A Network-aware Scheduler in Data-parallel Clusters for High Performance

Zhuozhao Li, Haiying Shen and Ankur Sarker

Department of Computer Science

University of Virginia

May, 2018

Introduction

- Data-parallel clusters
  - Used to process large datasets efficiently
- Deployed in many large organizations
  - E.g., Facebook, Google and Yahoo!
  - Shared by users from different groups

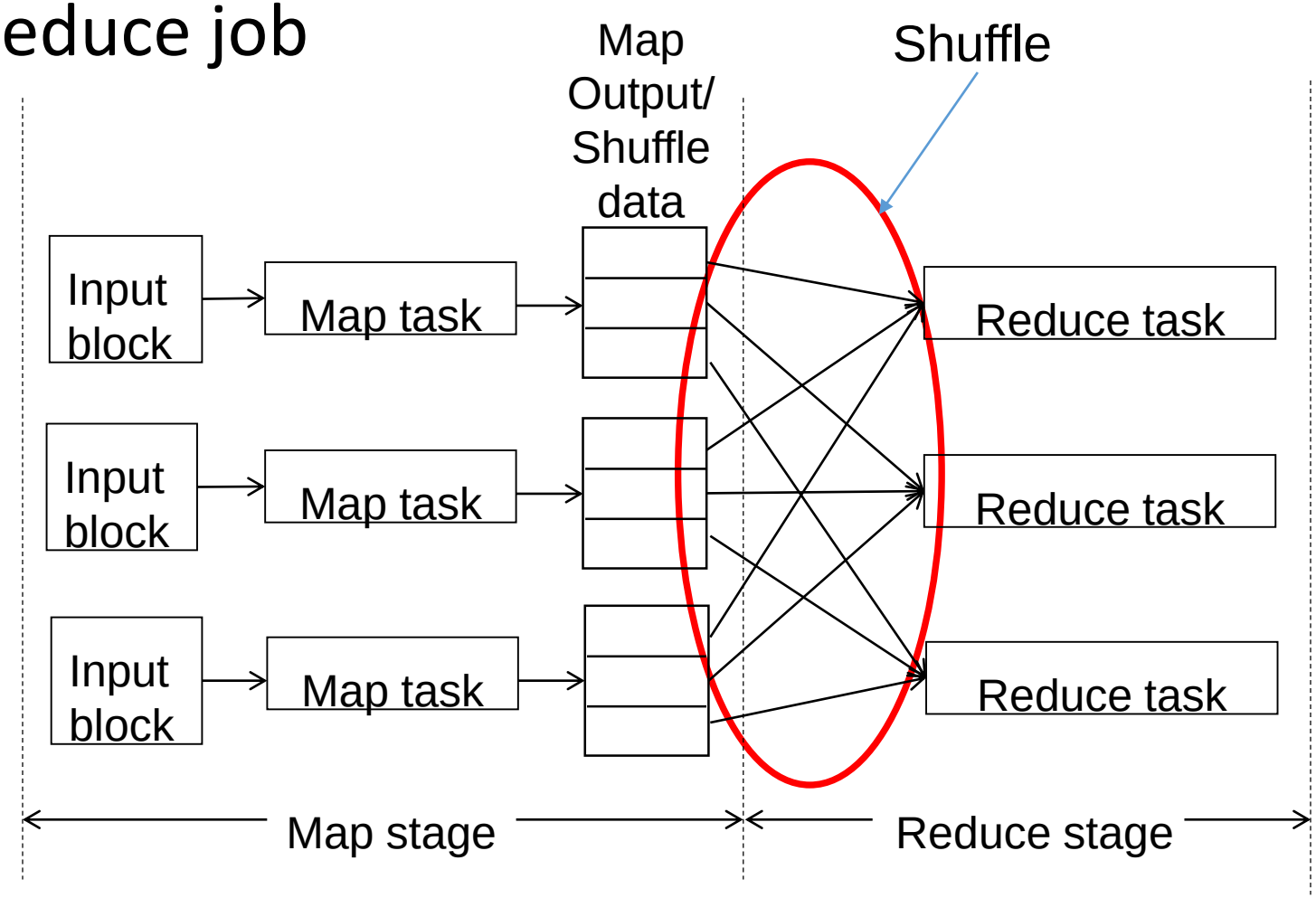
Motivations

- Network-intensive stages in data-parallel jobs

[1] M. Chowdhury, Y. Zhong, and I. Stoica. “Efficient coflow scheduling with varys”. In: Proc. of SIGCOMM. 2014.

MapReduce

A MapReduce job



Motivations

- Network-intensive stage
  - E.g., 60% and 20% of the jobs on the Yahoo and Facebook clusters, respectively, are reported to be shuffle-heavy
  - Jobs with large shuffle data size, generating a large amount of network traffic
  - More than 50% of time spent in network communication [2]

Problem: A large number of shuffle-heavy jobs may cause bottleneck on the cross-rack network

- Oversubscribed network from rack-to-core in datacenter
  - Oversubscription ratio ranging from 3:1 to 20:1

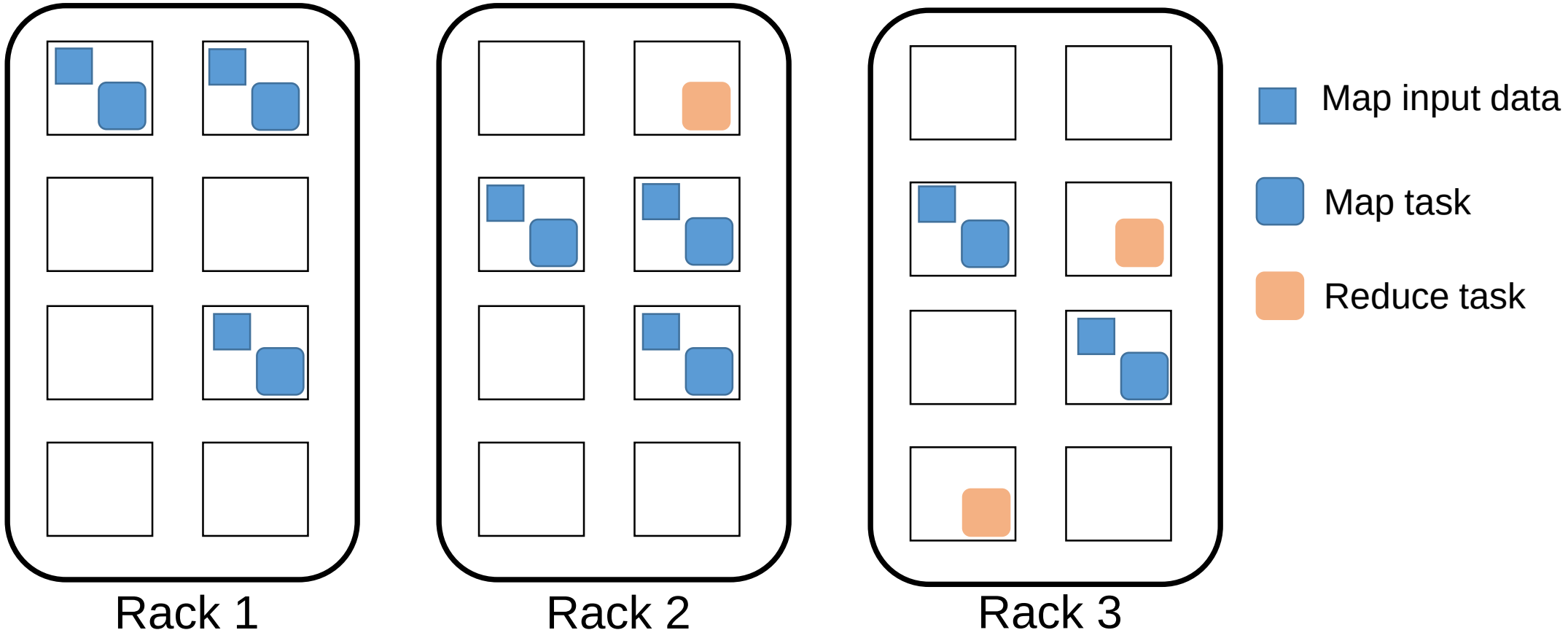
- Nearly 50% of cross-rack bandwidth used by background transfer

[1] M. Chowdhury, Y. Zhong, and I. Stoica. “Efficient coflow scheduling with varys”. In: Proc. of SIGCOMM. 2014.

Outline

- Introduction
- Related Work
- Network-Aware Scheduler Design (NAS)
- Evaluation
- Conclusion

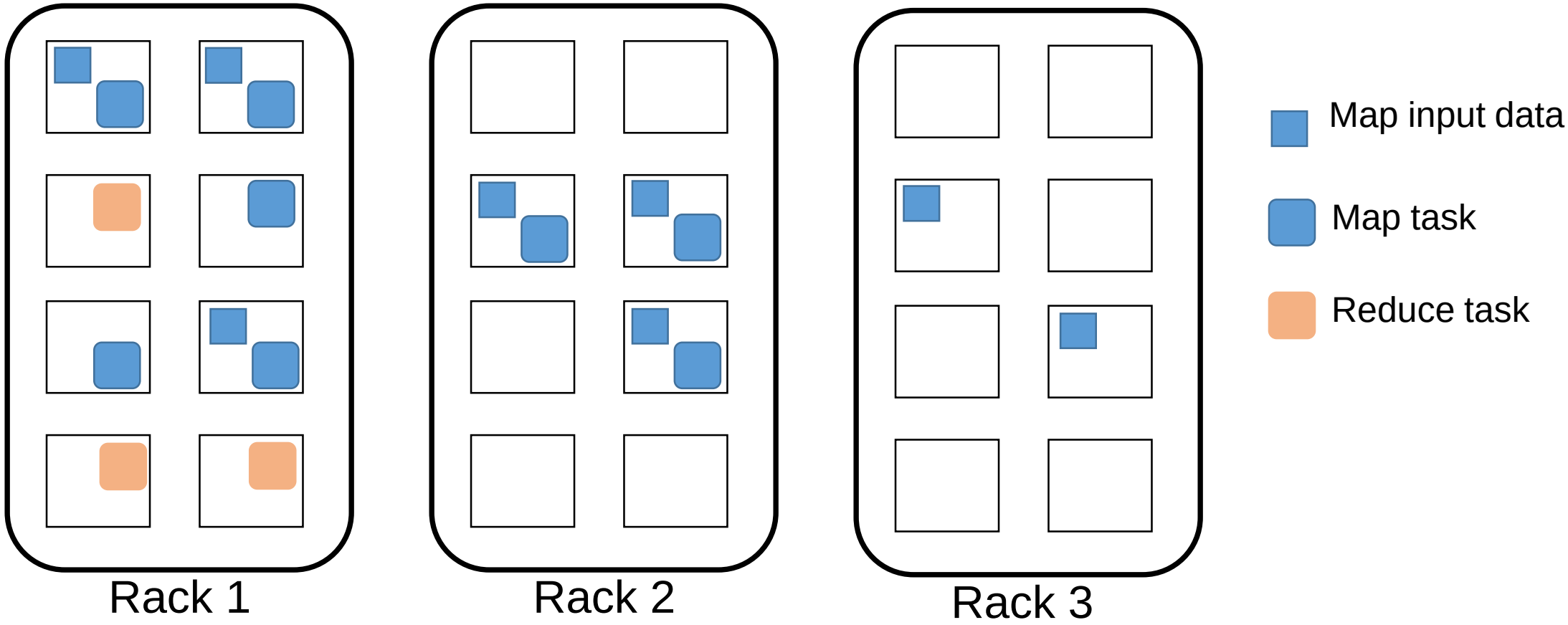
Related Work – Fair and Delay



Place the map task close to the input data – data locality

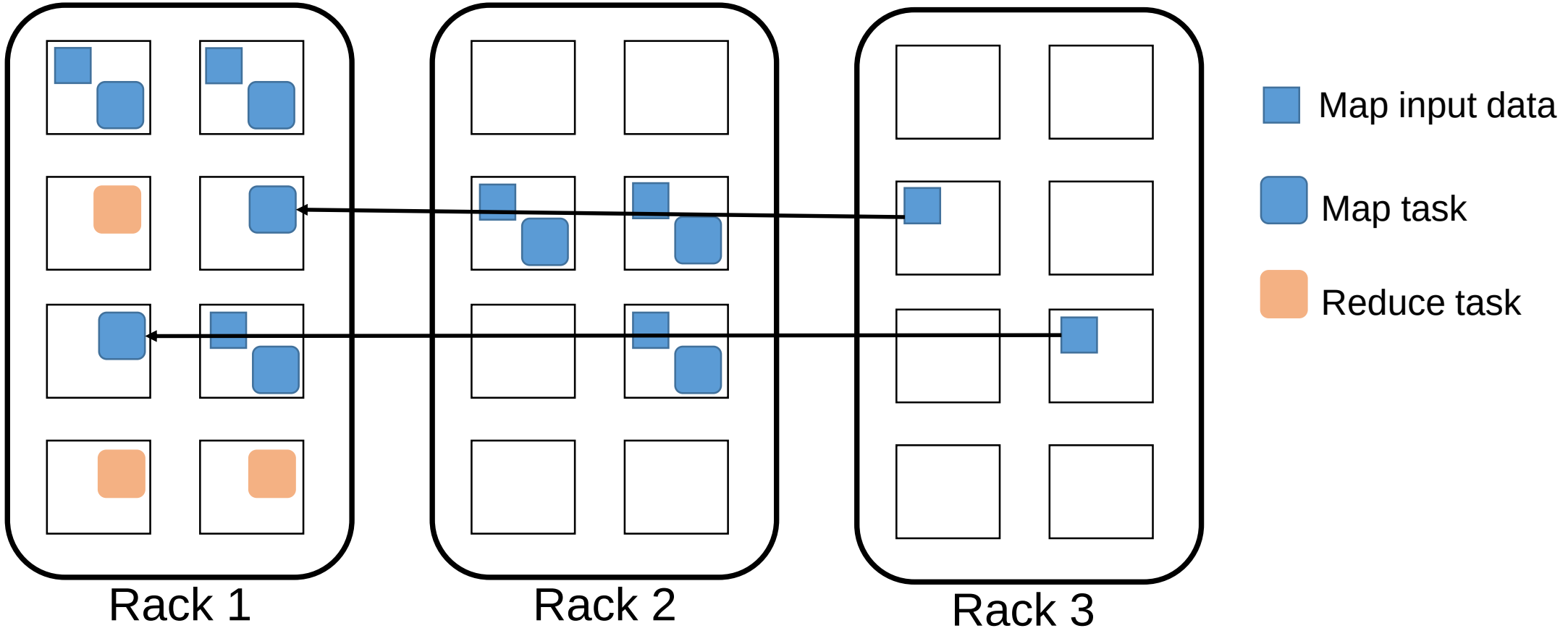
Problem: Place the reduce task randomly

Related Work – ShuffleWatcher (ATC'14)



Pre-compute the map and reduce placement and attempt to place map and reduce on the same racks to minimize the cross-rack traffic

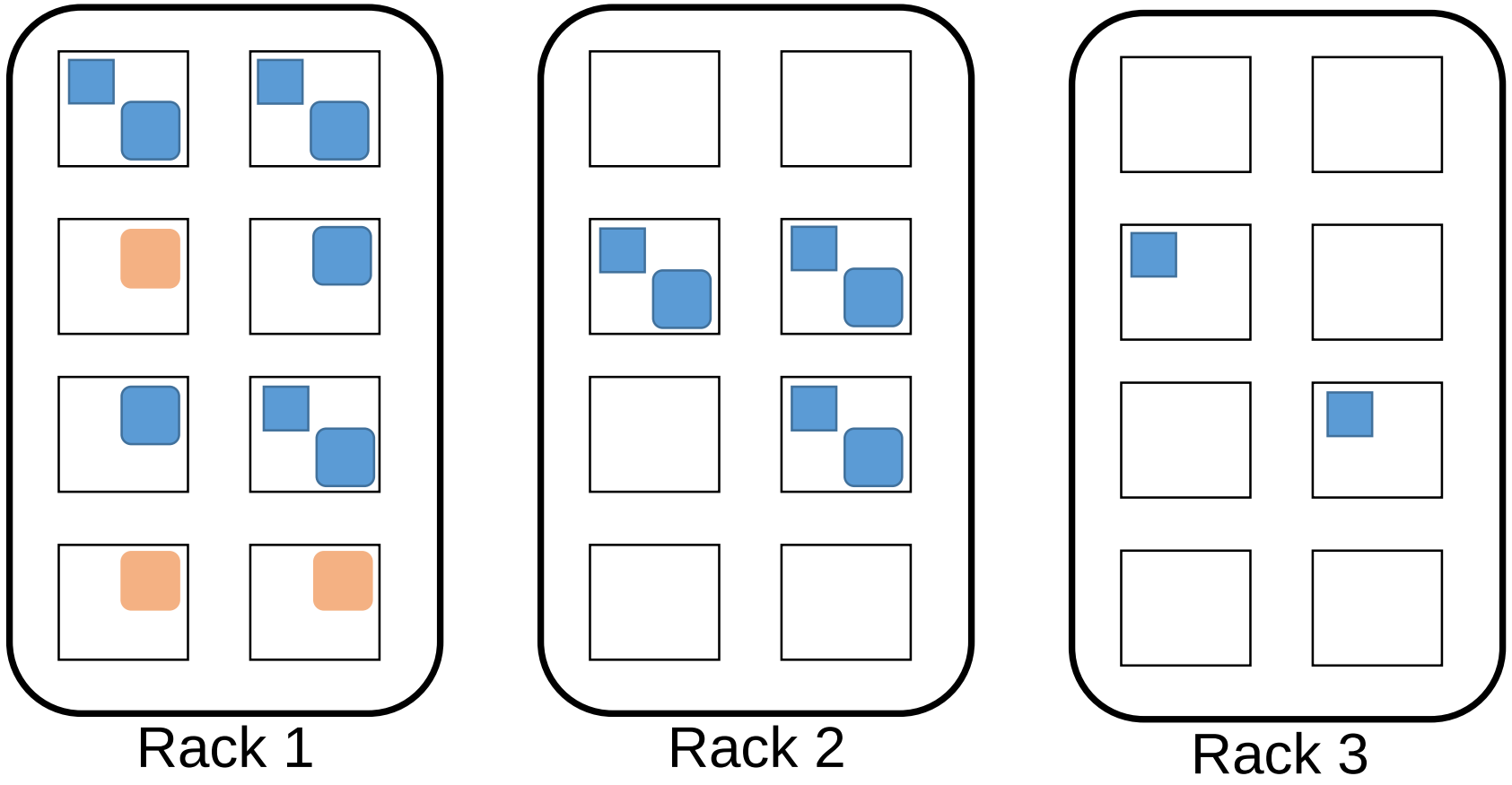
Related Work – ShuffleWatcher (ATC'14)



Problem:

Reduce the cross-rack shuffle traffic at the cost of reading remote map input data.

Related Work – ShuffleWatcher (ATC'14)



Problem:

Resource contention on the racks – intra-job and inter-job

Outline

- Introduction
- Related Work
- Network-Aware Scheduler Design (NAS)
- Evaluation
- Conclusion

Challenges

- Network-aware scheduler
  - How to reduce cross-rack congestion
  - How to reduce cross-rack traffic
- Idea
  - Network not saturated at all time
  - Designing schedulers to place tasks
    - Balance the network load
    - Consider shuffle data locality in addition to input data locality

Network-Aware Scheduler (NAS)

- Map task scheduling (MTS)
  - Balance the network load
- Congestion-avoidance reduce task scheduling (CA-RTS)
  - Consider shuffle data locality
- Congestion-reduction reduce task scheduling (CR-RTS)
  - Balance the network load

Map task scheduling (MTS)

- Goal: balancing the network load
- Set a TrafficThreshold for each node
  - Cannot process more shuffle data than this threshold at one time
  - Constrain the generated shuffle data size at a time
- Map task scheduling
  - Map input data locality and fairness
  - Whether the generated shuffle data size on a node exceeds the TrafficThreshold after placing a task

Map task scheduling (MTS)

- Setting the TrafficThreshold
  - Could be changed based on workloads
- Distribute the shuffle data into each wave
  - Task wave
    - Number of tasks >> number of containers
    - Tasks scheduled to all available containers, forming the first wave
    - Second wave, third wave …
  - TrafficThreshold $= \frac{TS}{N*W}$
  - TS – total shuffle data size of jobs in the cluster
  - N – the total number of nodes in the cluster
  - W – the number of waves: the total number of map tasks/the total number of containers

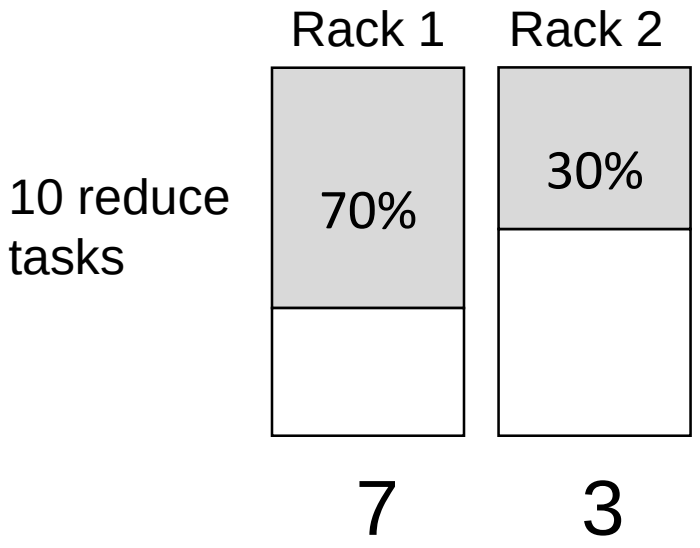
Map task scheduling (MTS)

- User 1:
  - Job1: 6 map tasks and 6 reduce tasks
  - Job3: 6 map tasks and 6 reduce tasks
- User 2:
  - Job2: 6 map tasks and 6 reduce tasks
  - Job4: 6 map tasks and 6 reduce tasks

- Each map -> each reduce
  - Job1 and Job2: 8
  - Job3 and Job4: 1

Congestion-avoidance Reduce Task Scheduling (CA-RTS)

- Check network status -- CongestionThreshold (e.g., 80% of cross-rack bandwidth)
- Used when the CongestionThreshold is NOT reached
- Goal: reduce cross-rack traffic
- A rack has more shuffle data of a job → assign more reduce tasks of the job on this rack to reduce cross-rack traffic



Congestion-reduction Reduce Task Scheduling (CR-RTS)

- Used when the CongestionThreshold is reached
- Goal: reduce cross-rack network congestion
- Launch a reduce task from a shuffle-light job
  - Small shuffle data size
  - Minimal impact on the cross-rack traffic

Outline

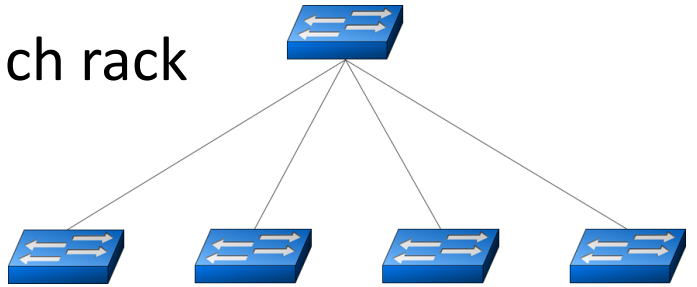
- Introduction
- Related Work
- Network-Aware Scheduler Design (NAS)
- **Evaluation**
- Conclusion

Evaluation

- Real cluster experiment
  - Throughput
  - Average job completion time
  - Cross-rack congestion
  - Cross-rack traffic
  - Sensitivity analysis
- Simulation study

Evaluation

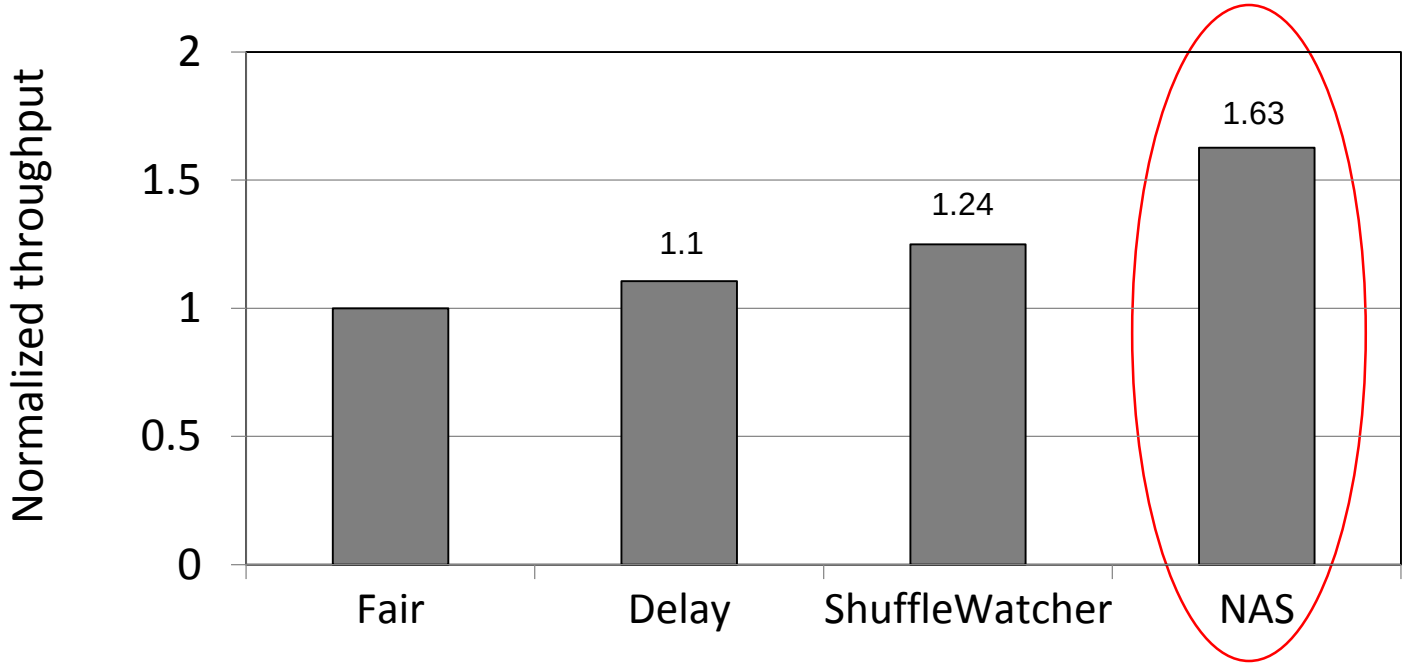
- Real cluster experiment
  - 40-node cluster organized into 8 racks, 5 nodes each rack
  - 8 racks interconnected by a core switch
  - Oversubscription 5:1 from the rack to core



- Workload
  - 200 jobs from the Facebook synthesized execution framework [1]
- Baselines
  - Fair Scheduler (current scheduler in Hadoop)
  - Delay Scheduler (current scheduler in Hadoop)
  - ShuffleWatcher (ATC'14)

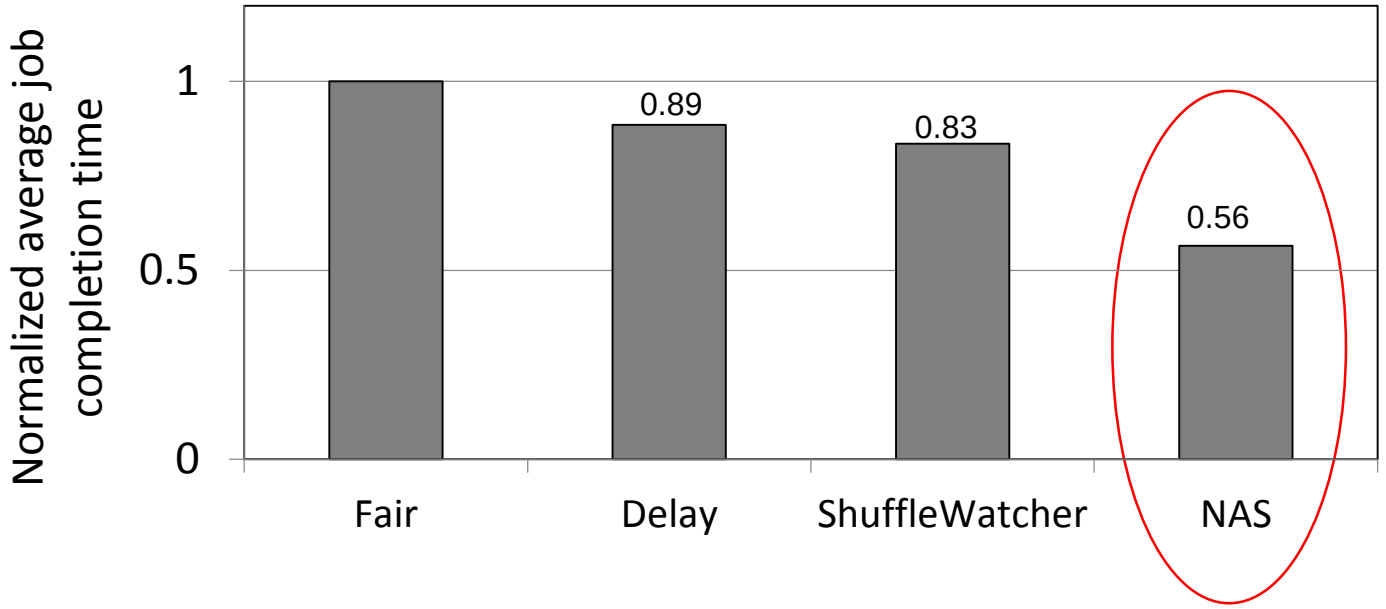
[1] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. "The Case for Evaluating MapReduce Performance Using Workload Suites." In: Proc. of MASCOTS. 2011.

Throughput



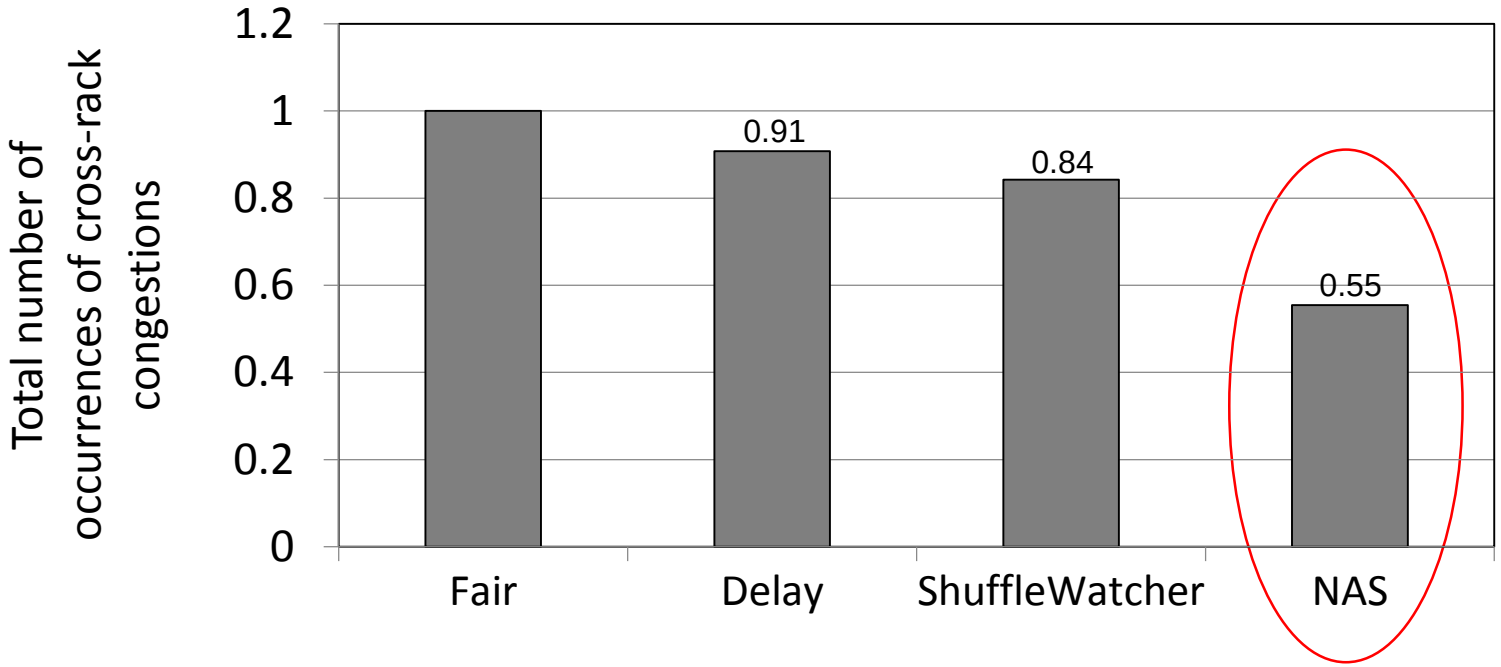
NAS improves the throughput over Fair, Delay and ShuffleWatcher by 63%, 48%, 31%, respectively

Average Job Completion Time



NAS reduces the average job completion time over Fair, Delay and ShuffleWatcher by 44%, 37%, 33%, respectively

Cross-rack congestion



NAS reduces the cross-rack congestion over Fair, Delay and ShuffleWatcher by 45%, 40%, 34%.

Conclusion

We can improve the performance of current state-of-the-art schedulers (e.g., Fair and Delay schedulers in Hadoop) by

- balancing the network traffic and enforcing the data locality for shuffle data,
- aggregating the data transfers to efficiently exploit optical circuit switch in hybrid electrical/optical datacenter network while still guaranteeing parallelism,
- and adaptively scheduling a job to either scale-up machines or scale-out machines that benefit the job the most in hybrid scale-up/out cluster.

Questions?

Zhuozhao Li

Ph.D. Candidate
Department of Computer Science
University of Virginia
ZL5UQ@VIRGINIA.edu

Backup

Shuffle Data Size Predictor

- MapOutput = (map output/input ratio) $*$ MapInput

- Unpredicted and predicted job

- Update in real time

Map task scheduling (MTS)

- Setting the TrafficThreshold
  - Could be changed based on workloads

- Distribute the shuffle data into each wave
  - Task wave
    - Number of tasks >> number of containers
    - Tasks scheduled to all available containers, forming the first wave
    - Second wave, third wave …
  - TrafficThreshold $= \frac{TS}{N*W}$
  - TS – total shuffle data size of jobs in the cluster
  - N – the total number of nodes in the cluster
  - W – the number of waves: the total number of map tasks/the total number of containers

Map task scheduling (MTS)

Algorithm 2 Pseudocode for MTS.

Require: Initialize skip count of the i^{th} user $D_i^m = 0$
maximum number of skips D^m

```
1: Calculate the available map output data size on the worker node.
2: for user i in the user list do
3:   if the user has data-local and shuffle-qualified map task then
4:     launch this map task on this node, set D_i^m = 0
5:   else
6:     if D_i^m == D^m then
7:       if we can find shuffle-qualified map tasks of this user then
8:         launch a map task in the following order:
9:             (1) map task from small-input unpredicted job
10:            (2) map task from small-input predicted job
11:            (3) map task from large-input unpredicted job
12:            (4) map task from large-input predicted job
13:      else
14:        launch a map task in the following order:
15:            (1) data-local map task
16:            (2) map task with the smallest map output data size
17:      end if
18:    else
19:      D_i^m++
20:    end if
21:  end if
22: end for
```

Congestion-avoidance Reduce Task Scheduling (CA-RTS)

Algorithm 3 Pseudocode for CA-RTS.

1: Select a user from the user list based on fairness.
2: Launch reduce task from a job that satisfies map completion threshold in the following order (a job with *delayed* or $MapProgressRate = 100\%$ has higher priority in the same category):
3: (1) Shuffle-heavy jobs whose $ReduceNum$ is not reached,
4: (2) Shuffle-medium jobs whose $ReduceNum$ is not reached
5: (3) Shuffle-light jobs whose $ReduceNum$ not reached
6: (4) Shuffle-light jobs whose $ReduceNum$ is reached
7: (5) Shuffle-medium jobs whose $ReduceNum$ is reached
8: (6) Shuffle-heavy jobs whose $ReduceNum$ is reached

Congestion-reduction Reduce Task Scheduling (CR-RTS)

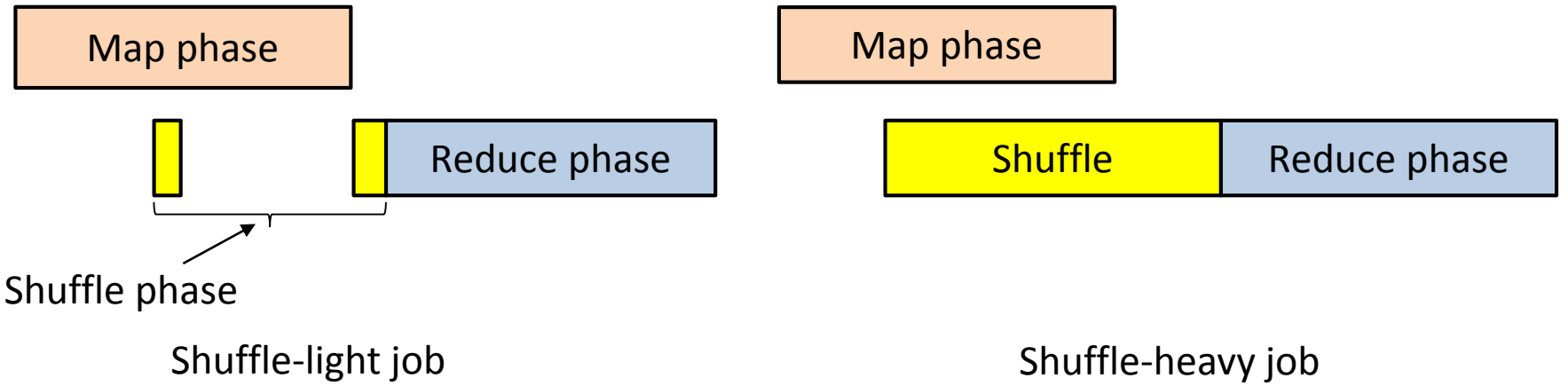
Algorithm 4 Pseudocode for CR-RTS.

Require: Initialize skip count of the i^{th} user $D_i^r = 0$
maximum number of skips D^r

```
1: for user i in the user list do
2:   if D_i^r < D^r then
3:     if this user has shuffle-light jobs then
4:       Select a reduce task from shuffle-light jobs, set D_i^r = 0
5:     else
6:       D_i^r++ and skip this user
7:     end if
8:   else
9:     Select a reduce task from any jobs
10:  end if
11: end for
```

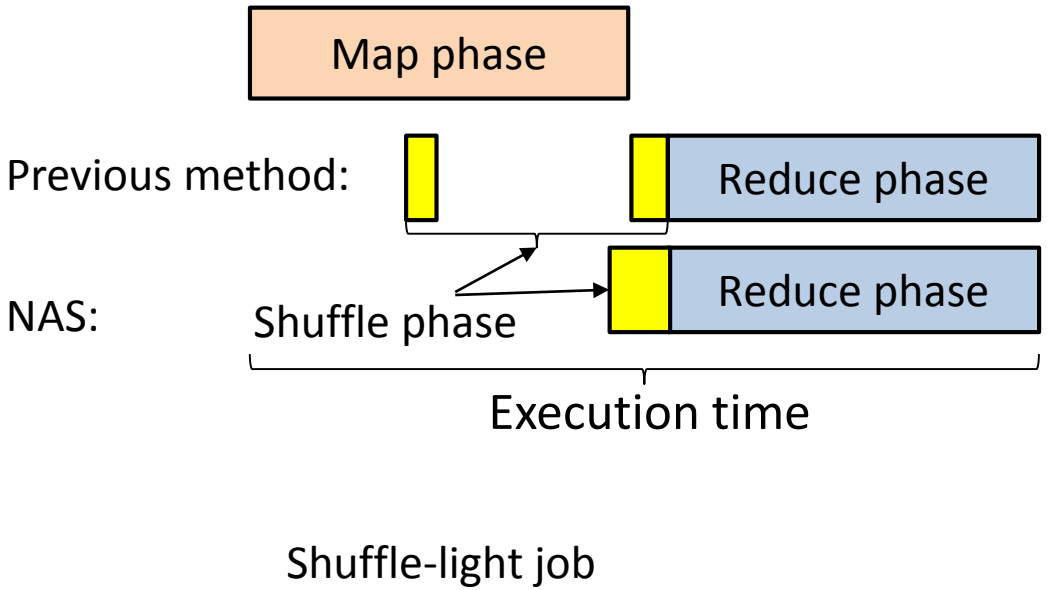
Optimization of Map Completion Threshold

- Map completion threshold (slowstart threshold)
  - Start scheduling reduce tasks
  - Start shuffle transfer immediately after the reduce task is scheduled a container



Optimization of Map Completion Threshold

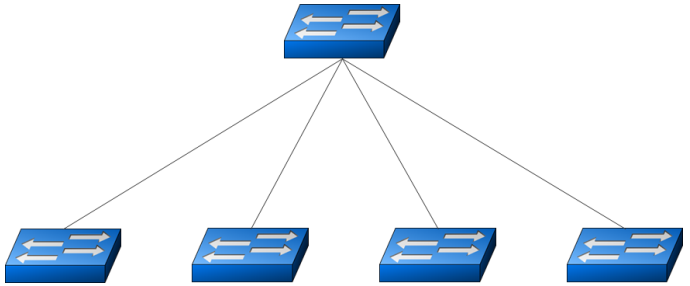
- Drawback: occupy the container without processing but just waiting for shuffle data
- Adaptive map completion threshold for different jobs



Classification of Jobs in NAS

Type	Range
Shuffle-light	< 1MB
Shuffle-medium	1 – 100MB
Shuffle-heavy	> 100MB

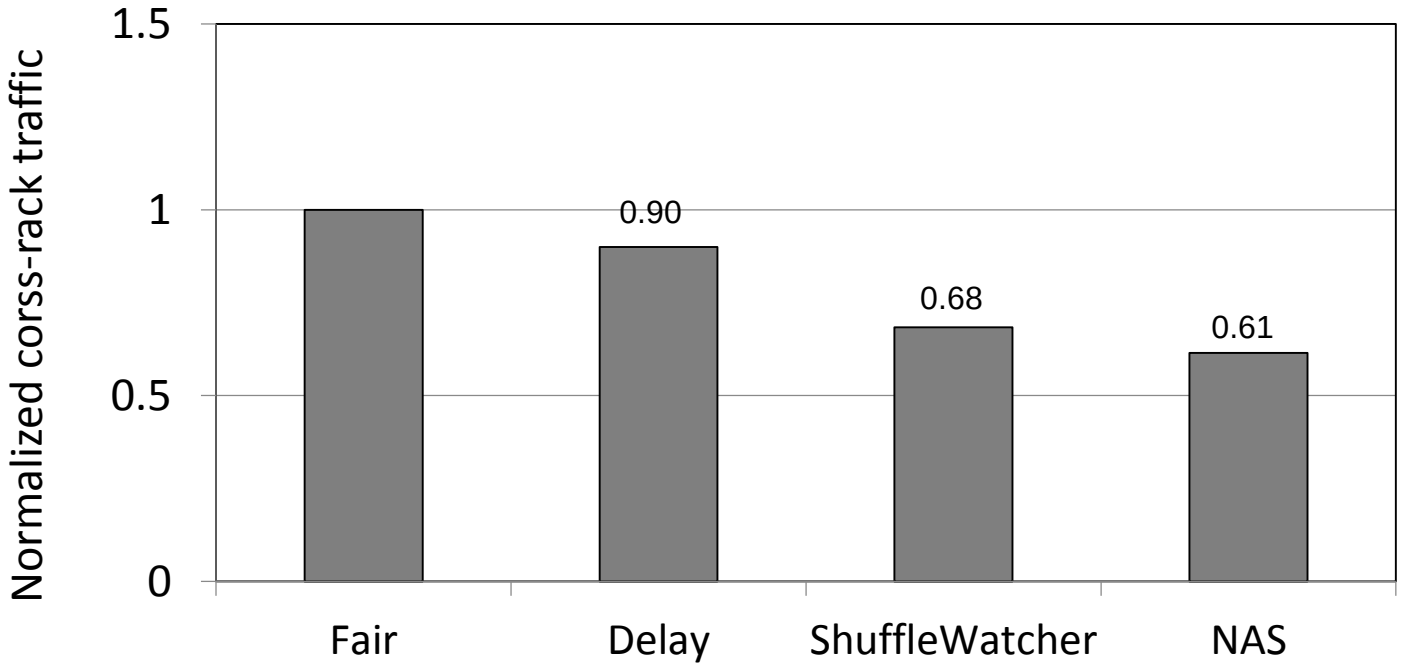
Evaluation



- Real cluster experiment
  - 40-node cluster organized into 8 racks, 5 nodes each rack, 1Gbps each node
  - All ToRs connected by a core switch. 1Gbps from core to ToR, oversubscription 5:1
  - 16 containers on each node
- Workload
  - 200 jobs from the Facebook synthesized execution framework [1]
  - Arrival in exponential distribution with a mean of 14 seconds
- Baselines
  - Fair Scheduler
  - Delay Scheduler
  - ShuffleWatcher

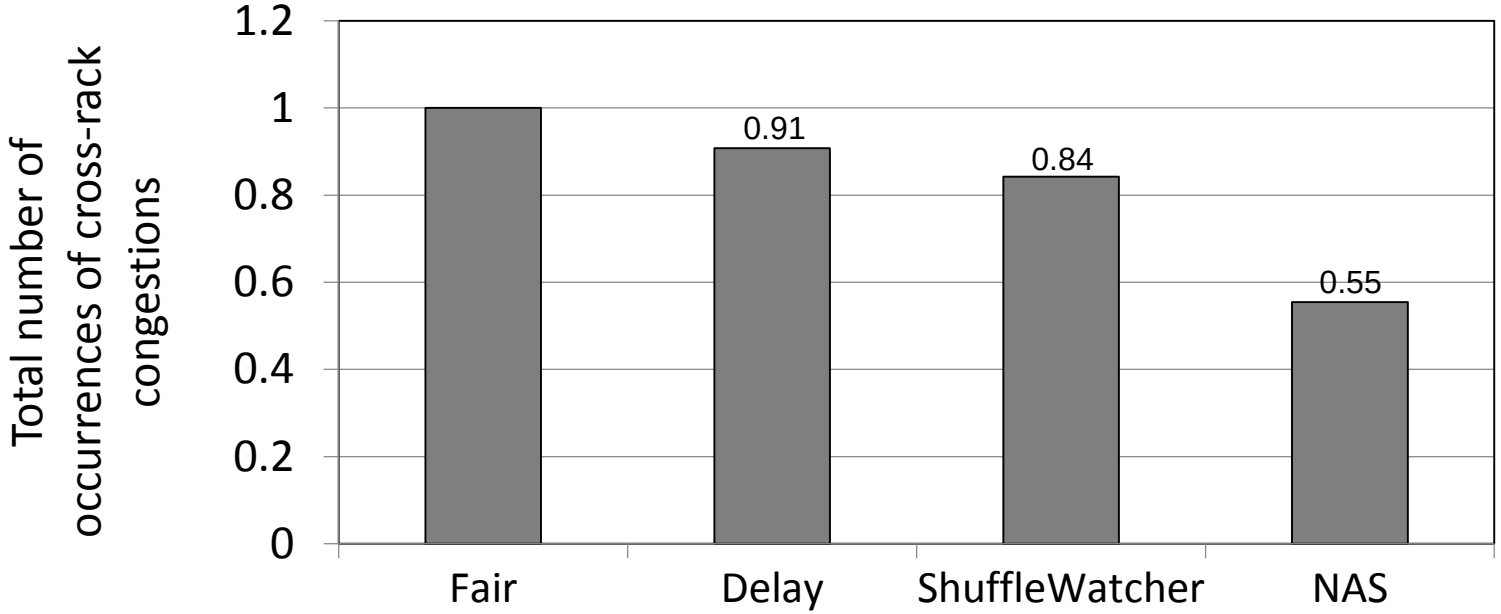
[1] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. “The Case for Evaluating MapReduce Performance Using Workload Suites.” In: Proc. of MASCOTS. 2011.

Cross-rack traffic in real cluster



NAS reduces the cross-rack traffic over Fair, Delay and ShuffleWatcher by 39%, 32%, 11%.

Cross-rack congestion in real cluster



NAS reduces the cross-rack congestion over Fair, Delay and ShuffleWatcher by 45%, 40%, 34%.