Joint Scheduling of Overlapping Phases in the MapReduce Framework

Collaborators: Huanyang Zheng and Yang Chen

Center for Networked Computing
Temple University
Road Map

1. Introduction
2. Model and Formulation
3. General Greedy Solutions
4. Experiment
5. Conclusion
1. Introduction

Map-Shuffle-Reduce
- Map and Reduce: CPU-intensive
- Shuffle: I/O-intensive

TeraSort
- Map: sample & partition data
- Shuffle: partitioned data
- Reduce: locally sort data
Map-Shuffle-Reduce

Multiple jobs
    TeraSort, WordCount, etc.

Reduce is not significant (Zaharia, OSDI 2008)
    7% of jobs are reduce-heavy

Centralized scheduler
    Determines a sequential order for jobs on the map and shuffle pipelines
Job Classification

Dependency relationship

Map *emits* data at a *certain* rate
Shuffle *waits* for the map data

Job classification

Map-heavy: \( \text{map} > \text{shuffle} \) \((m > s)\)
Balanced: \( \text{map} = \text{shuffle} \) \((m = s)\)
Shuffle-heavy: \( \text{map} < \text{shuffle} \) \((m < s)\)
Execution Order

Impact of overlapping map and shuffle

Map pipeline

Shuffle pipeline

WordCount (map-heavy)

TeraSort (shuffle-heavy)
2. Model and Formulation

Schedule objective:
Minimize the average job completion time for all jobs; \( J_i \) includes the wait time before the job starts.

Schedule is NP-hard

Pairing factor
Small job factor

Offline scenarios
All jobs arrive at the beginning (and wait for schedule)
Related Work: Flow Shop

Minimize last job completion time

1-phase flow shop is solvable when $l=2$
- $G_m$: map-heavy jobs sorted in increasing order of map load
- $G_s$: shuffle-heavy jobs sorted in decreasing order of shuffle load

Optimal schedule: $G_s$ followed by $G_m$

\[
\begin{array}{cccccc}
J_1 & J_2 & J_4 & J_1 & J_3 & J_4 \\
J_1 & J_2 & J_4 & J_1 & J_3 & 4
\end{array}
\]

Related Work: Strong Pair

Minimize average job completion time

Strong pair
- $J_1$ and $J_2$ are a strong pair if $m_1 = s_2$ and $s_1 = m_2$

Optimal schedule: jobs are strong pairs
Pair jobs and rank pairs by total workloads

First Special Case

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

Optimal schedule:
  Sort jobs ascendingly by dominant workload $\max\{m, s\}$
  Execute smaller jobs earlier

Finishing times $J_1, J_2, J_3$: 1, 3, 6 vs. $J_3, J_2, J_1$: 3, 5, 6
Second Special Case

Jobs $J_1$ and $J_2$ can be “paired”
if $m_1 \leq m_2$, $s_1 \geq s_2$, and $m_1+m_2=s_1+s_2$

(non-dominance) (balance)

Optimal schedule:
Pair jobs: shuffle-heavy before map-heavy
Sort job pair: by total workload $m+s$
Execute smaller pairs earlier
Why Non-dominance?

Cannot pair small and large jobs $J_1$ and $J_2$
If jobs can be paired, paired job scheduling is optimal if
(1) job pairs with smaller workloads are executed earlier and
(2) all pairs are executed together (shuffle-heavy first).

Proof ideas

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

Paired jobs should not be separately executed (a bit more involved)
$S_1$ is better than $S_3$ and $S_4$ when $J^*$ is large
$S_2$ is better than $S_3$ and $S_4$ when $J^*$ is small

Proof

(a) Schedule $S_1$.  
(b) Schedule $S_2$.  
(c) Schedule $S_3$.  
(d) Schedule $S_4$.  

Map CPU utilization

Shuffle I/O utilization
3. First General Algorithm

Sort jobs based on their sizes ("workload")

Partition sorted list in $k$ (group factor) groups

Execute each group in order based on workload
   Order matters for inter-group!

Pair jobs in each group
   Pairing matters for intra-group!

[Diagram showing group and working order]
Group-Based Scheduling Policy (GBSP)

Group jobs by their workloads (first factor)
Optimally divide jobs into \( k \) groups
minimize the sum of maximum job workload difference in each group
Execute the group of smaller jobs earlier

Pair jobs in each group (second factor)
Jobs in each group have similar workloads
Pair shuffle-heaviest and map-heaviest jobs

Time complexity is \( O(n^2k) \)
Example 1

Group-based scheduling policy

map $\iff$ 

shuffle $\iff$

group jobs by workloads

pair jobs in each group

schedule
**Workload Definition**

**Dominant workload scheduling policy (DWSP)**
- Groups jobs by dominant workloads, $\max(m, s)$
- Performs well when jobs are simultaneously map-heavy, balanced, or shuffle-heavy

**Total workload scheduling policy (TWSP)**
- Groups jobs by total workloads, $m+s$
- Performs well when jobs can be perfectly paired

**Weighted workload scheduling policy (WWSP)**
- A tradeoff between DWSP and TWSP
- Groups jobs by weighted workloads, $\alpha \cdot \max(m,s) + (1-\alpha) \cdot (m+s)$
Pair jobs through minimum weight maximum matching

Matching weight for $J_1$ and $J_2$:

$$\beta \times \text{balance factor} + (1-\beta) \times \text{non-dominance factor}$$

Balance factor: $\frac{|m_1 + m_2 - s_1 - s_2|}{m_1 + m_2 + s_1 + s_2}$

Non-dominance factor: $\mathbb{I}(m_1 - m_2)(s_1 - s_2) \geq 0$
Match-Based Scheduling Policy (MBSP)

Sort jobs by map-shuffle workload difference
Cut jobs into two parts
Use minimum weight maximum matching to pair jobs in the second part
Exhaust all possible cuts and pick the best cut
Sort jobs by their workloads after pairing
Paired jobs are regarded as one job

Map-dominant
<table>
<thead>
<tr>
<th>Paired jobs</th>
<th>Single jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort jobs by $m-s$ (↓ or ↑)</td>
<td></td>
</tr>
</tbody>
</table>

Shuffle-dominant

<table>
<thead>
<tr>
<th>Paired jobs</th>
<th>Single jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort jobs by $s-m$ (↓ or ↑)</td>
<td></td>
</tr>
</tbody>
</table>
Example 2

Match-based scheduling policy

map $\rightarrow$ 

shuffle $\rightarrow$

divide jobs into 2 groups

1\textsuperscript{st} group: shuffle-heavy

2\textsuperscript{nd} group: try to pair

sort jobs by dominant workload

schedule
Theorem

Match-based scheduling policy has an approximation ratio of 2 if

1. some jobs can be perfectly paired,
2. all remaining jobs are map-heavy, balanced, or shuffle-heavy,
3. dominant workload is used to sort jobs.

Time complexity is $O(n^{3.5})$

Exhausting all cuts takes $O(n)$ iterations
Matching in each iteration takes $O(n^{2.5})$
4. Experiment

Google Cluster Simulation
About 11,000 machines
96,182 jobs over 29 days in May 2011

Number of job submissions per hour (arrival rate)
Google Cluster Dataset

Distribution of map and shuffle time

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

Slightly more map-heavy jobs
Comparison Algorithms

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

**MaxTotal**: ranks jobs by total workload $m+s$ and executes jobs with smaller total workloads earlier

**MaxSRPT**: ranks jobs by dominant workload $\max\{m,s\}$ and executes jobs with smaller dominant workloads earlier
Waiting, Execution, and Completion

Results (group $k = 20$, weight $\alpha = 0.5$, $\beta = 0.5$)

<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>$\alpha/(\alpha + \beta)$</td>
<td>50% 75% 25%</td>
<td>50% 75% 25%</td>
<td>50% 75% 25%</td>
</tr>
<tr>
<td>Pairwise</td>
<td>8289 7652 3609</td>
<td>149 23 28</td>
<td>8438 7675 3637</td>
</tr>
<tr>
<td>MaxTotal</td>
<td>5054 4586 2525</td>
<td>362 32 156</td>
<td>5416 4618 2681</td>
</tr>
<tr>
<td>MaxSRPT</td>
<td>4768 4546 2591</td>
<td>840 32 150</td>
<td>5608 4578 2741</td>
</tr>
<tr>
<td>DWSP</td>
<td>4809 4519 2545</td>
<td>581 53 85</td>
<td>5390 4572 2630</td>
</tr>
<tr>
<td>TWSP</td>
<td>4787 4501 2522</td>
<td>563 49 104</td>
<td>5350 4550 2626</td>
</tr>
<tr>
<td>WWSP</td>
<td>4619 4482 2479</td>
<td>532 45 079</td>
<td>5151 4527 2558</td>
</tr>
<tr>
<td>MBSP</td>
<td>4562 4314 2142</td>
<td>193 26 36</td>
<td>4340 4755 2178</td>
</tr>
</tbody>
</table>

Control job waiting time using the workload of each group
Control job execution time by pairing jobs within a group

The average job completion time ratio between MBSP and WWSP is 92.3%, 95.8% and 85.1%, respectively.
Group-based scheduling policy with $k$ groups
Sorts jobs by $\alpha \cdot \max(m,s) + (1-\alpha)(m+s)$

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

Minimized when $\alpha = 0.57$
Impact of $\beta$ in MBSP

Match-based scheduling policy matches $J_1$ and $J_2$ by

$$\beta \times \text{balance factor} + (1-\beta) \times \text{non-dominance factor}$$

Small/large weight $\beta$

Minimized when $\beta = 0.68$
Hadoop Testbed on Amazon EC2

Testbed
  Ubuntu Server 14.04 LTS (HVM)
  Single core CPU and 8G SSD memory

Jobs: WordCount jobs and TeraSort jobs
  6 WordCount uses books of different sizes
    2MB, 4MB, 6MB, 8MB, 10MB, 12MB
  6 TeraSort uses instances of different sizes
    1KB, 10KB, 100KB, 1MB, 10MB, 100MB
**Waiting, Execution, and Completion**

**Hadoop:** one master node + several data nodes

Number of data nodes: 1, 2, 4, 8, 16

---

**MBSP** has a slightly larger job waiting time than **WWSP**, but a smaller job makespan.
Performance Comparison

Pairwise has the smallest average execution time, but a large job wait time since workloads are ignored.

MaxTotal and MaxSPRT do not balance the trade-off between job sizes and job pairs.

DWSP, TWSP, WWSP, and MBSP jointly consider job sizes and job pairs.
5. Conclusion

Map and Shuffle phases can overlap
   CPU and I/O resource

Objective: minimize average job completion time

Group-based and match-based schedules
   Job workloads (dominant factor)
   Job pairs (avoid I/O underutilization)
   Optimality under certain scenarios
Future Work

Multiple phases
  Beyond 2-phase

Batched online scheduling
  Window-based approach

More simulations
  Imbalanced map and shuffle
  Impact of $k$, $\alpha$, and $\beta$

More testbed cases