Optimizing MapReduce through Joint Scheduling of Overlapping Phases

Huanyang Zheng, Ziqi Wan, and Jie Wu

Dept. of Computer and Info. Sciences

Temple University

Road Map

- Introduction
- Model and Formulation
- Observation and Ideas
- Algorithms
- Experiments
- Conclusion

1. Introduction

Map-Shuffle-Reduce

Map and Reduce: CPU-intensive

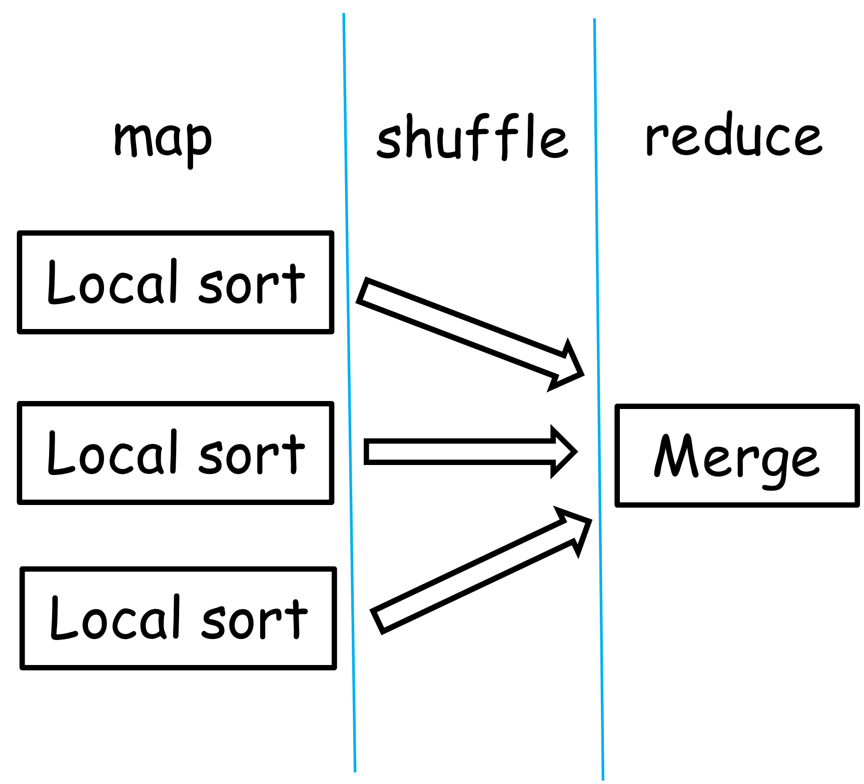
Shuffle: I/O-intensive

Merge Sort

Map: sorts local arrays

Shuffle: shuffles sorted arrays

Reduce: merges sorted arrays



Introduction

Map-Shuffle-Reduce Jobs

- Reduce is not discussed (Zaharia, OSDI 2008)
  - Only 7% of jobs in MapReduce are reduce-heavy
- Map and Shuffle
  - CPU-intensive and I/O-intensive (can overlap)
- Centralized scheduler
  - Determine an execution order of jobs on map pipeline and shuffle pipeline

Introduction

Dependency relationship

The map emits data at a given rate

Shuffle waits for the data emitted by map

may be delayed by the scheduling policy

Job classification

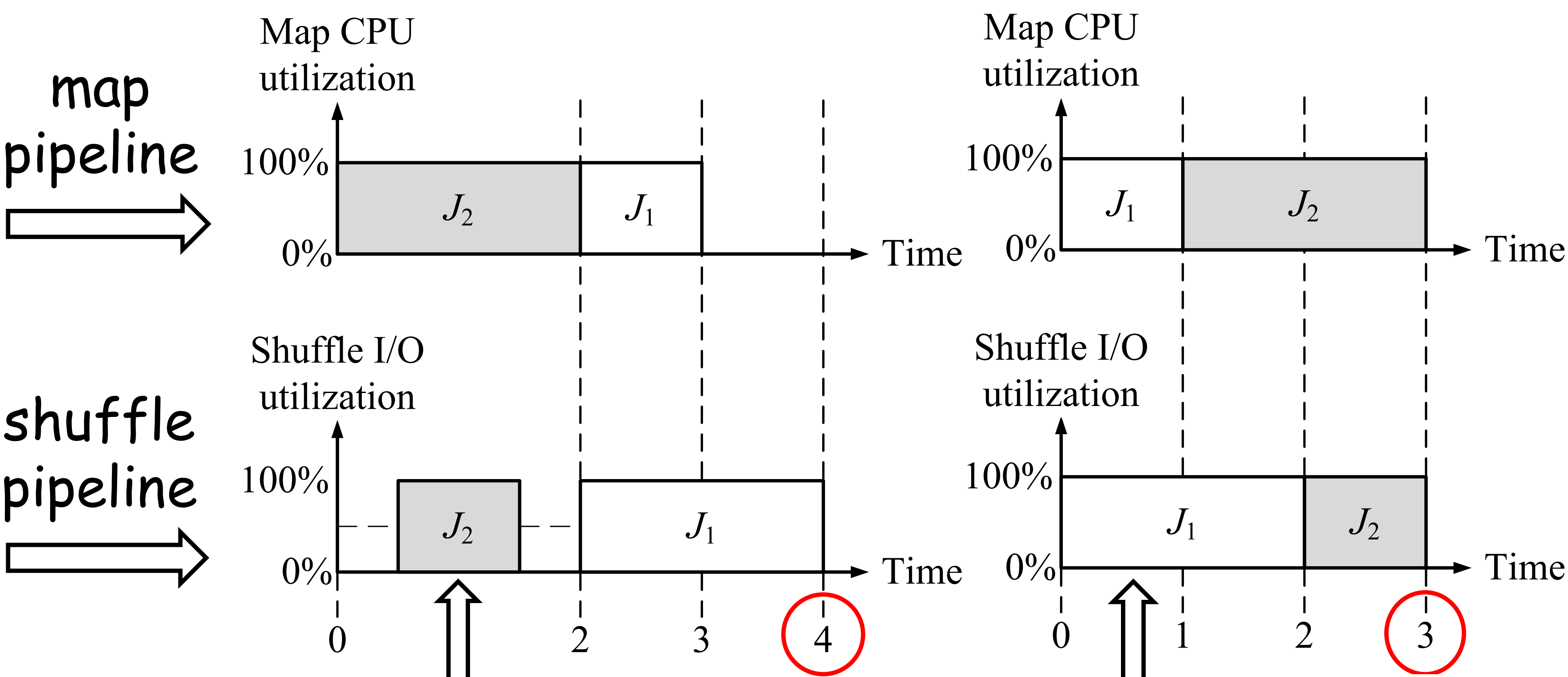
Map-heavy: map workload > shuffle workload

Balanced: map workload = shuffle workload

Shuffle-heavy: map workload < shuffle workload

Introduction

Impact of overlapping map and shuffle phases



Word count (map-heavy)

Merge sort (shuffle-heavy)

2. Model and Formulation

Jobs in Map-Shuffle-Reduce

A set of n jobs:

$J = \{ J_1, J_2, \ldots, J_n \}$

t_i^m map workload of J_i

t_i^s shuffle workload of J_i

Job classification:

Map-heavy if $t_i^m > t_i^s$

Shuffle-heavy if $t_i^m < t_i^s$

Balanced if $t_i^m = t_i^s$

Model and Formulation

Schedule objective

- Minimize average job completion time
  - includes waiting time before job start

- Schedule is NP-hard

Offline scenarios

- All jobs arrival at the beginning (waiting for schedule)

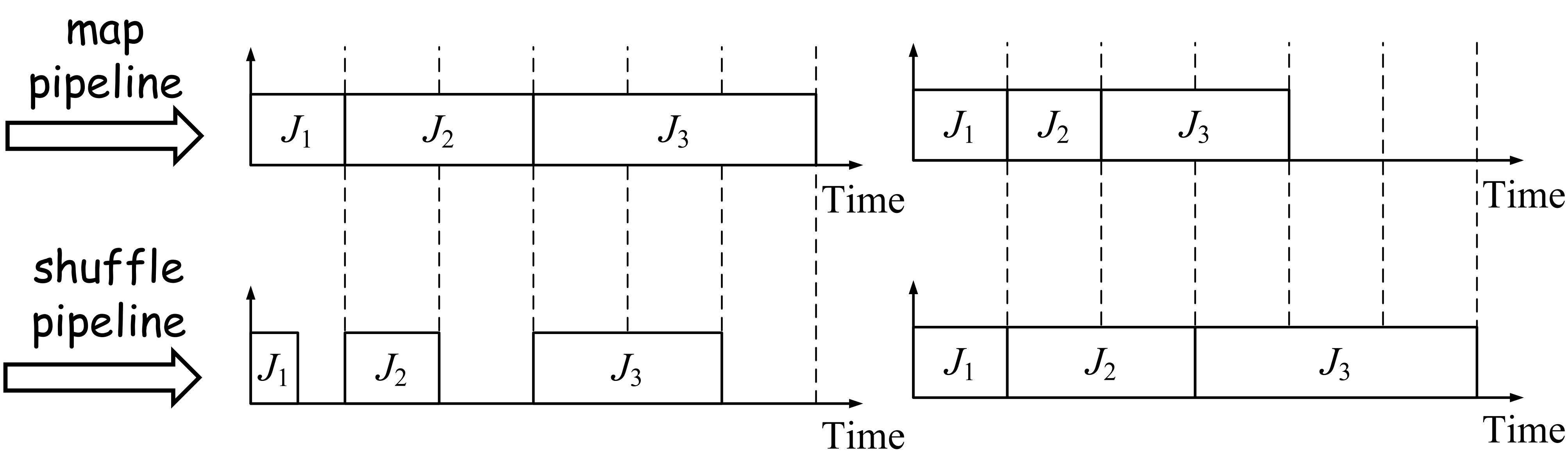
3. Observation and Ideas

When all jobs are map-heavy, balanced, or shuffle-heavy

Optimal schedule:

Sort job by dominant workload $\max(t_i^m, t_i^s)$

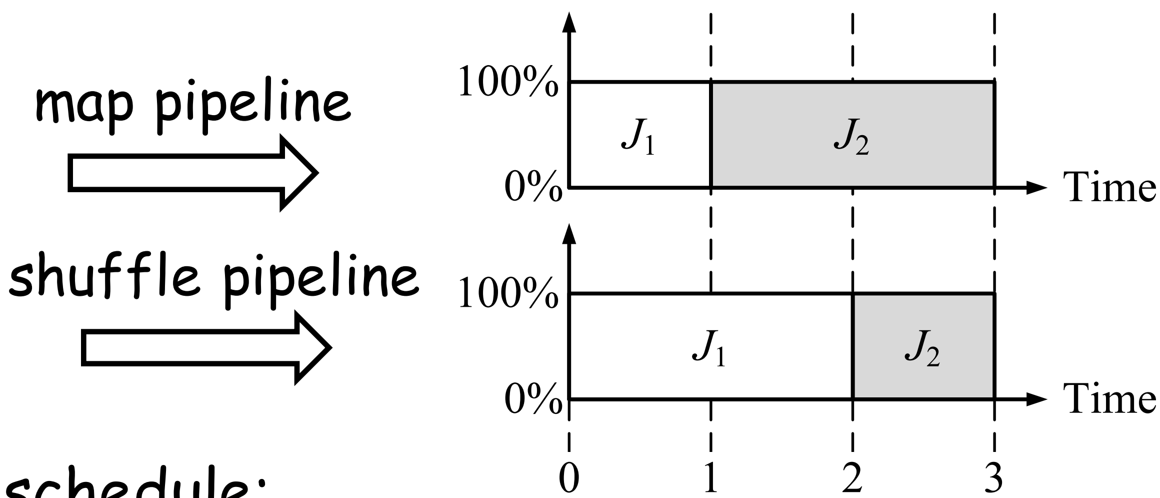
Smaller jobs are executed earlier



Perfect Pair

When jobs can be perfectly "paired"

Jobs J_i and J_j are paired, if $t_i^m = t_j^s$ and $t_i^s = t_j^m$



Optimal schedule:

- Pair jobs (shuffle-heavy before map-heavy)
- Sort job pair by total workload $t_i^m + t_i^s$
- Smaller pairs are executed earlier

Theorem

Theorem: If jobs can be perfectly paired, the optimal schedule pairwisely executes jobs in a pair.

- In each pair, shuffle-heavy job is executed before map-heavy job
- Job pairs with smaller total workloads are executed earlier

Proof:

In each pair, shuffle-heavy job is executed before map-heavy job

Otherwise a swap leads to a better result

Job pairs with smaller total workloads are executed earlier

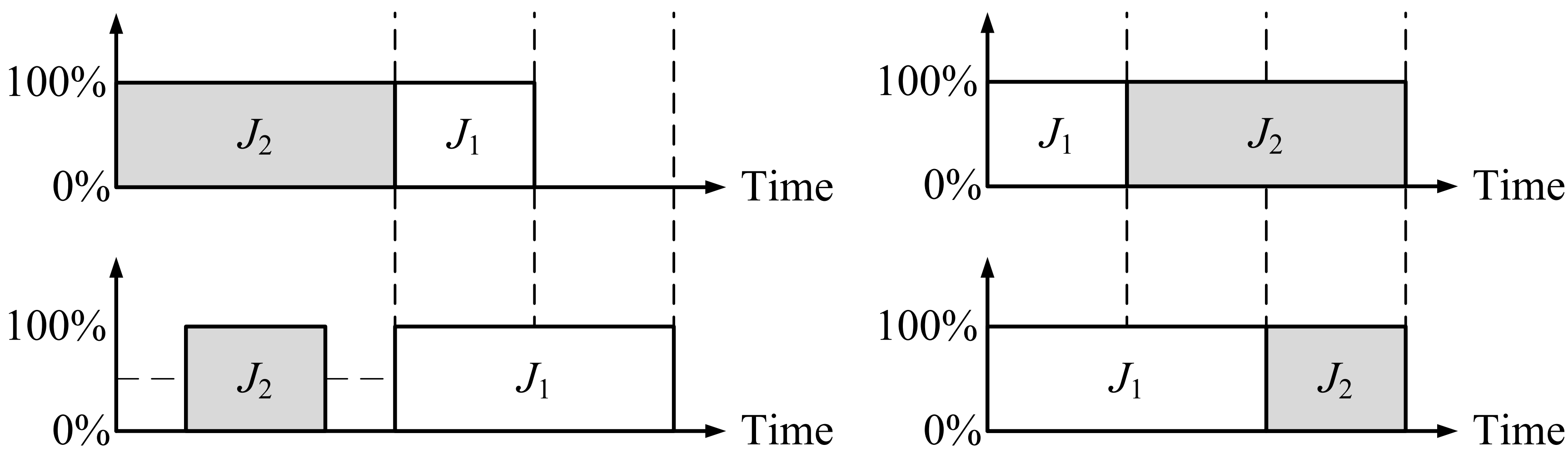
Otherwise a swap leads to a better result

Proof

Proof: jobs in a pair are executed together

Induction: shuffle-heavy J_1 and map-heavy J_2

Base case validates



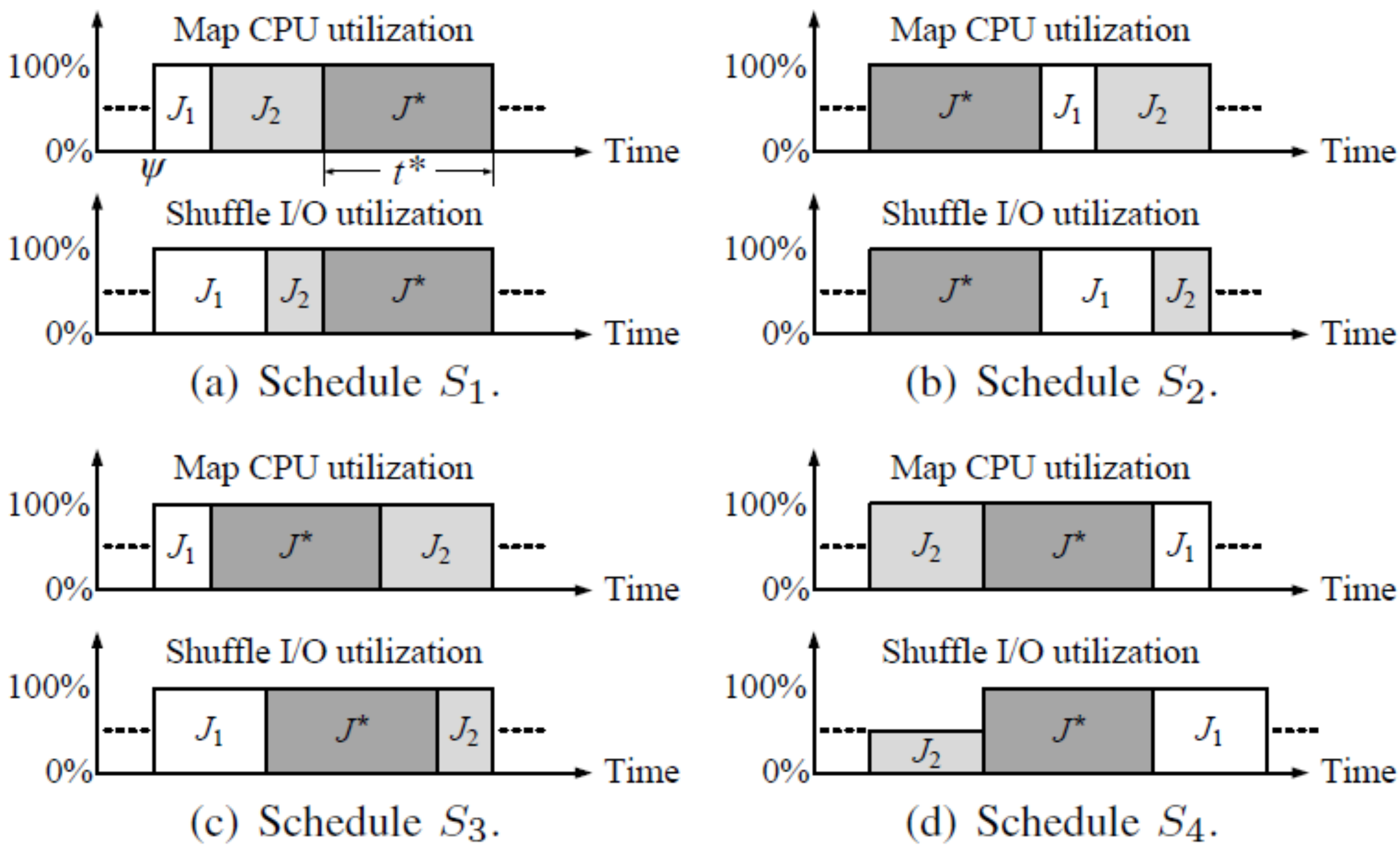
Suppose the theorem validates for J

Prove validation for J_1, J_2, and J

Theorem also holds for uniform data rate

Proof

Induction validates: the best schedule is S_1 or S_2



(a) Schedule S_1.

(b) Schedule S_2.

(c) Schedule S_3.

(d) Schedule S_4.

Two Insights

- Two scheduling factors for non-perfectly paired
  - Schedule smaller jobs first (dominant)
  - Jobs should be paired (non-dominant)

4. Algorithms

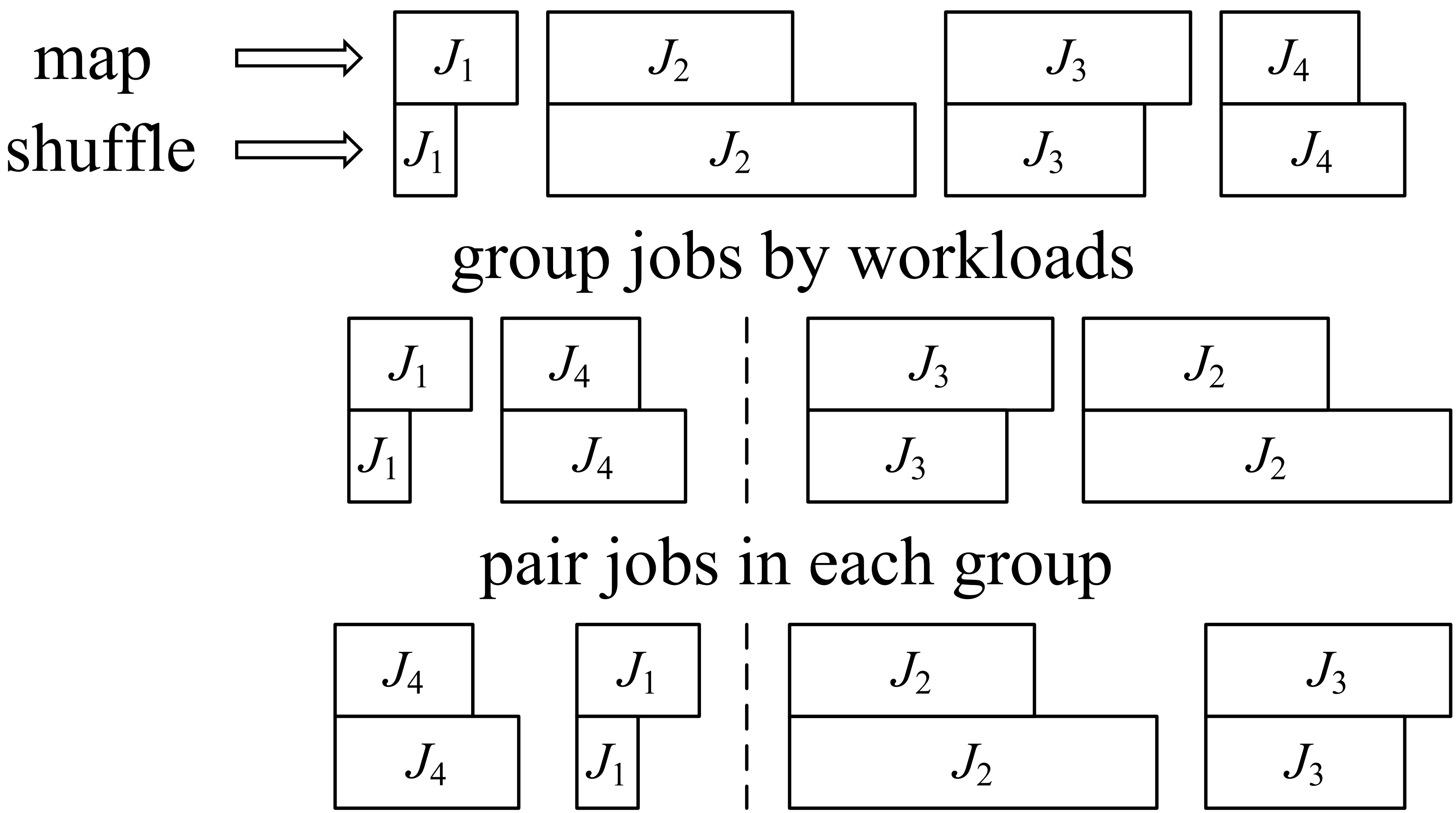
Two-stage scheduling algorithm

- Group jobs by their workloads (first factor)
  - Optimally divide jobs into k groups
    - Criterion: minimize the sum of maximum job workload difference within each group
  - Execute the group of smaller jobs earlier
- Job are paired in each group (second factor)
  - Jobs in each group have close workloads
  - Pair shuffle-heaviest and map-heaviest jobs:

$$J_i = \arg\max_i (t_i^s - t_i^m) \text{ and } J_j = \arg\max_j (t_j^m - t_j^s)$$

Algorithms

Example: two-stage scheduling algorithm (order only)



Algorithms

Dominant workload scheduling policy (DWSP)

Group jobs by dominant workloads, $\max(t_i^m, t_i^s)$

Performs well when jobs are simultaneously map-heavy, balanced, or shuffle-heavy

Total workload scheduling policy (TWSP)

Group jobs by total workloads, $t_i^m + t_i^s$

Performs well, when jobs can be perfectly paired

Weighted workload scheduling policy (WWSP)

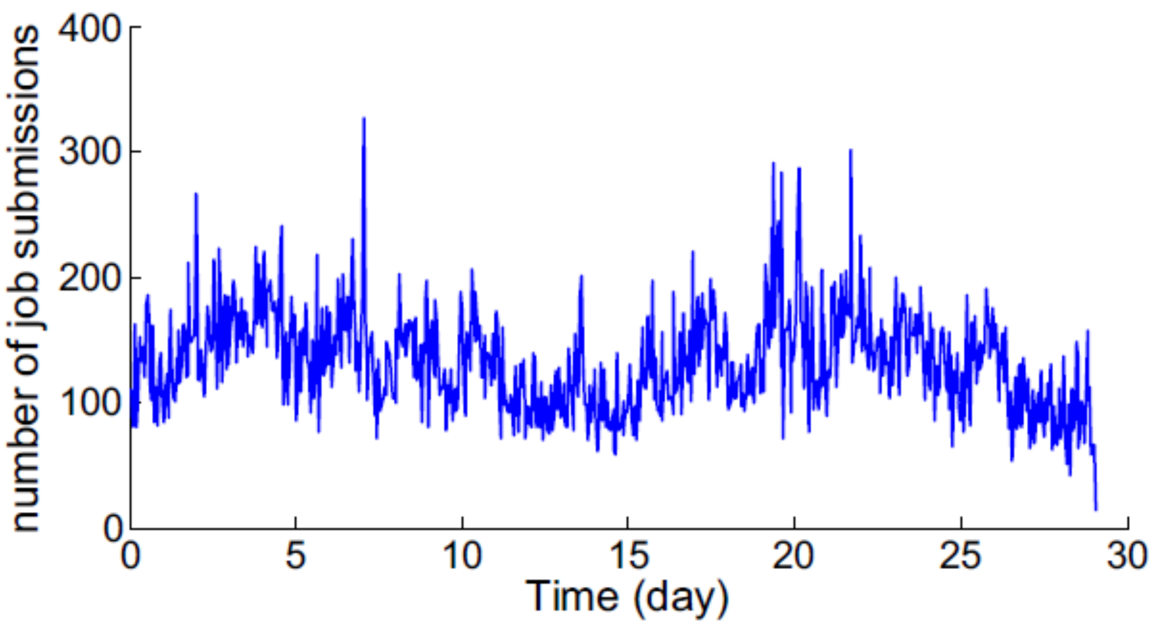
A tradeoff between pair-based and couple-based policies

Group jobs by weighted workloads $[\alpha \cdot \max(t_i^m, t_i^s) + (1-\alpha) \cdot (t_i^m + t_i^s)]$

5. Experiments

Google Cluster Dataset

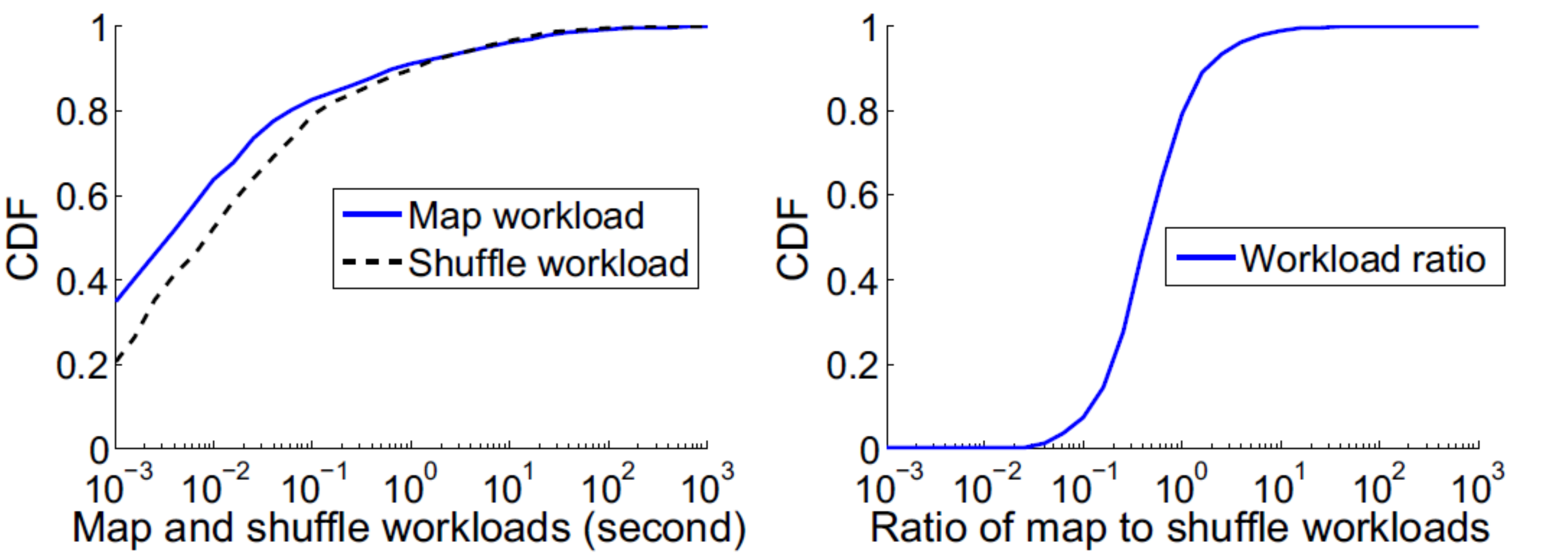
- About 11,000 machines
- 96,182 jobs over 29 days in May 2011 (time collapsed)
- Number of job submissions per hour (arrival rate)



Experiments

Google Cluster Dataset

Distribution of map and shuffle time



(a) Map and shuffle workloads. (b) Workload ratio distribution.

Experiments

Comparison algorithms

Pairwise: has only one group, then iteratively pairs the map-heaviest and shuffle-heaviest jobs in the group

MaxTotal: rank jobs by total workload $t_i^m + t_i^s$
smaller total workload is executed earlier

MaxSRPT: rank jobs by dominant workload $\max(t_i^m, t_i^s)$
smaller dominant workload is executed earlier

Experiments

Performance (group k = 20 and weight α = 0.5)

Scheduling algorithms	Average job waiting time	Average job execution time	Average job completion time
Pairwise	8289	149	8438
MaxTotal	5054	362	5416
MaxSRPT	4768	840	5608
DWSP	4809	581	5390
TWSP	4787	563	5350
WWSP	4619	532	5151

Improvement by considering both job workloads and pairs

Experiments

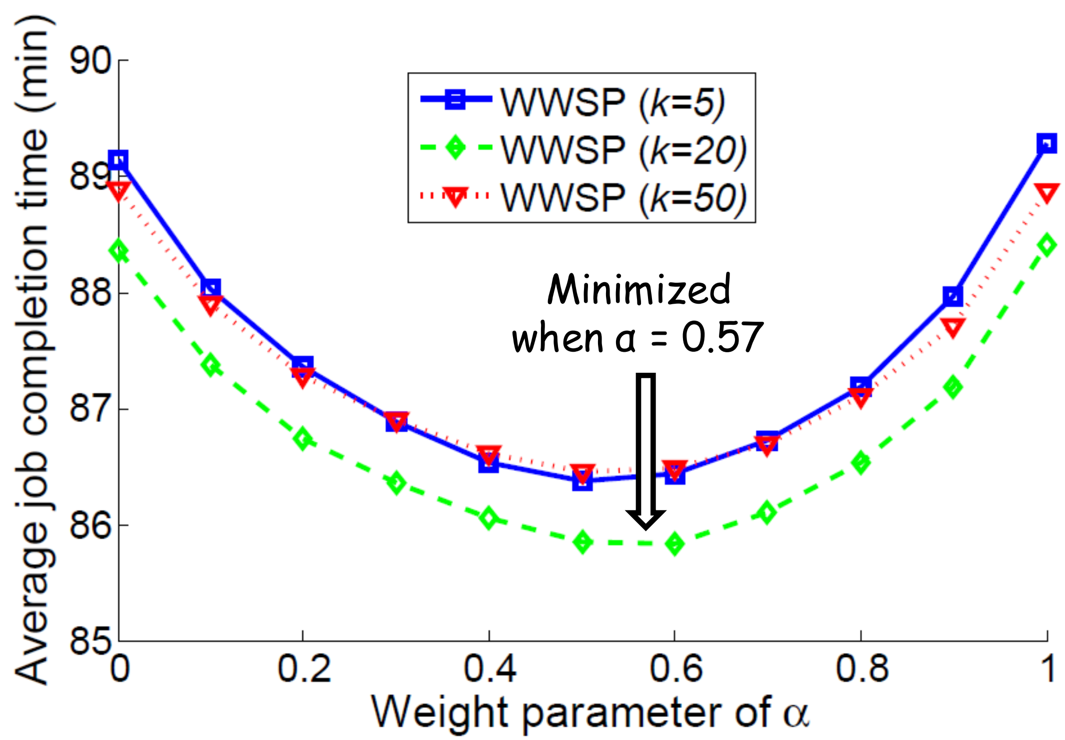
Impact of k and α

Group-based scheduling policy with k groups

Sort jobs by $[\alpha \cdot \max(t_i^m, t_i^s) + (1-\alpha) \cdot (t_i^m + t_i^s)]$

Small/Large group k

Small/large weight α



Simulation Summary

- Pairwise has the smallest average job execution time, but large job waiting time, since job workloads are ignored
- MaxTotal and MaxSPRT do not balance the trade-off between job size and job pair
- DWSP, TWSP, and WWSP jointly consider job sizes and job pairs

6. Conclusion

Map and Shuffle phases can overlap

- CPU and I/O resource

Objective: minimize average job completion time

Two-stage schedule

- Job workloads (dominant factor)
- Job pairs (avoid I/O underutilization)
- Optimality under certain scenarios