Scaling Large Production Clusters with Partitioned Synchronization

Yihui Feng[*], Zhi Liu[*†], Yunjian Zhao[†], Tatiana Jin[†], Yidi Wu[†],
Yang Zhang, James Cheng[†] , Chao Li, Tao Guan
Alibaba Group [†]*The Chinese University of Hong Kong*

[*]Co-first-authors ordered alphabetically.
[†]This work was done when the authors were visiting Alibaba.

Scale of computer clusters

100k

machines

Billions

of tasks

Scale of computer clusters

100k

machines

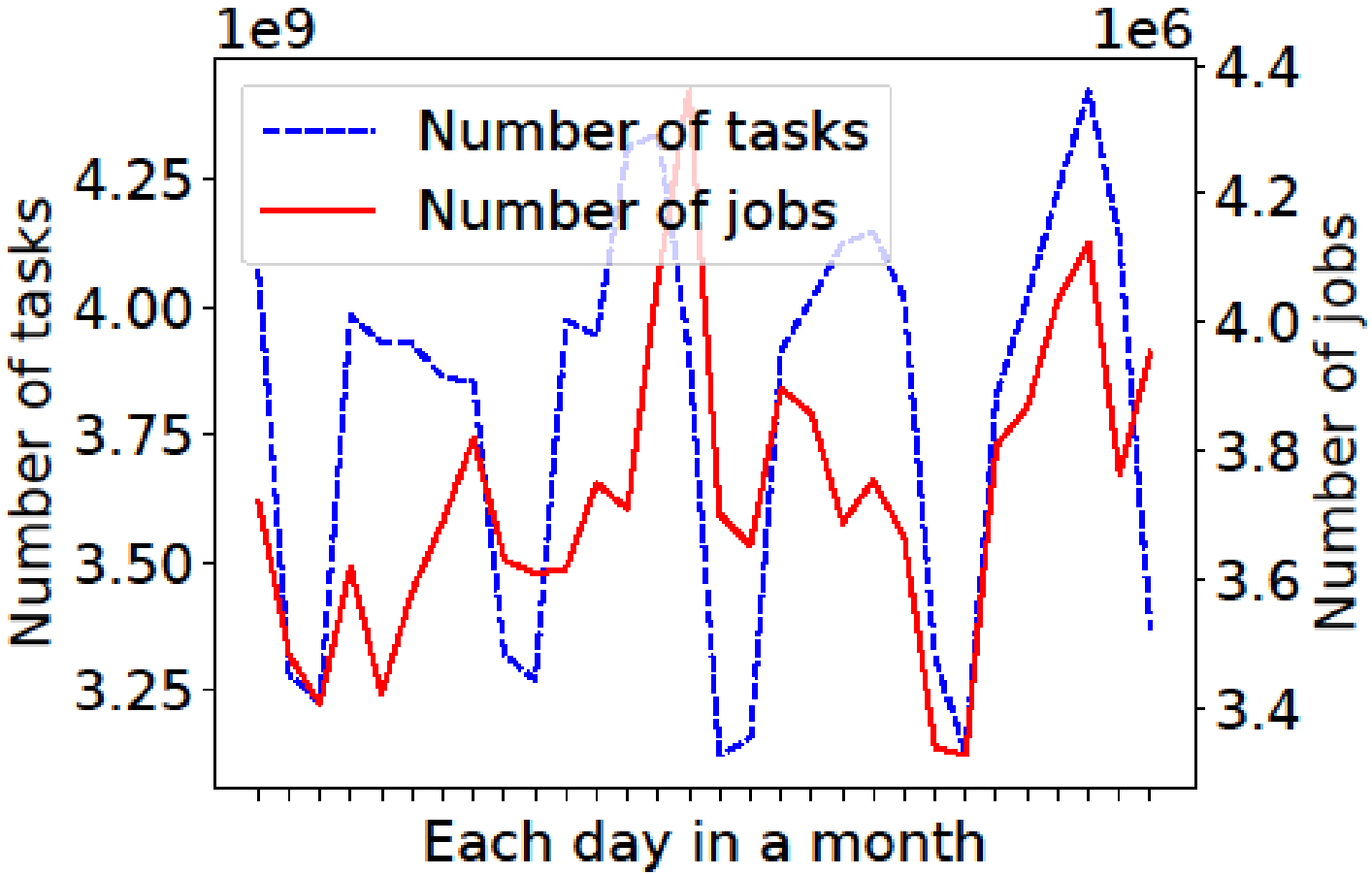
Billions

of tasks

Resource utilization

Cluster upgrade

Job/task statistics in Alibaba



Single-master architecture

- The scheduler can be overloaded with *heartbeat messages* from numerous workers

- The large number of tasks in our current production cluster exceeds the capability of a single scheduler

Scheduling objectives

- Scheduling efficiency, or scheduling delay
- Scheduling quality
- Fairness and priority between jobs and users
- Resource utilization

Scheduling objectives

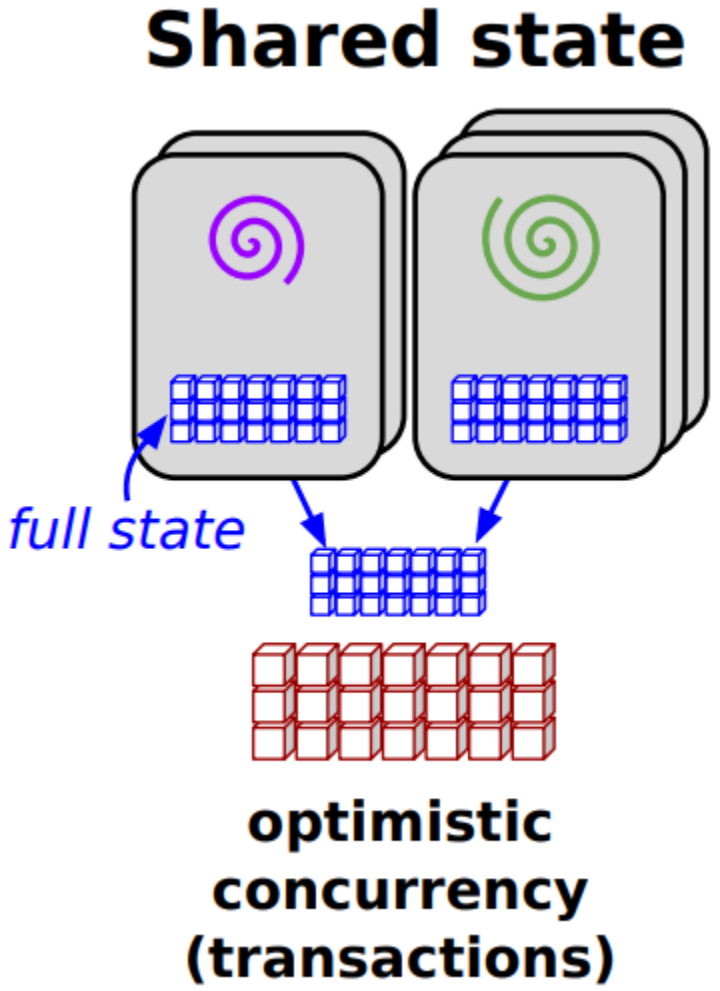
- Scheduling efficiency, or scheduling delay
- Scheduling quality
- Fairness and priority between jobs and users
- Resource utilization

Engineering objectives

- Robustness
- Backward compatibility
- Transparent-to-user upgrade

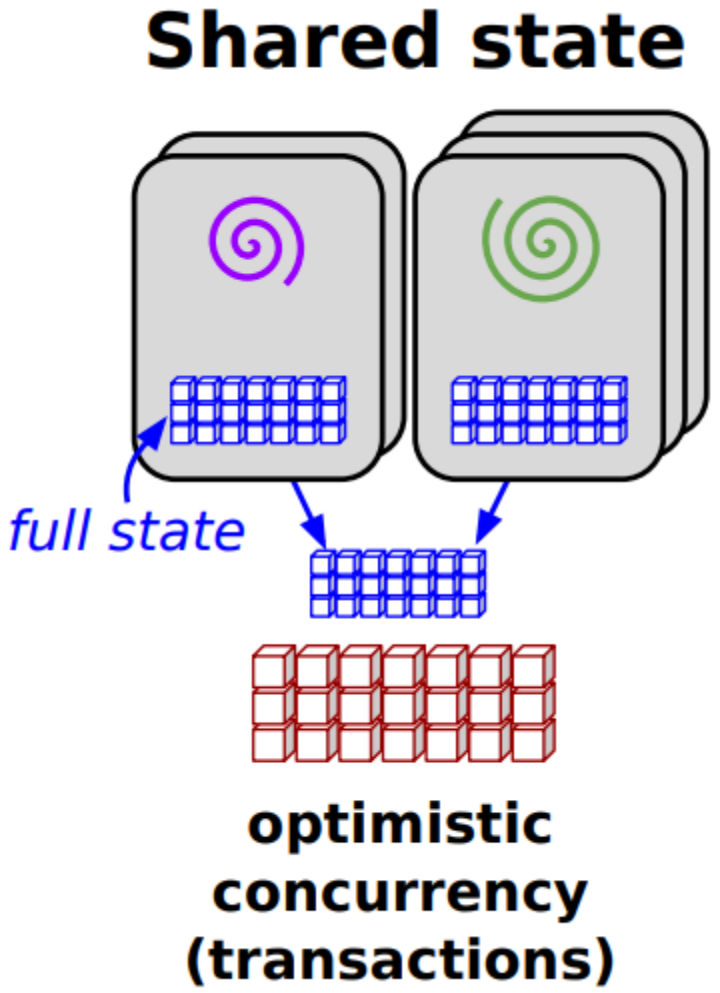
Shared-state architecture

- Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, John Wilkes. **Omega: flexible, scalable schedulers for large compute clusters**. (EuroSys 2013)



Shared-state architecture: benefits

- Each scheduler can run different scheduling strategies programmed in separate code bases for different types of jobs
- Each scheduler has a global view of the cluster
- Each scheduler can assign tasks to any machine in the cluster instead of a fixed subset of partitions



Shared-state architecture: limitations

- Omega assumes no synchronization overhead

- However, in our production system, there is a gap between consecutive synchronizations as our scheduler can be overloaded with network communication:
  - Frequent communication with application masters, a large number of worker machines in different racks, and a massive number of front-end requests
  - Large state in scheduling algorithms

Conflict modeling

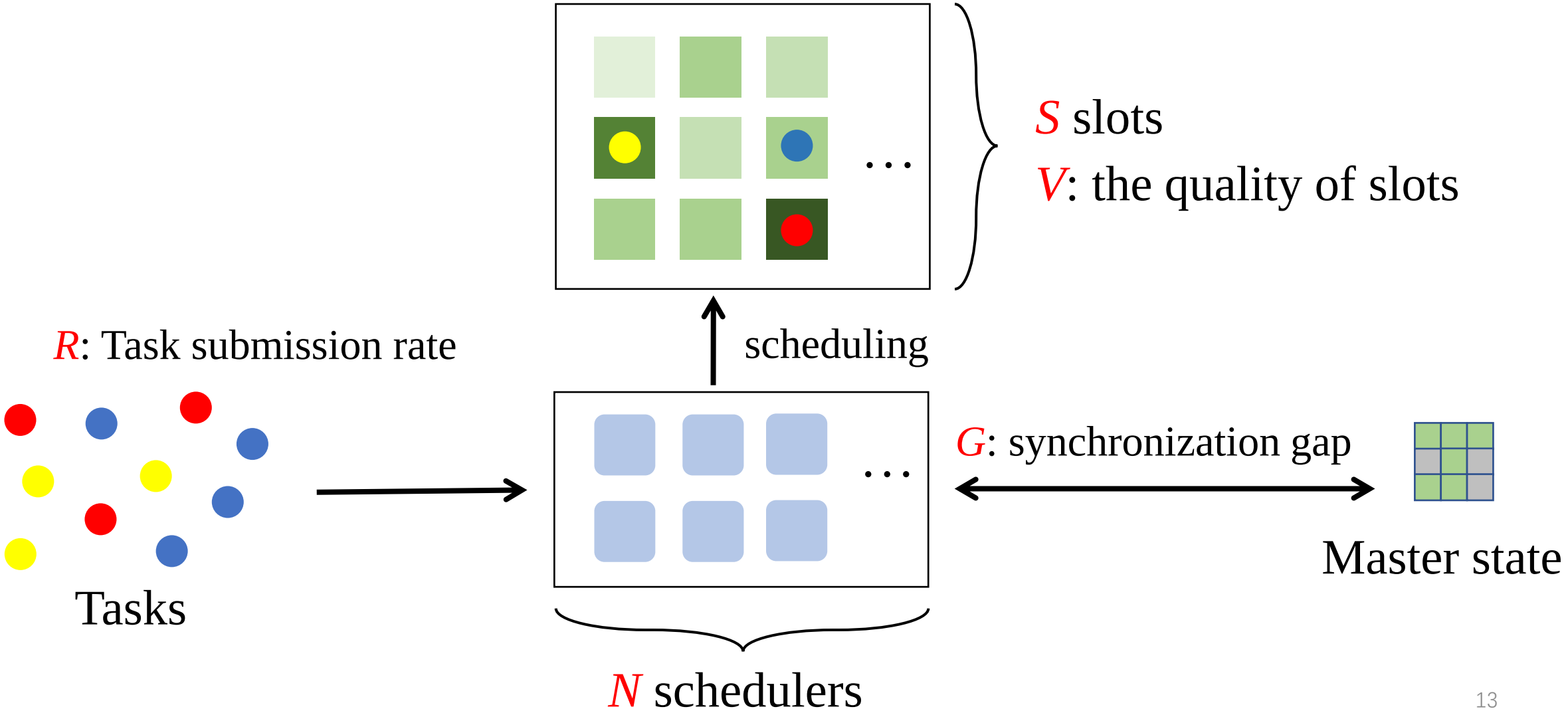
- The expectation of the number of conflicts Y_i at slot i, where S_{idle} is the number of idle slots, N is the number of schedulers and NK is the number of tasks to be scheduled:

$$\mathbb{E}(Y_i) = \frac{NK}{S_{idle}} - 1 + \left(1 - \frac{K}{S_{idle}}\right)^N$$

- The expectation of the total number of conflicts for all the slots, where S is the number of slots:

$$\mathbb{E}(\sum_{i=1}^{S} Y_i) = NK - S_{idle} + S_{idle} * (1 - \frac{K}{S_{idle}})^N$$

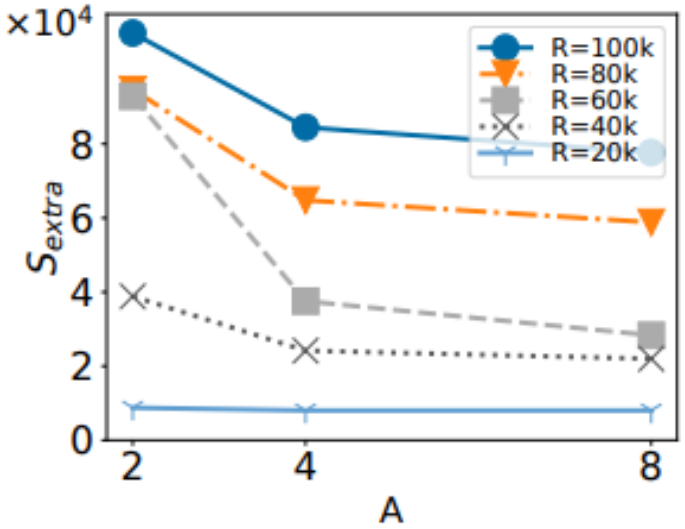
Light weight simulation



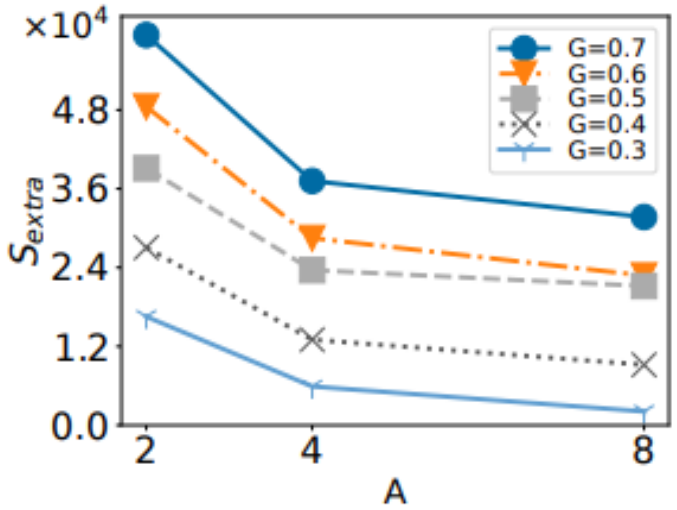
Light weight simulation

- An extreme case: the task submission rate and the resource needs of the tasks match with the total amount of resources in a cluster
- Ideal setting w/o any conflict: N schedulers → 0 scheduling delay
- When conflicts happen, we need extra schedulers and/or slots

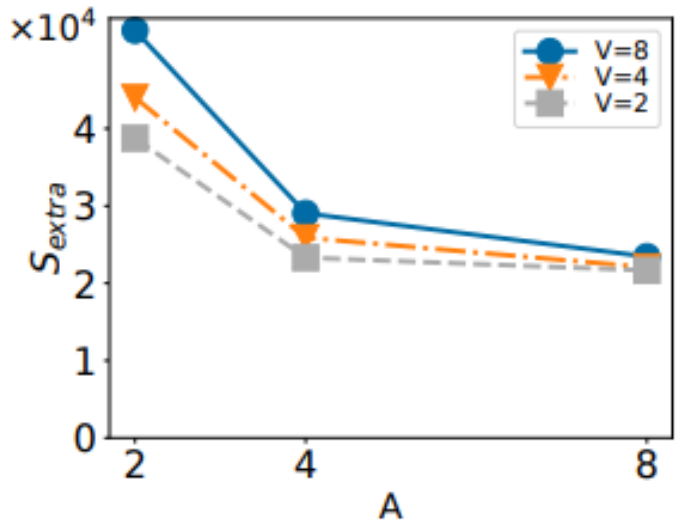
Light weight simulation: results



(a) Task submission rate (R)

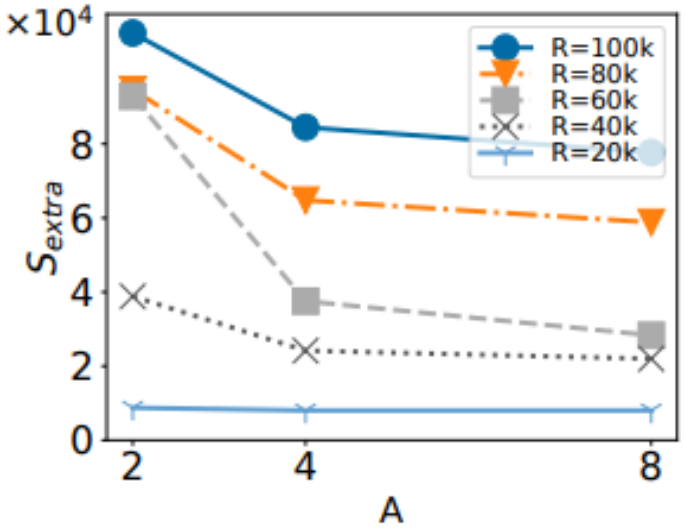


(b) Synchronization gap (G)

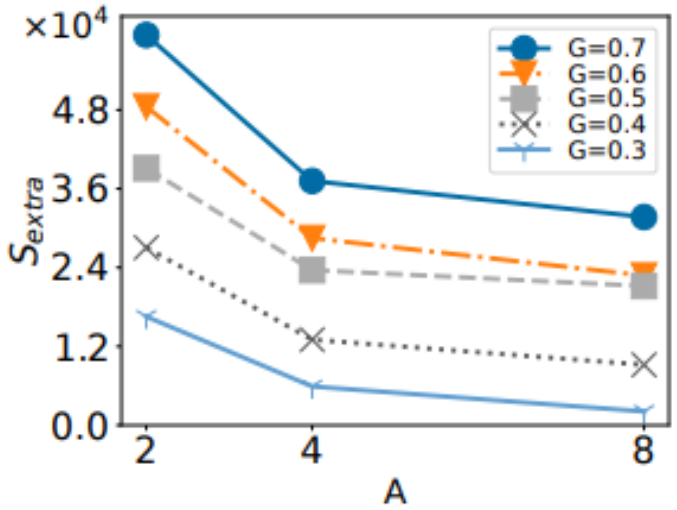


(c) Variance of slot scores (V)

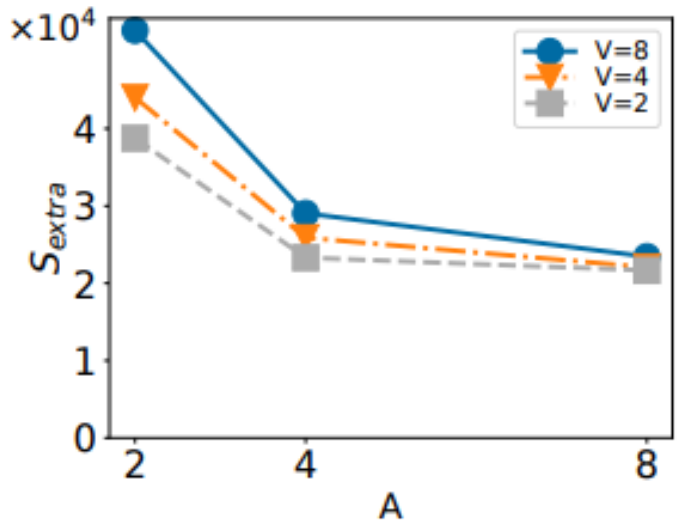
Light weight simulation: results



(a) Task submission rate (R)



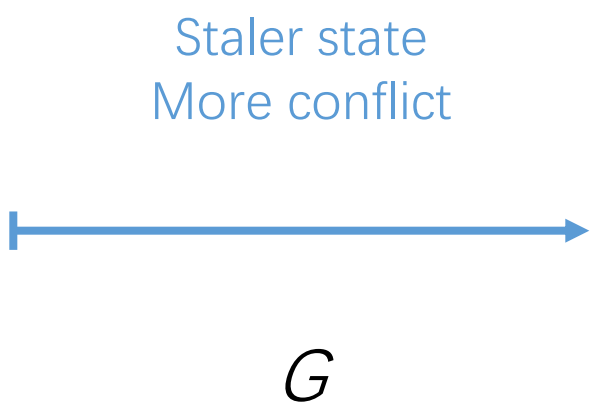
(b) Synchronization gap (G)



(c) Variance of slot scores (V)

Observation

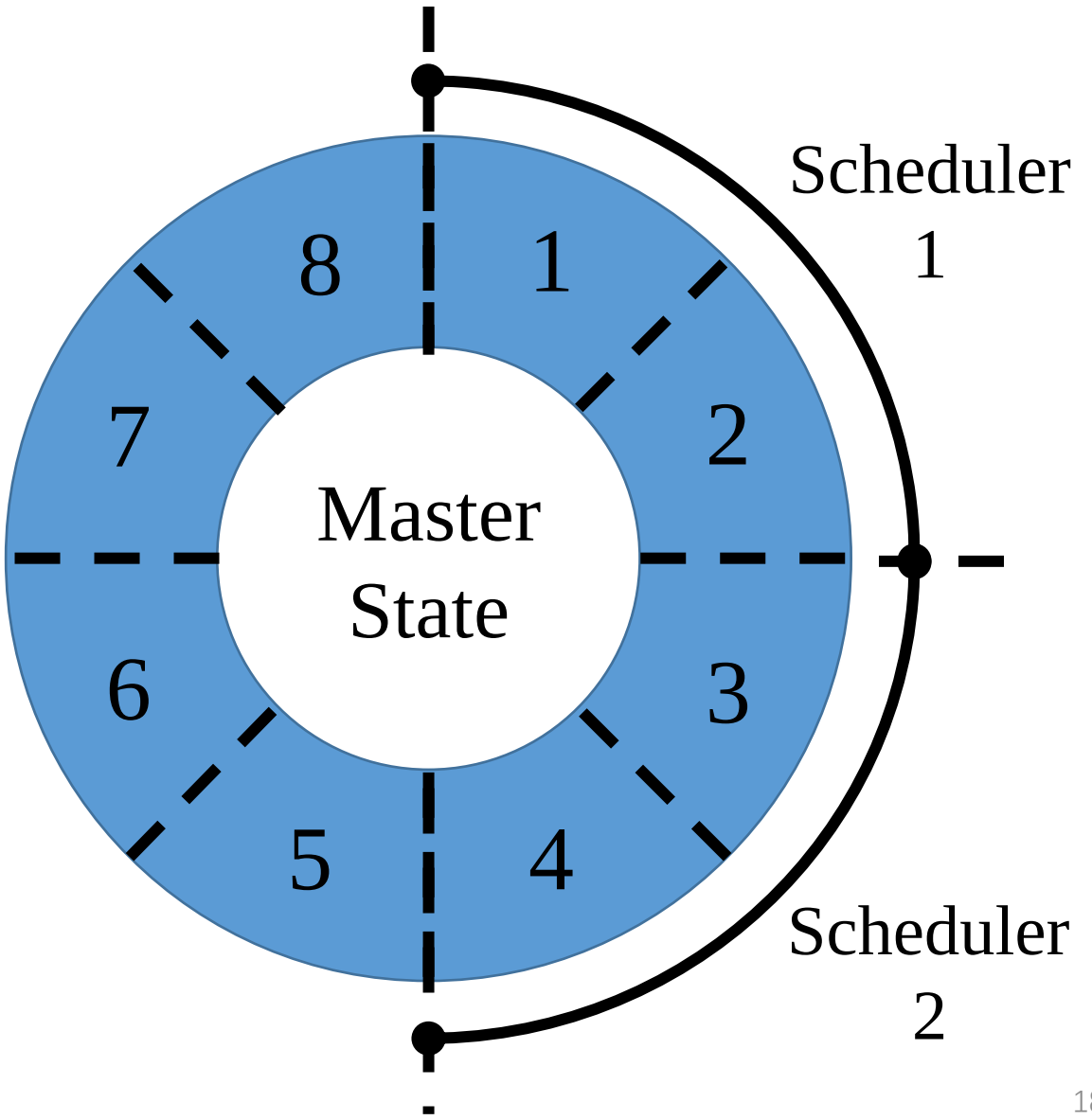
- Scheduling delay increases disproportionally within the gap G
  - When the state is synchronized, the scheduling has fewer conflicts
  - When the state is outdated, the scheduling results in more conflicts



- Most of the delay is caused by the staler view of the state in the later interval of the gap

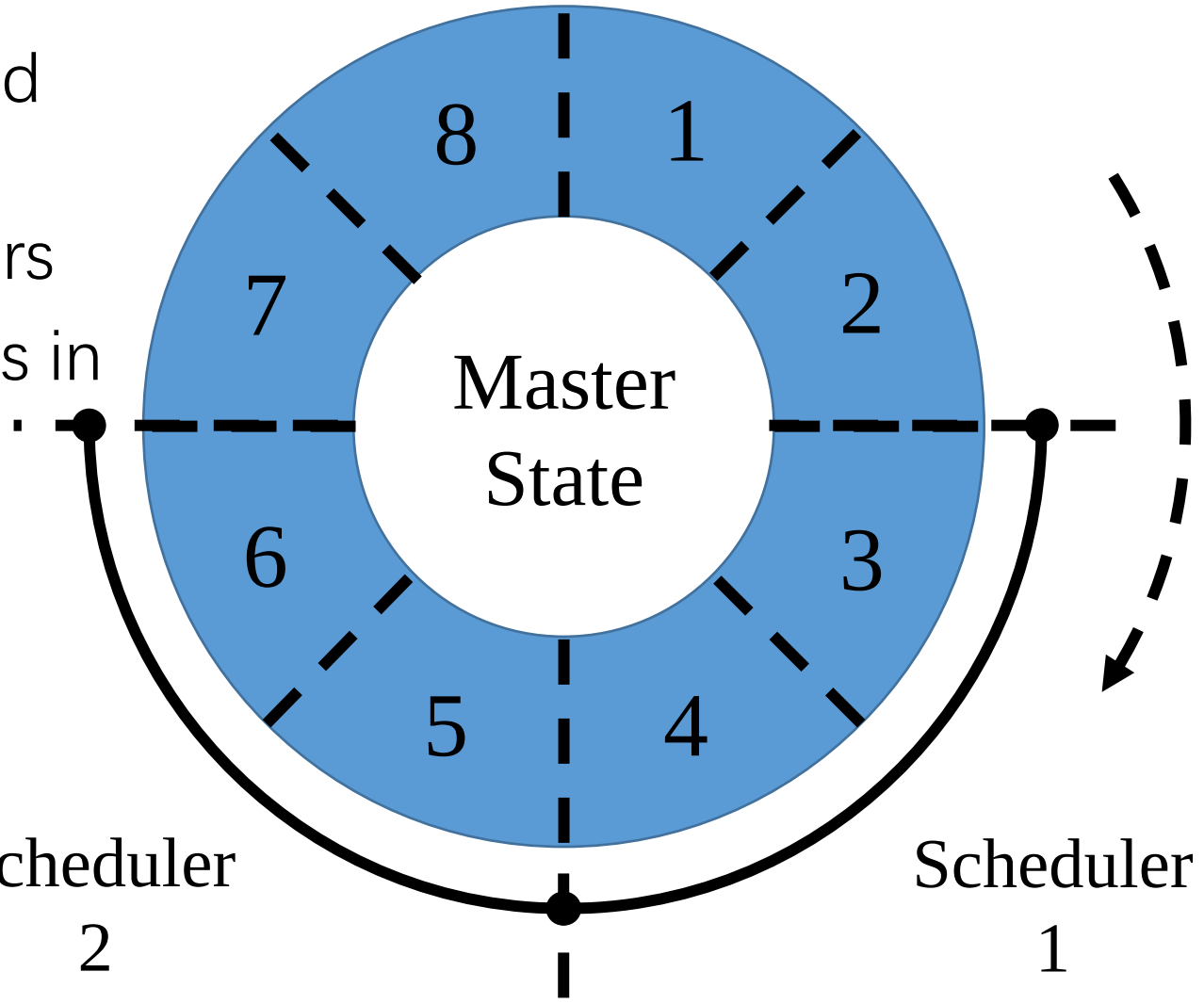
ParSync

- The master state is divided into P = 8 parts
- There are N = 4 schedulers
- Synchronization continues in a round-robin manner

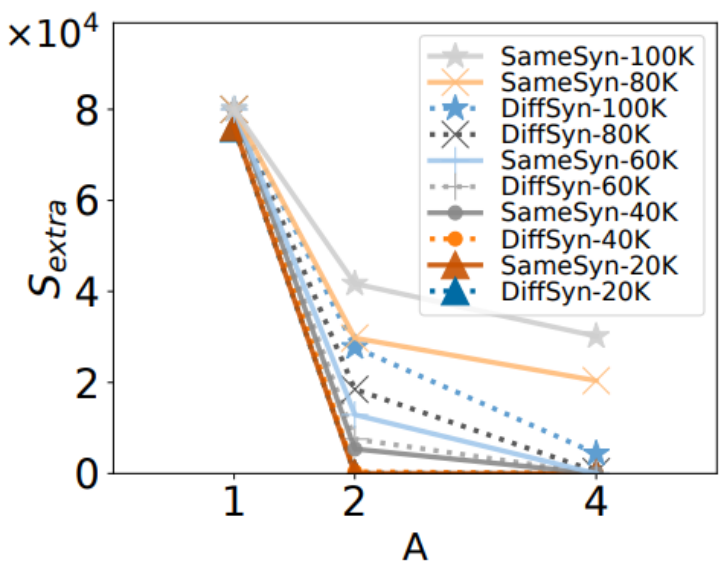


ParSync

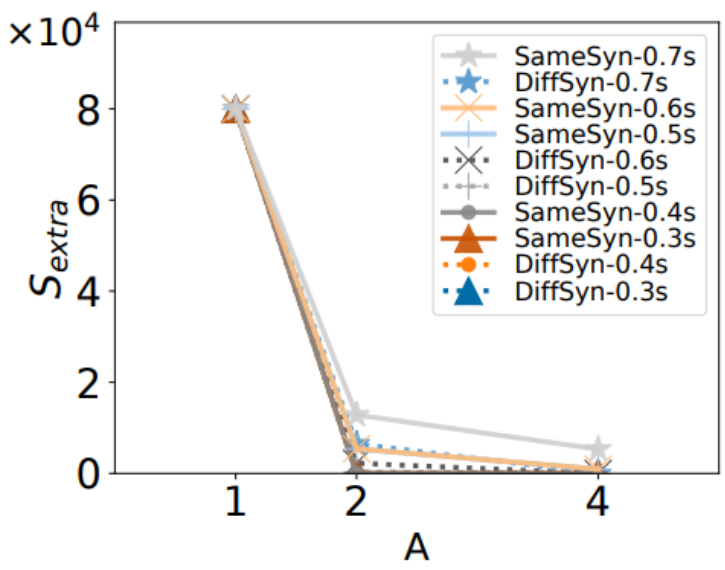
- The master state is divided into P = 8 parts
- There are N = 4 schedulers
- Synchronization continues in a round-robin manner



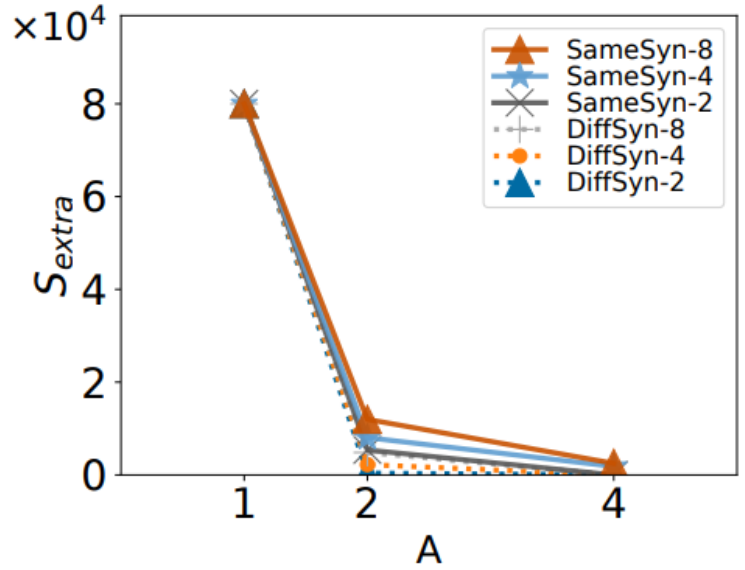
Effects of synchronization strategies



(a) Task submission rate (R)



(b) Synchronization gap (G)



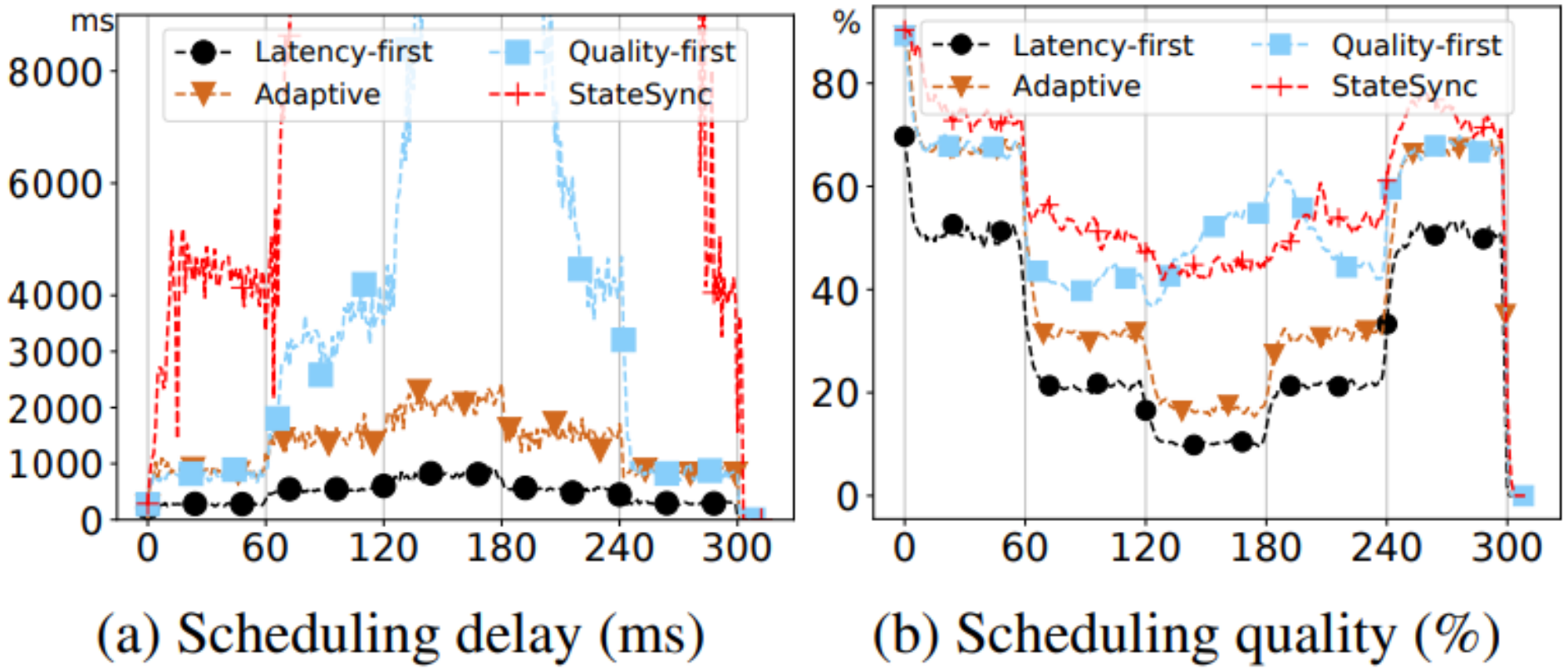
(c) Variance of slot scores (V)

Adaptive scheduling strategy

Quality-first	Latency-first	Adaptive	StateSync
Prioritize choosing of high-quality slots	Prioritize choosing of slots with fresher state	Hybrid of quality-first and latency-first	Synchronize entire state each time

DiffSync

Adaptive scheduling strategy



(a) Scheduling delay (ms)

(b) Scheduling quality (%)

Conclusions

ParSync

- Increases the scheduling capacity of our production cluster
- Reduces conflicts in contending resources to achieve low scheduling delay and high scheduling quality
- Allows us to maintain user transparency and backward compatibility

Thank you!

Q & A

If you have any question about our work, please contact

{ zliu, yjzhao, tjin } @cse.cuhk.edu.hk