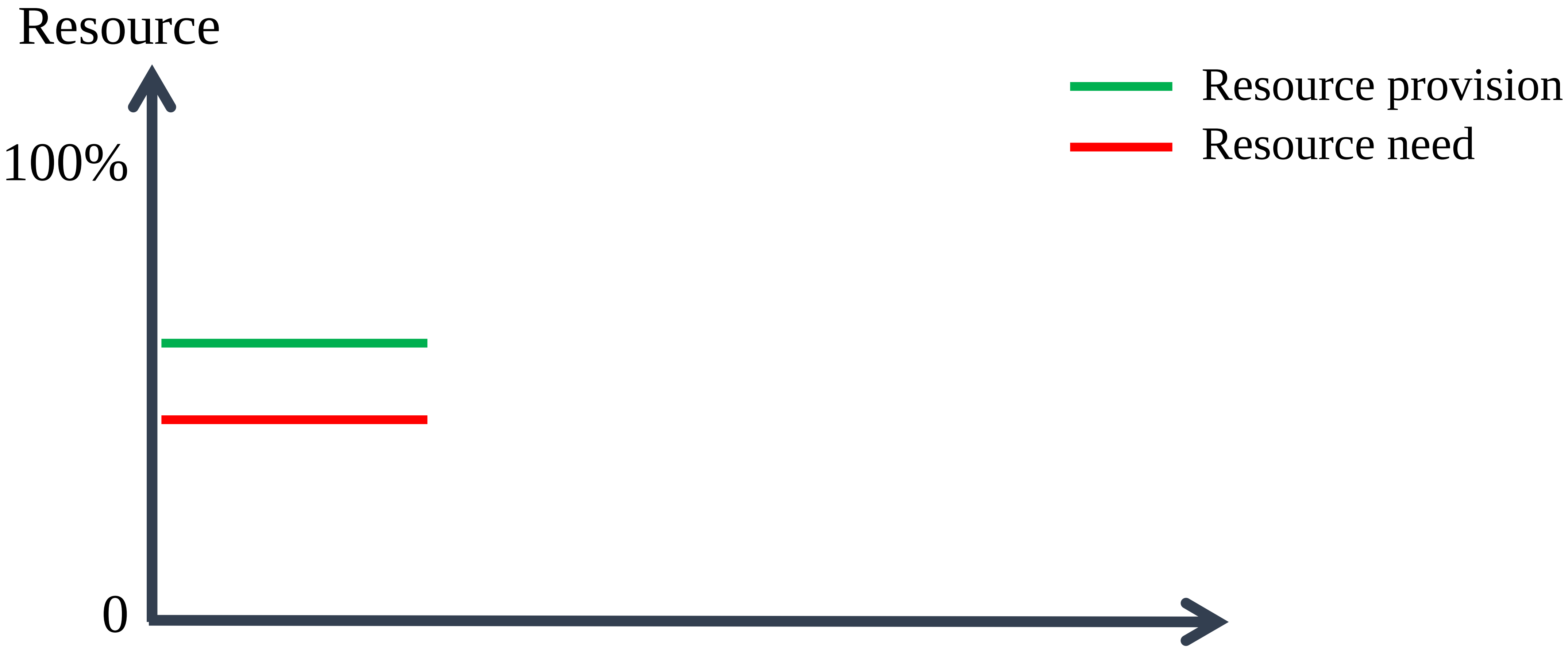
More elaborative views into the co-location

- **Elasticity**
  - resist a distorting influence and to return to its original size and shape when that influence or force is removed*

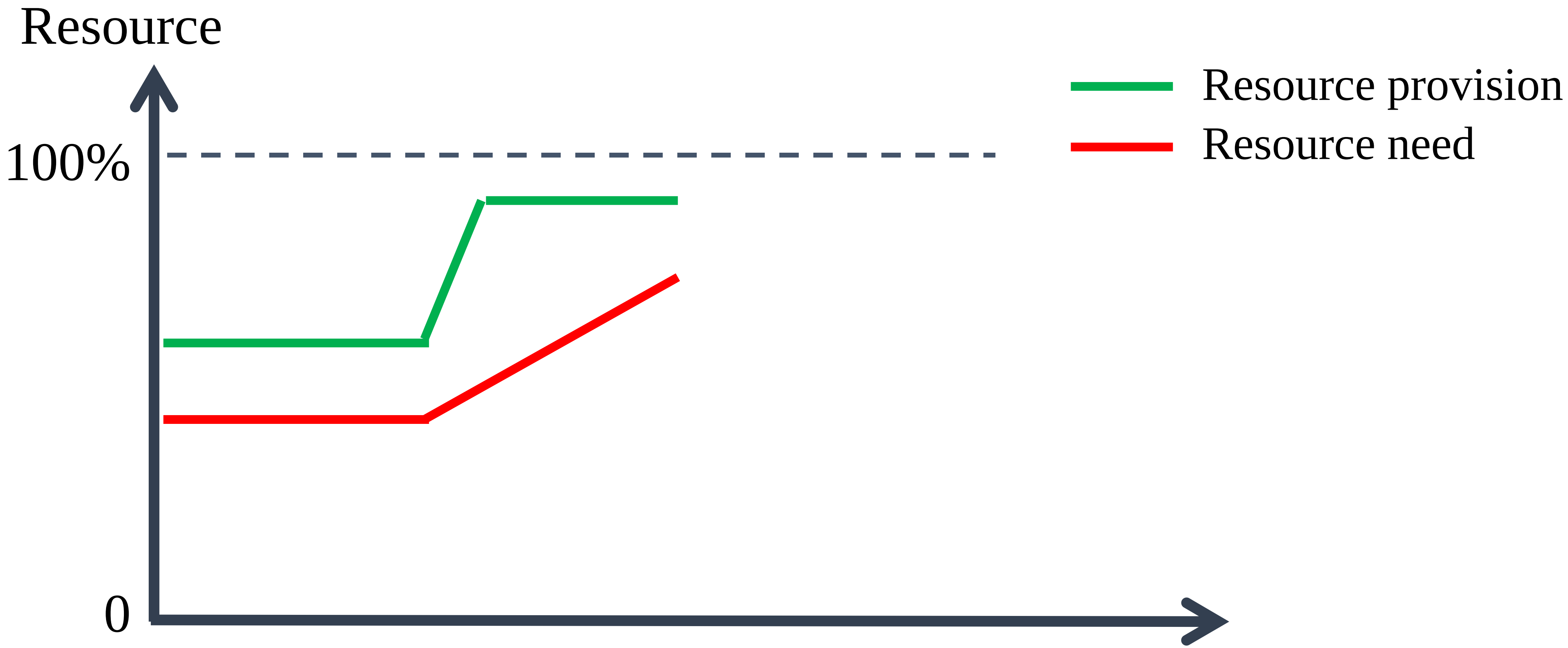
- **Plasticity**
  - non-reversible changes of shape in response to applied forces*

* Elasticity, plasticity (physics), wikipedia

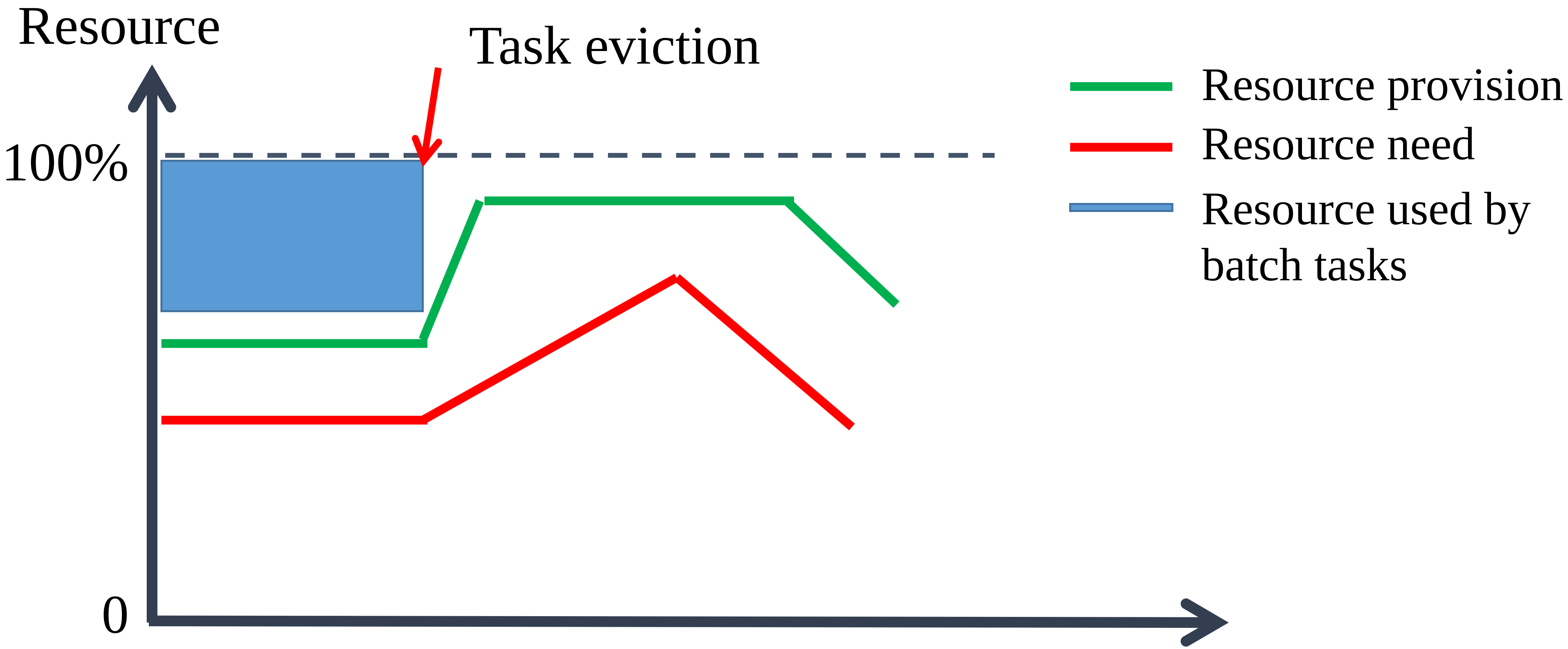
Elastic computing



Elastic computing

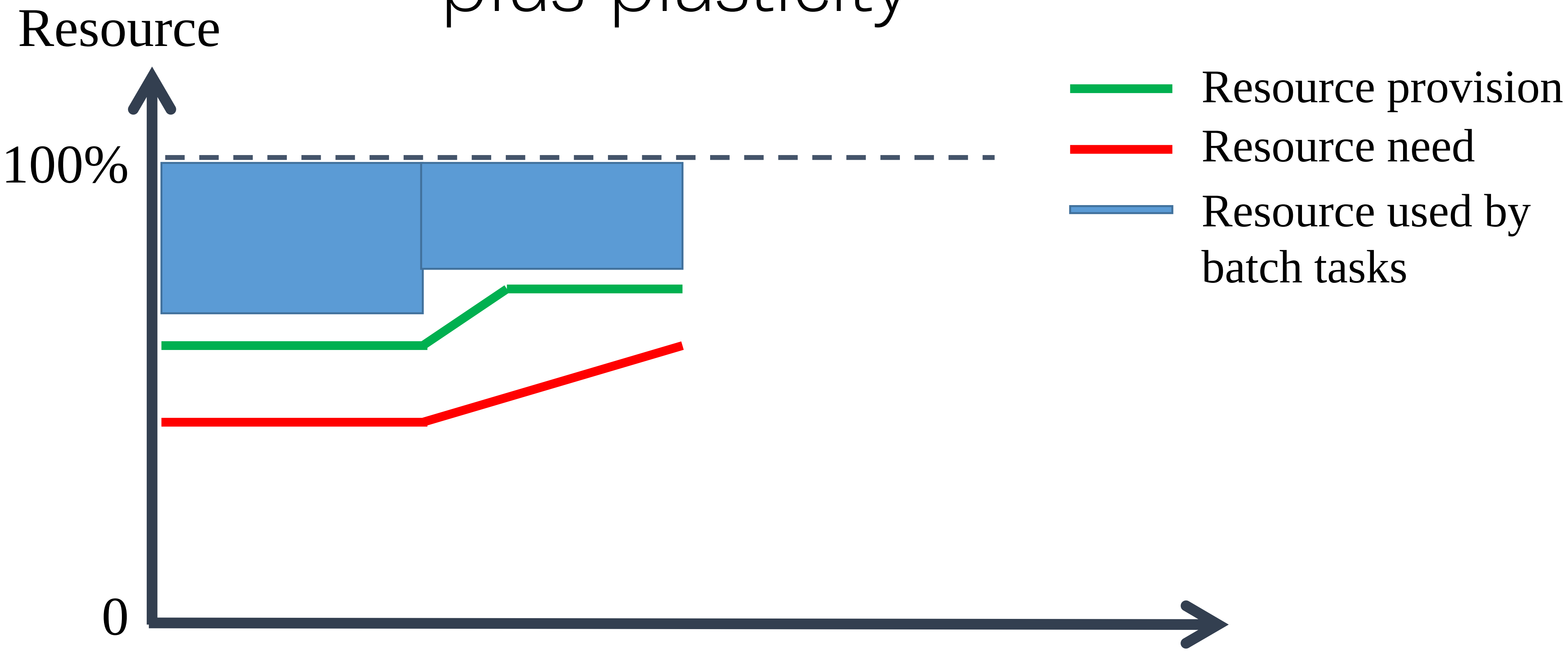


Elastic computing in co-location



Resource utilization ↑ Batch job performance ?

Ideal elastic computing in co-location plus plasticity

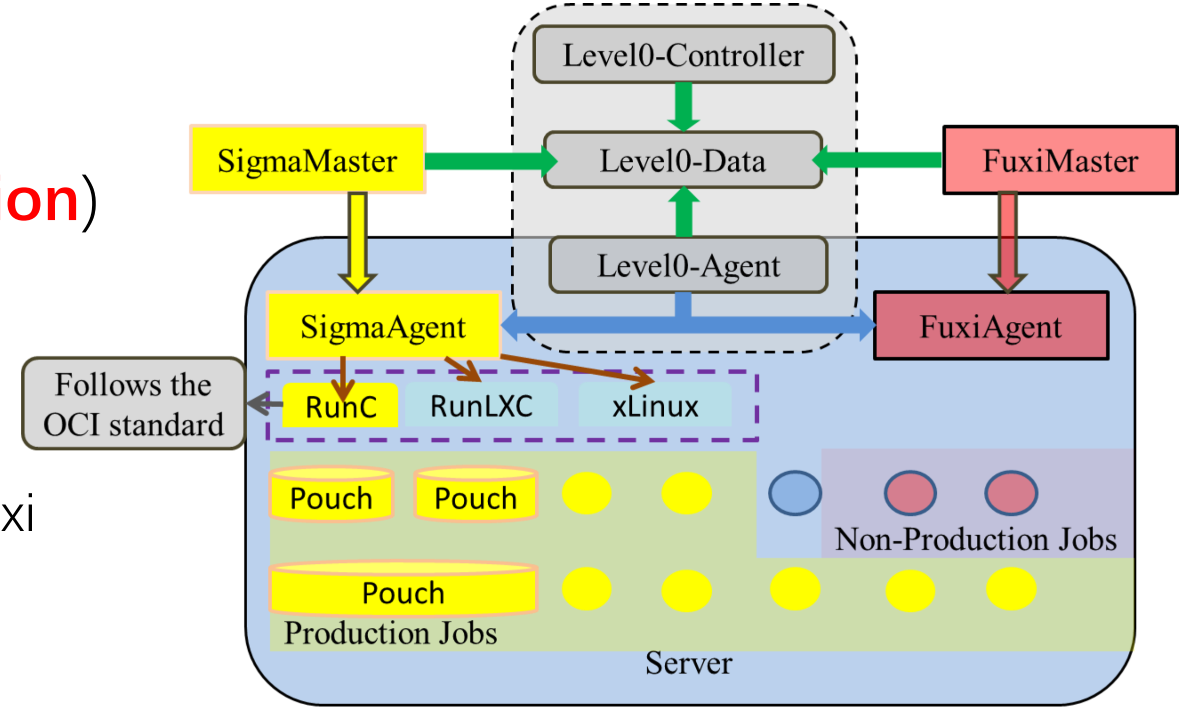


Outline

- Trace overview
- Shape the workload
- Statistics analysis of containers and batch jobs
- Co-location analysis
- Discussion
- Conclusion

Alibaba cluster management architecture

- Mixed workload
  - Online services
  - Batch jobs
- Mixed entities (**semi-containerization**)
  - Container
  - Tasks
- Mixed architecture
  - **Concurrent schedulers**: Sigma and Fuxi
  - Level0-controller



technical

– **Novelty?**

legacy

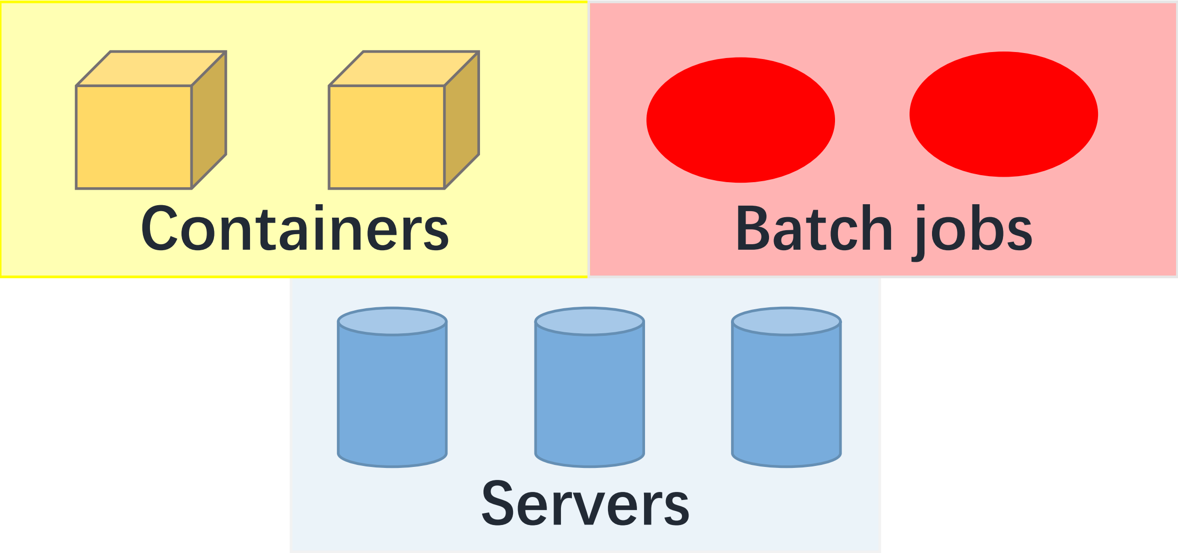
It is in production.

Trace structure

Static information:
ID, Machine ID;
Req CPU/mem/disk;
Create time;
Allocated CPU ID.

Runtime statistics:
Container ID;
Sample point;
Used CPU/mem/disk;
CPU load, CPI,
mem miss.

Sampled every 5min



Static information:
Jobid/taskid;
Create/end time;
Req CPU/memory;
Status, #instance.

Runtime statistics:
Jobid/taskid;
Instance start/end time;
Status;
Sequence;
Used CPU/memory

Execution based

Static information:
Machine ID;
Config. CPU/mem/Disk;
Event and time.

Runtime statistics:
Machine ID;
Sample point;
Used CPU/mem/disk;
CPU load, CPI,

Sampled every 5min

Outline

- Trace overview
- Shape the workload
- Statistics analysis of containers and batch jobs
- Co-location analysis
- Discussion
- Conclusion

Container

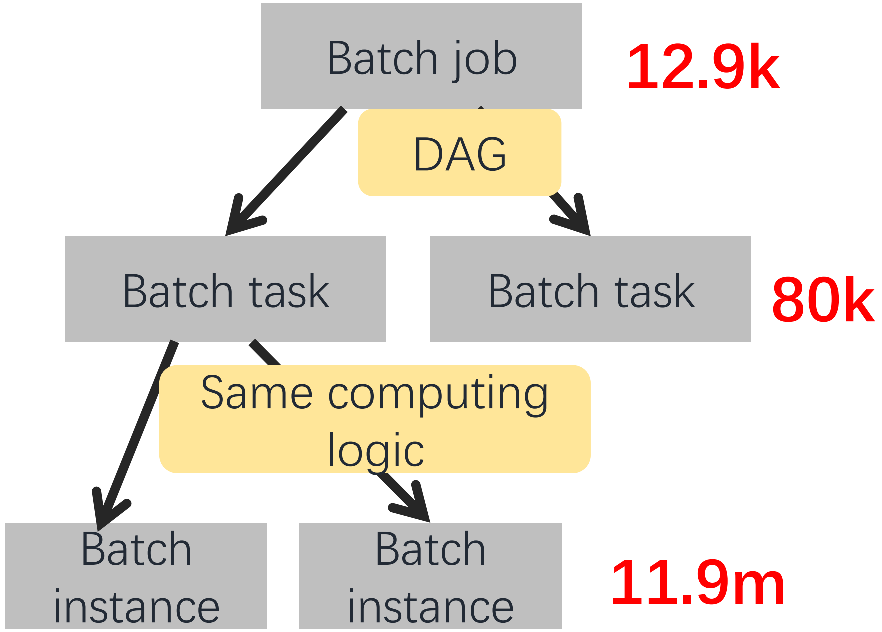
- 11089 containers, each runs one online service, for 24 hours
- Container requests CPUs, memory and disk
  - Req. #CPU: 1, 4, 6, 8, 16
  - Memory capacity (normalized): 0.002 to 0.318
  - Disk capacity (normalized): 3e-11 to 0.113
  - **25** <CPU, memory, disk> patterns for all containers, **19** are valid.
- **Requested resource over server capacity** (ROC):

*(Resource_req/Server capacity)*100%*

	CPU	Memory	Disk
ROC	9.5%	10.9%	4.9%
ROC SD	4.4%	8.8%	2.1%

Batch job

- Batch job structure: job, task, instance
- Job->task: DAG
- Task->instance: same computing logic, resource request



Batch job

- Batch task requests CPU and memory
  - – #CPU: 0.45 to 8 (14 values in total, 0.05 basic unit)
  - – Memory: 0.0027 to 0.1273 (750 values)
  - – 989 <CPU, memory> patterns in 80k tasks

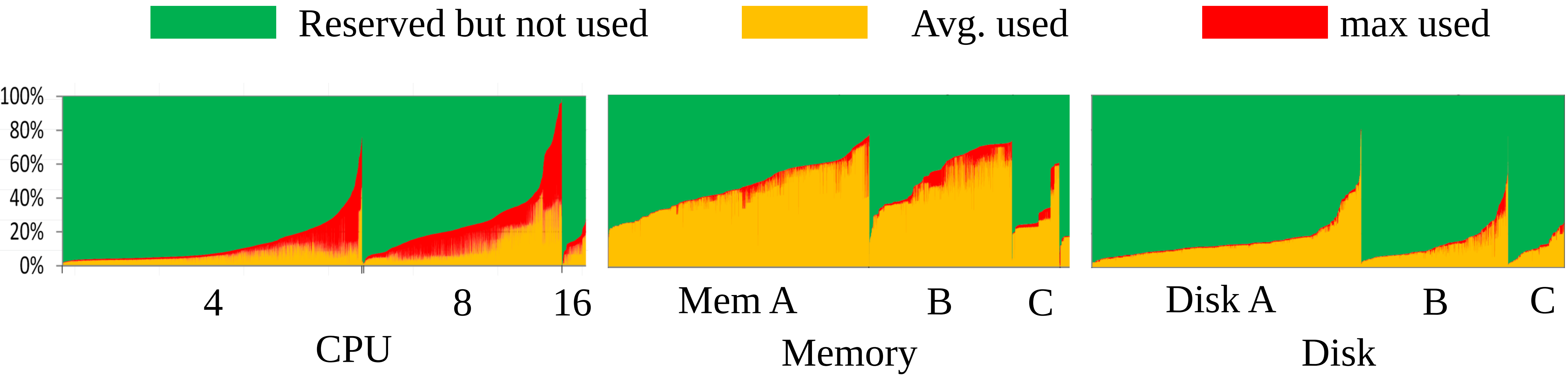
	CPU	Memory
ROC	0.8%	0.9%
ROC SD	0.5%	0.7%

- Batch instance status:
  - – Failed, interrupted, ready, running, terminated, wait
  - – Failed/interrupted rate are **1.5%**
    - Google trace: ‘half submissions are resubmissions’

Outline

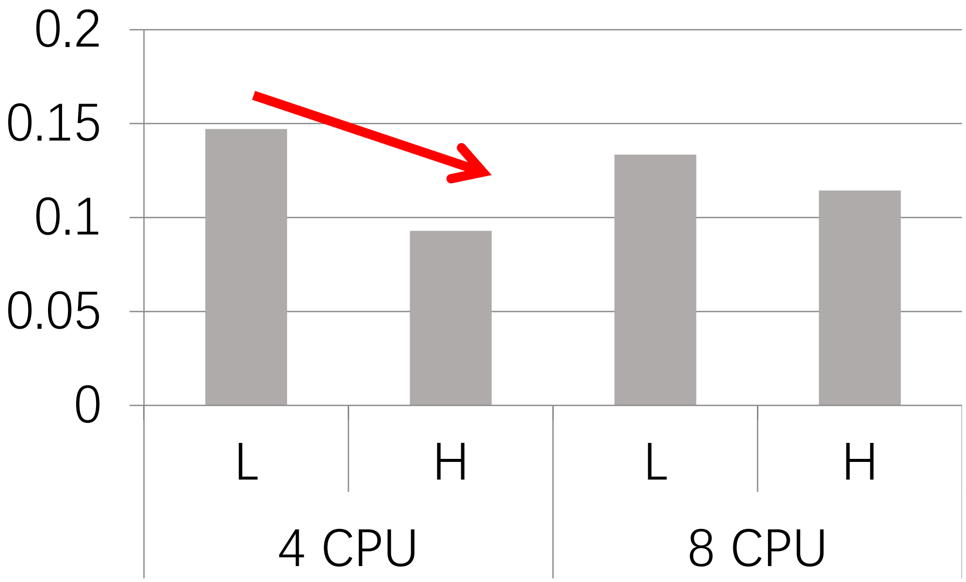
- Trace overview
- Shape the workload
- Runtime statistics analysis of containers and batch jobs
- Co-location analysis
- Discussion
- Conclusion

Container resource utilization



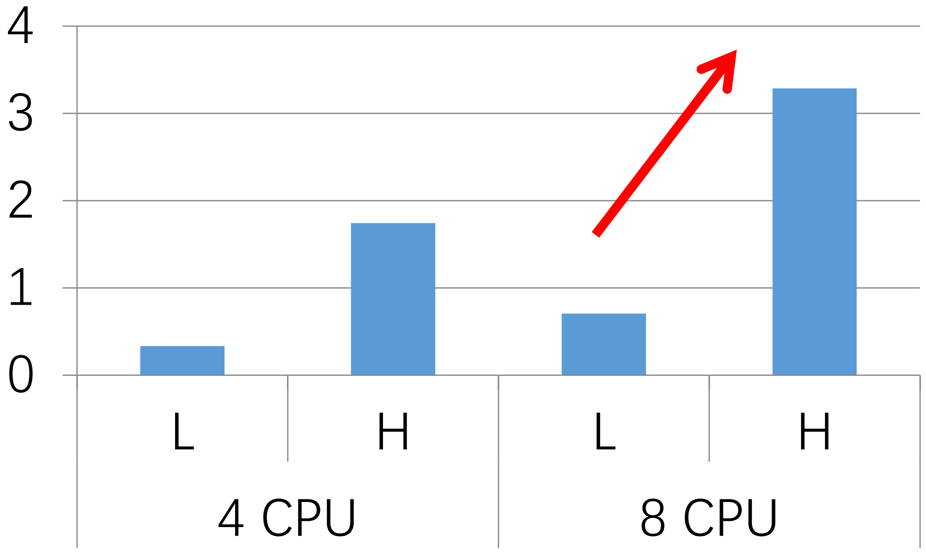
- – **Resource overprovisioning**
- – **Max vs. average** resource used
  - steady memory and disk utilization, but CPU varies significantly

Container performance



CPI

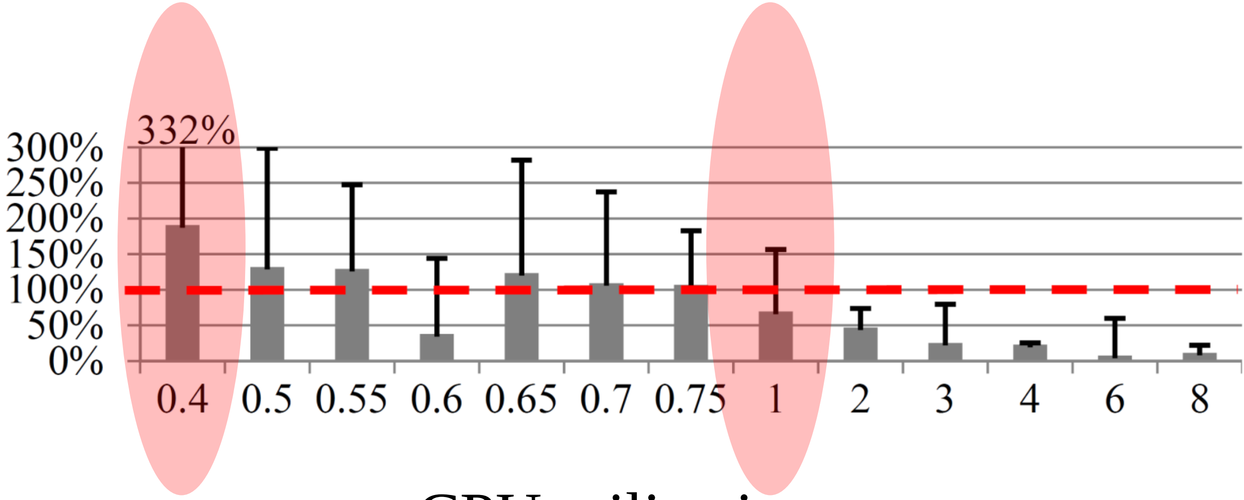
L/H: low /high CPU utilization



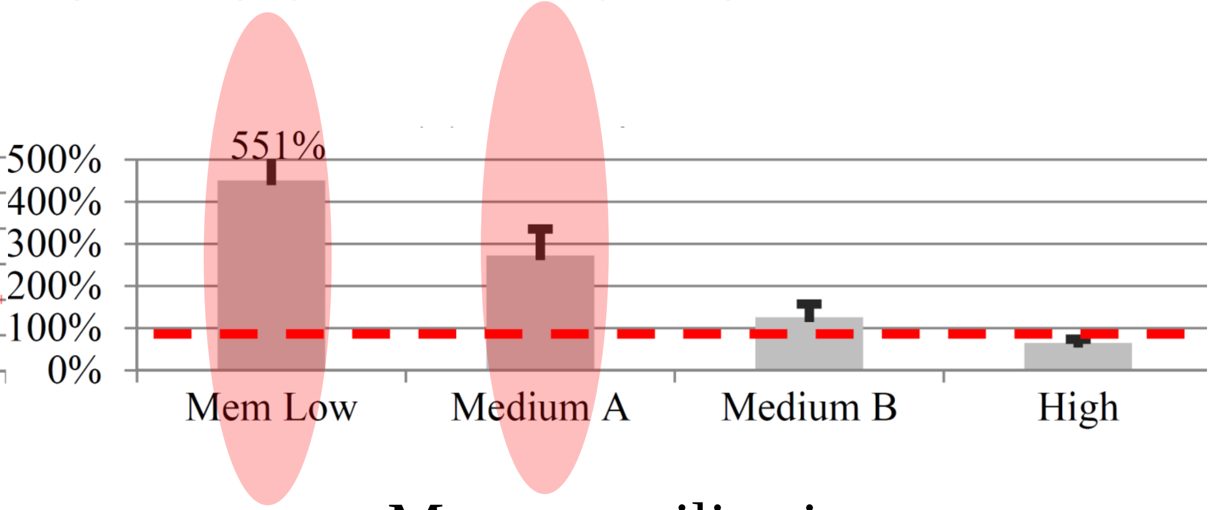
CPU load

- Containers are guaranteed resources when load rises;
- Higher load increases resource utilization, but not hurting the performance.

Batch instance resource utilization



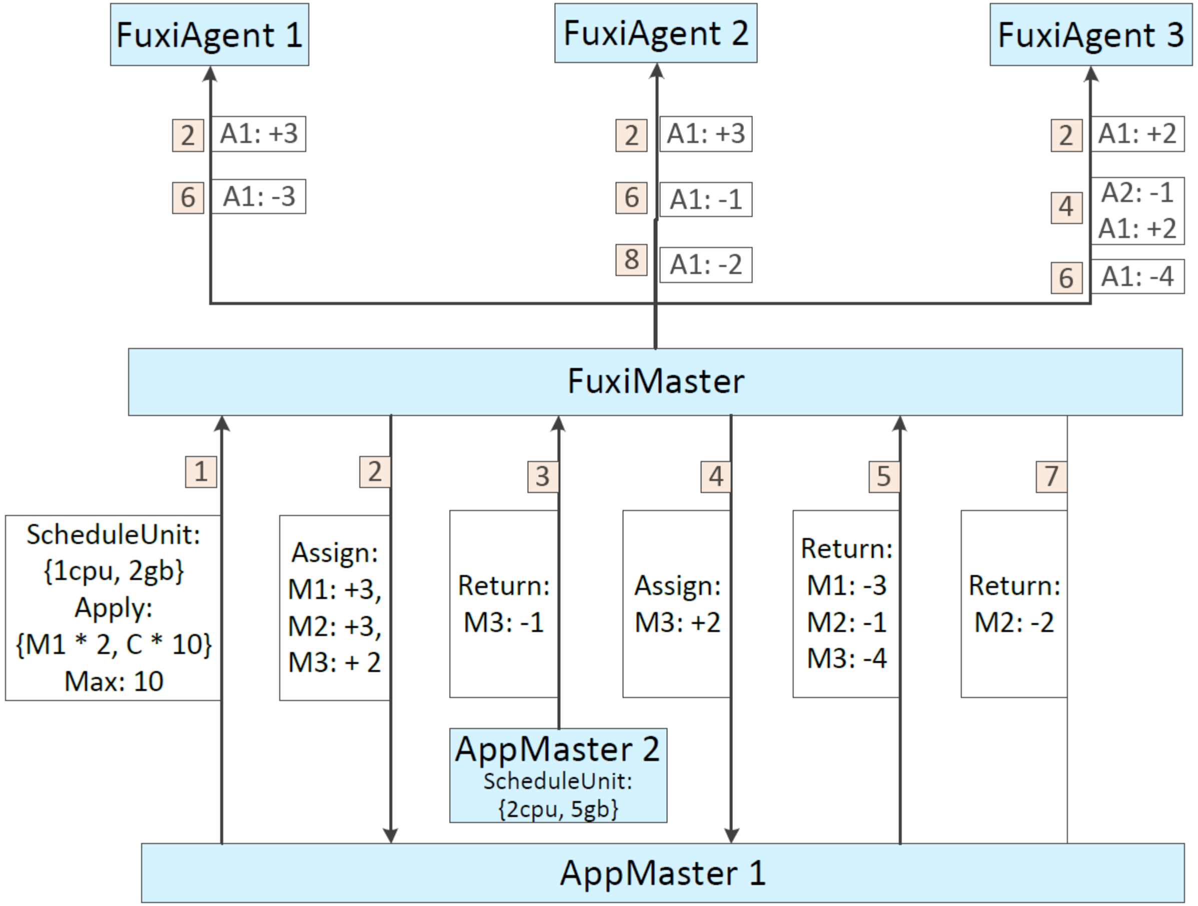
CPU utilization



Memory utilization

- **Resource overcommit**, the amount of its actual used resources is greater than that it requested at submission.
- Both CPU and memory overcommit.

Incremental resource allocation in Fuxi



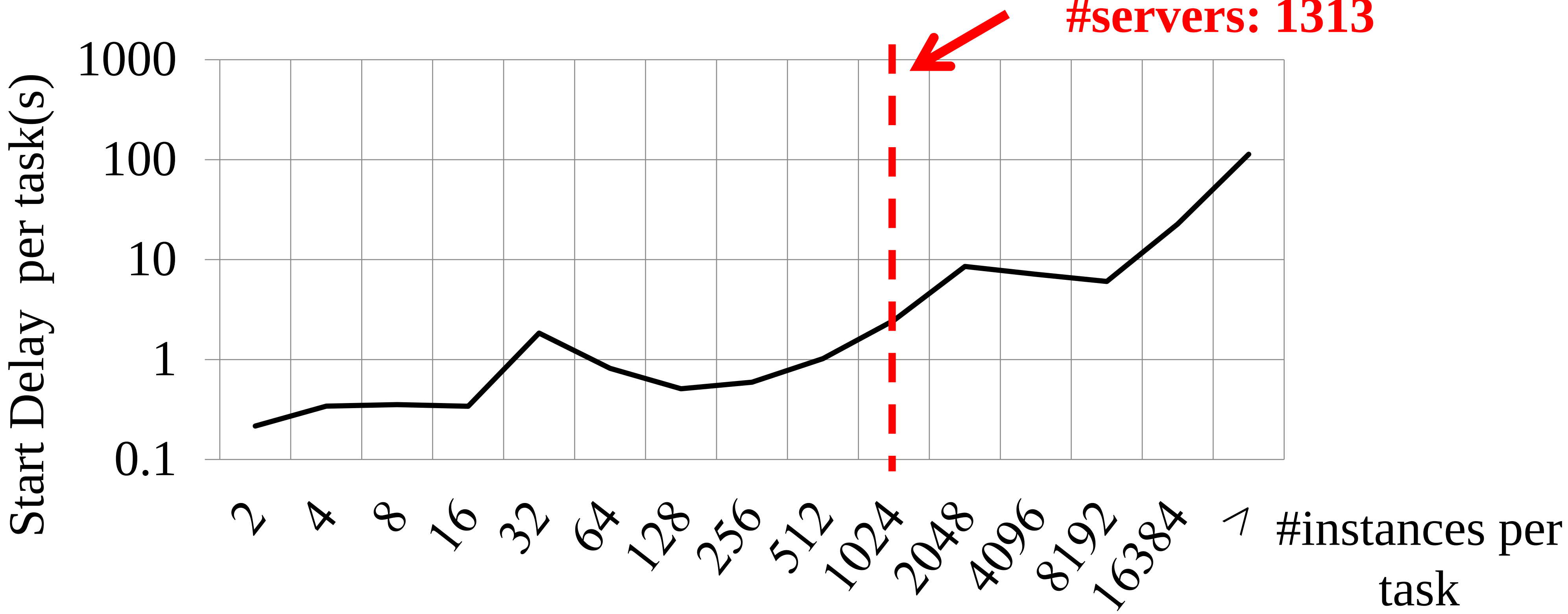
- FUXI, VLDB 2014

Incremental resource allocation in Fuxi

- Local queue in node
- Resource request:
  - Initial resource request (low)
  - Actual (peak) request (high)

- FUXI, VLDB 2014
- Start to run a batch instance with its initial resource request, increase its allocation when more resources become available.
- Batch instance with lower resource request has a better chance get to run

Cluster wide resource allocation efficiency

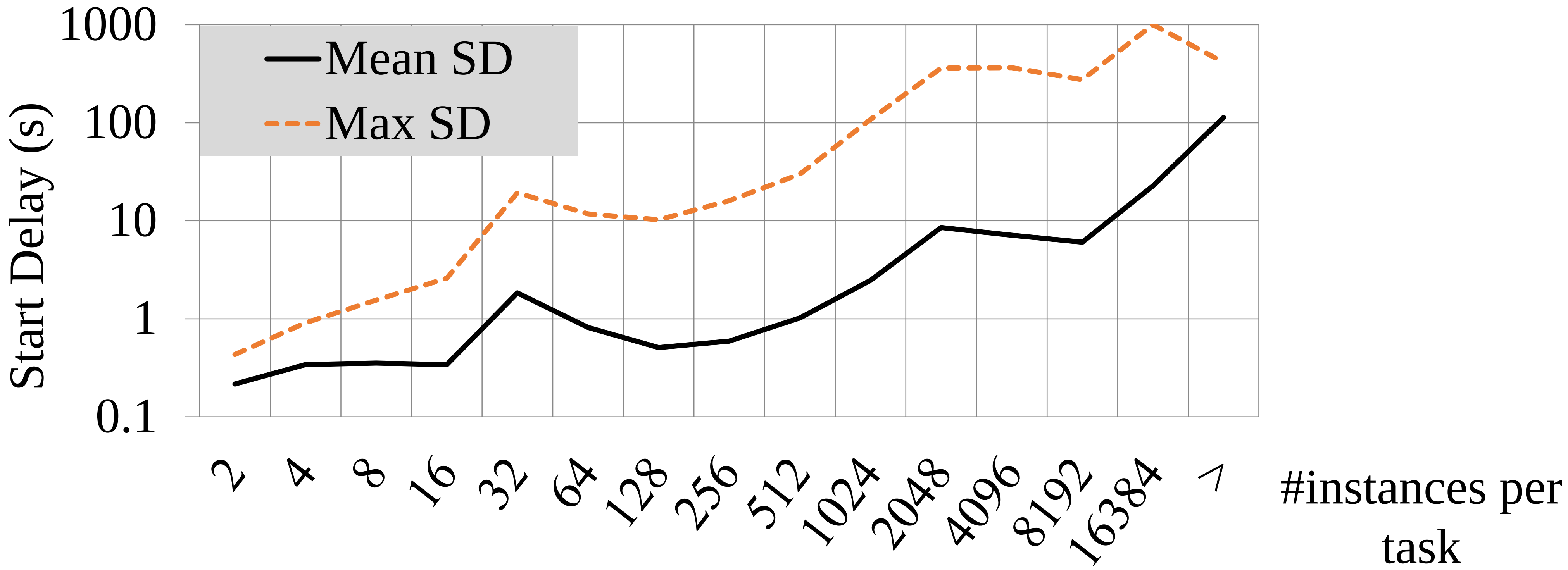


Instances from the same task, get scheduled at the same time

Start delay:

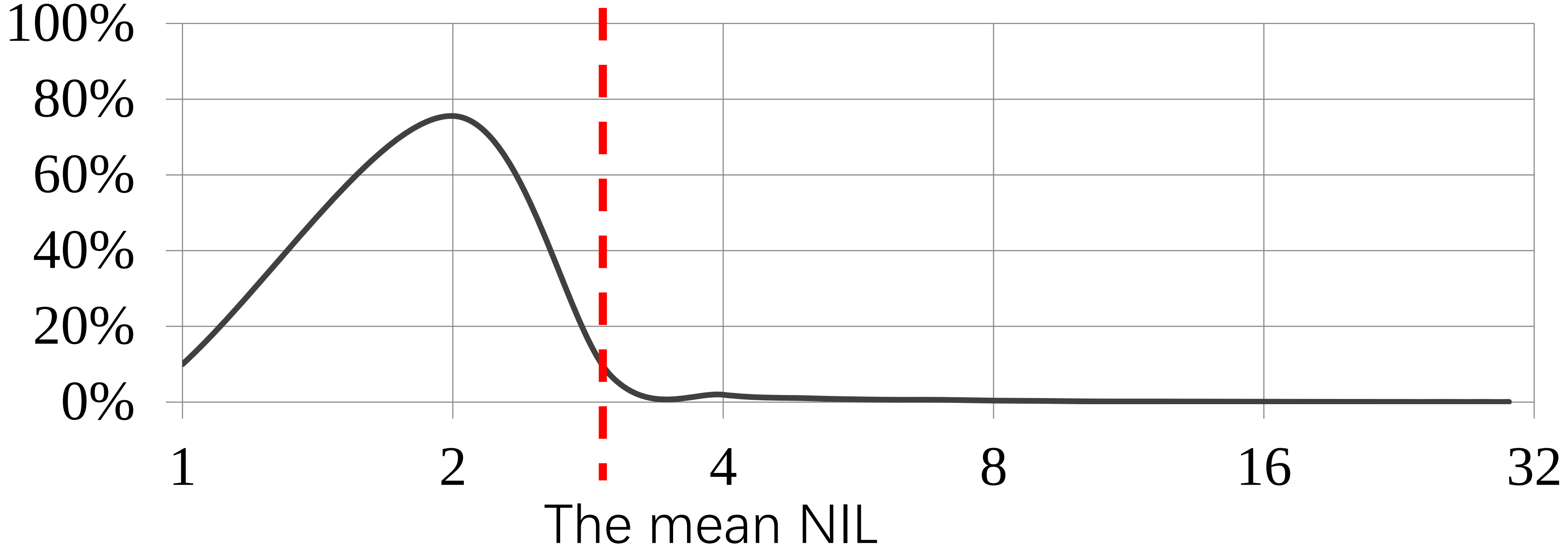
$$\mathrm{SD} = start\_time_i - Ref\_start\_time^T$$

Cluster wide resource allocation efficiency



The latest one most likely delay the result delivery of the task/job

Cluster-wide batch instance performance

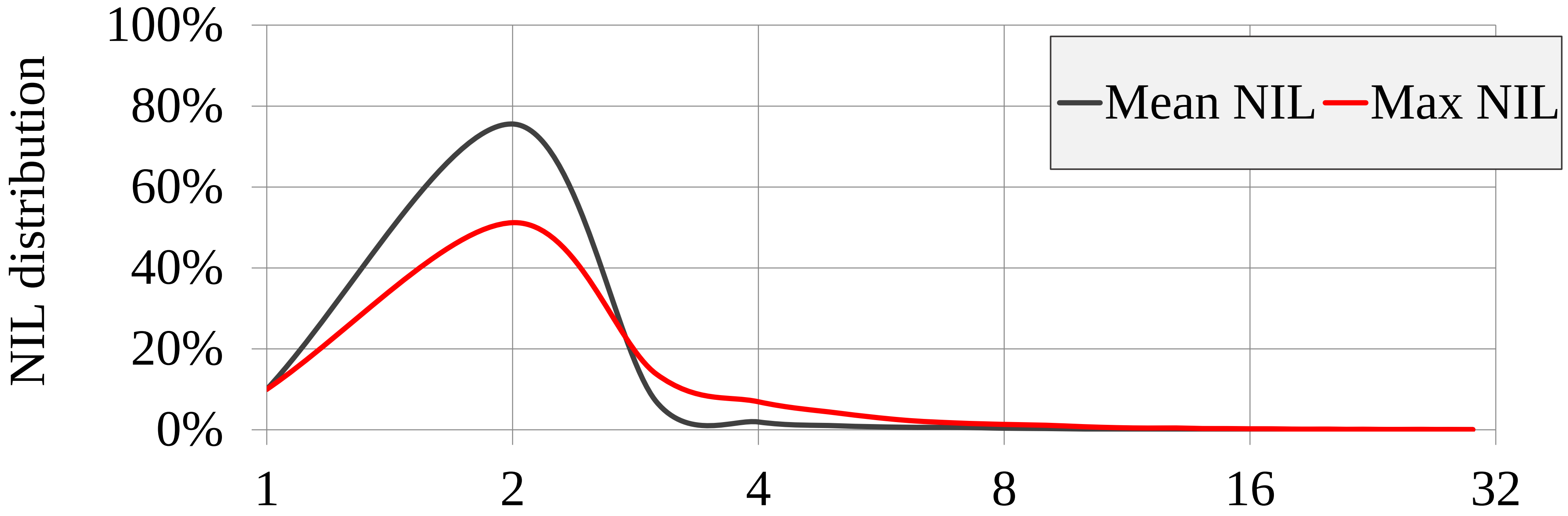


Normalized instance latency:

$$\mathrm{NIL} = Execution\_time_i / Ref\_time^T$$

- most tasks have their avg NIL below 3.

Cluster-wide batch instance performance

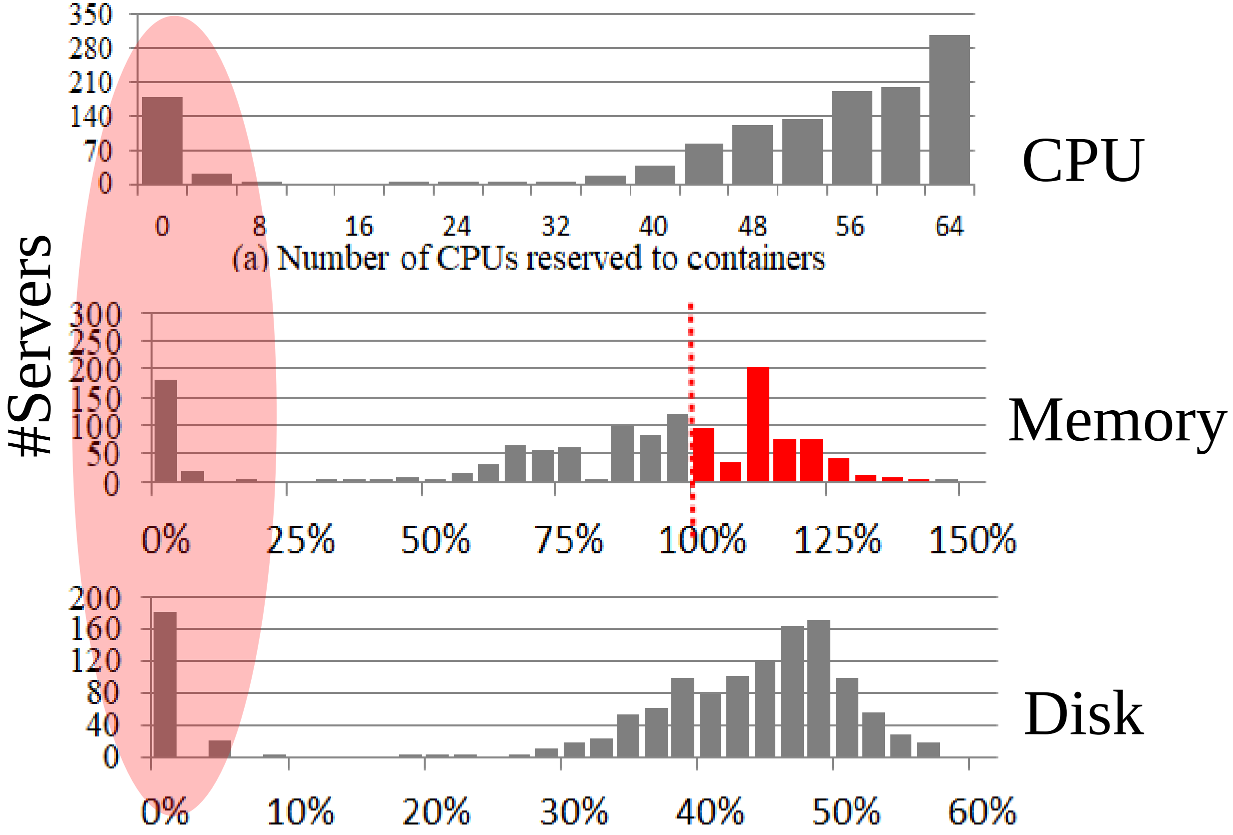


- the max NIL of few tasks deviate from the average.

Outline

- Trace overview
- Shape the workload
- Statistics analysis of containers and batch jobs
- Co-location analysis
- Discussion
- Conclusion

Container deployment

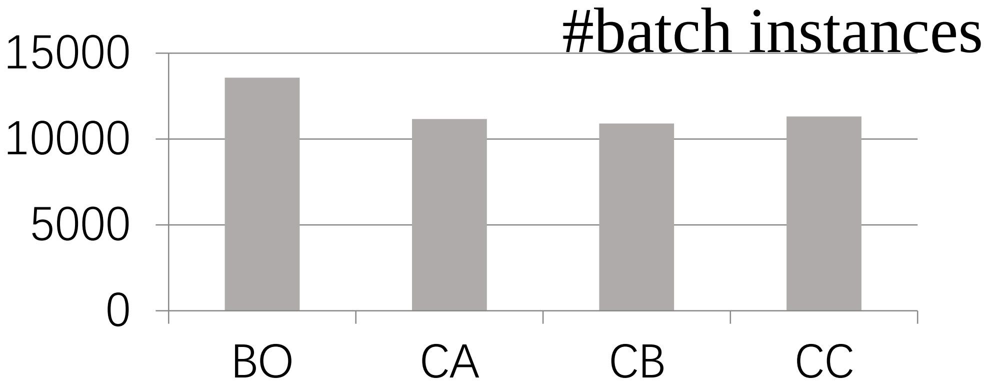


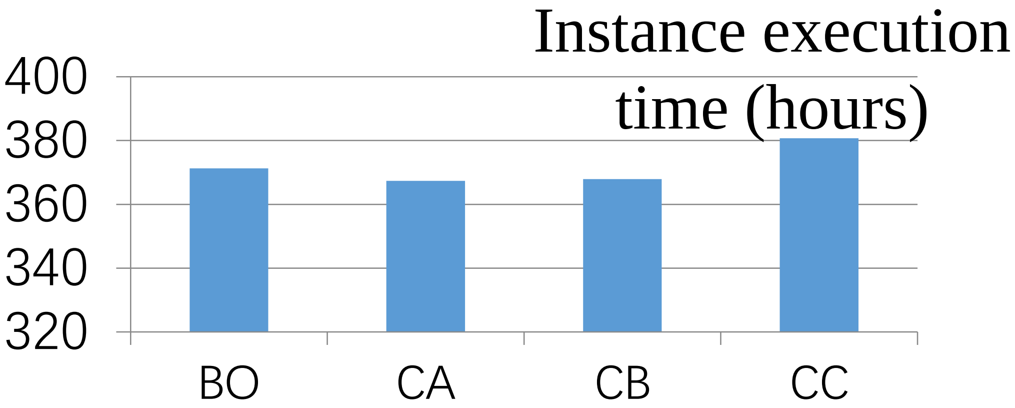
Containers reserve resources

- 0~64 CPUs;
- 0~150% memory; overbooking
- 0~60% disk.

- Containers are deployed using different policies;
- CPU remains the main constraint.

Batch instance scheduling in the co-location



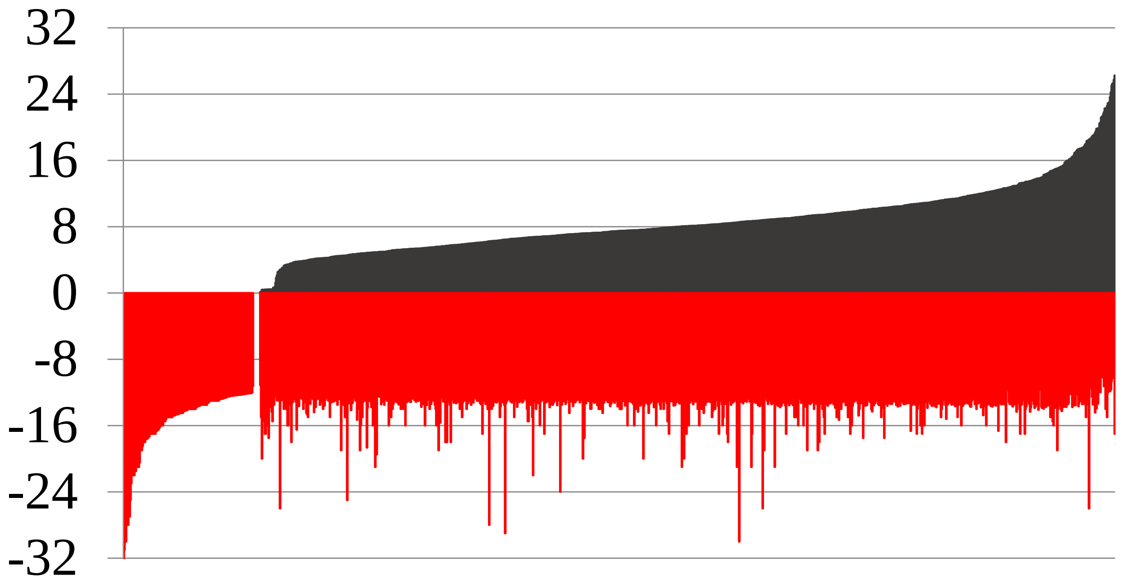


BO: Batch instance only servers
CA, CB, CC: low, medium, full resources reserved by containers

No obvious difference to schedule a batch instance in the cluster:

- Similar **accumulated instance execution time** on all servers, although BO has more batch instances running.

Resource allocate to batch instances

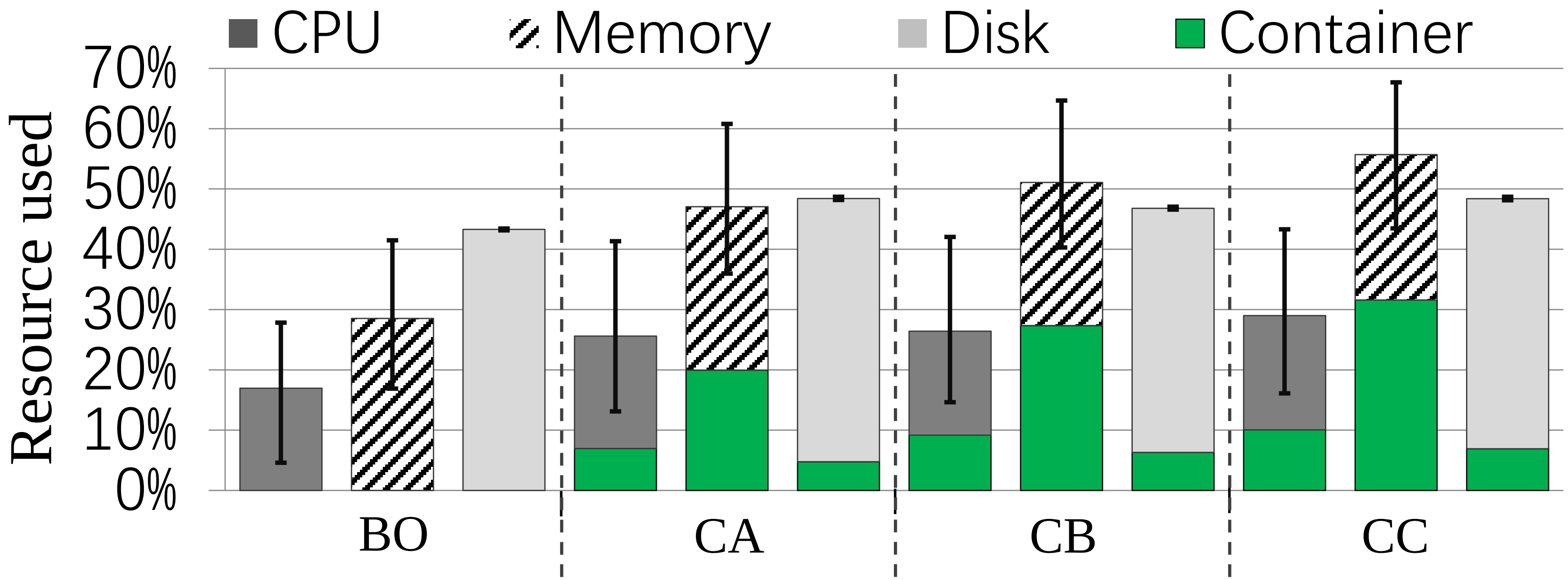


Max #CPU used by containers on each server (sampled every 5 min)

Max #CPU used by batch a instance on each server (max CPUs during execution)

Max #CPU allowed for batch instances to use on servers does not depend on the #CPU used by containers.

Resource utilization in the cluster



Outline

- Trace overview
- Shape the workload
- Statistics analysis of containers and batch jobs
- Co-location analysis
- Discussion
- Conclusion

Discussion

- Elasticity
  - Resource **overprovisioning** (containers).
  - Resource **overcommitment** (batch instances).
  - Resource **overbooking**.
- Plasticity
  - Very low task eviction rate in the cluster (1.5%).
  - Accumulated batch instance execution time on most servers is similar.
  - SD increases radically when a task owns more than 1000 instances (there are 1313 servers).
  - No obvious difference between the maximum allowed #CPU for batch instance to use on most servers

Outline

- Trace overview
- Shape the workload
- Statistics analysis of containers and batch jobs
- Co-location analysis
- Discussion
- Conclusion

Conclusion

- Alibaba presents a trace, using semi-containerized cluster management
- Concurrent traces for online services and batch jobs allow more elaborative characterization of the mixed workload
- Elasticity and plasticity in the cluster management promoted the batch job performance.

Thanks for attention!! Also @poster