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 \textit{waits} for the data emitted by map
may be delayed by the scheduling policy

Job classification

\textbf{Map-heavy}: map workload > shuffle workload

\textbf{Balanced}: map workload = shuffle workload

\textbf{Shuffle-heavy}: map workload < shuffle workload
Introduction

Impact of overlapping map and shuffle phases

Word count (map-heavy) vs. Merge sort (shuffle-heavy)
2. Model and Formulation

Jobs in Map-Shuffle-Reduce

A set of n jobs:

\[ J = \{ J_1, J_2, \ldots, J_n \} \]

- \( t^m_i \) map workload of \( J_i \)
- \( t^s_i \) shuffle workload of \( J_i \)

Job classification:

- Map-heavy if \( t^m_i > t^s_i \)
- Shuffle-heavy if \( t^m_i < t^s_i \)
- Balanced if \( t^m_i = t^s_i \)
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: 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: 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
Induction validates: the best schedule is $S_1$ or $S_2$.
Two Insights

Two scheduling factors for non-perfectly paired jobs:

- 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:
  \[ J_i = \arg \max_i (t_i^s - t_i^m) \quad \text{and} \quad J_j = \arg \max_j (t_j^m - t_j^s) \]
Example: two-stage scheduling algorithm (order only)

map $\longrightarrow$ 
shuffle $\longrightarrow$

group jobs by workloads

pair jobs in each group
Algorithms

Dominant workload scheduling policy (DWSP)
Group jobs by dominant workloads, \( \max(t^m_i, t^s_i) \)
Performs well when jobs are simultaneously map-heavy, balanced, or shuffle-heavy

Total workload scheduling policy (TWSP)
Group jobs by total workloads, \( t^m_i + t^s_i \)
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 \[ \alpha \cdot \max(t^m_i, t^s_i) + (1-\alpha) \cdot (t^m_i + t^s_i) \]
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)

<table>
<thead>
<tr>
<th>Scheduling algorithms</th>
<th>Average job waiting time</th>
<th>Average job execution time</th>
<th>Average job completion time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairwise</td>
<td>8289</td>
<td>149</td>
<td>8438</td>
</tr>
<tr>
<td>MaxTotal</td>
<td>5054</td>
<td>362</td>
<td>5416</td>
</tr>
<tr>
<td>MaxSRPT</td>
<td>4768</td>
<td>840</td>
<td>5608</td>
</tr>
<tr>
<td>DWSP</td>
<td>4809</td>
<td>581</td>
<td>5390</td>
</tr>
<tr>
<td>TWSP</td>
<td>4787</td>
<td>563</td>
<td>5350</td>
</tr>
<tr>
<td>WWSP</td>
<td>4619</td>
<td>532</td>
<td>5151</td>
</tr>
</tbody>
</table>

Improvement by considering both job workloads and pairs
Experiments

Impact of $k$ and $\alpha$

Group-based scheduling policy with $k$ groups
Sort jobs by $[\alpha \cdot \max(t^m_i, t^s_i) + (1-\alpha) \cdot (t^m_i + t^s_i)]$

Small/Large group $k$
Small/large weight $\alpha$

Minimized when $\alpha = 0.57$
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