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: a popular computation paradigm

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

Master node: pipeline scheduling
Data node: data parallelism

E.g., TeraSort
Map: sample & partition data
Shuffle: move data
Reduce: locally sort data
Scheduling of Multiple Jobs

Multiple jobs
Terasort, wordcount, ...

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 pipeline
Job Classification

Dependency relationship

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

Job classification

Map-heavy: \( \text{map} \geq \text{shuffle} \) \((m \geq s)\)
Shuffle-heavy: \( \text{map} \leq \text{shuffle} \) \((m \leq s)\)
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 offline and APX-hard online (Lin 2013)

Offline

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

Minimize last job completion time

$l$-phase flow shop is solvable when $l=2$

$G_s$: shuffle-heavy jobs sorted in decreasing order of shuffle load
$G_m$: map-heavy jobs sorted in increasing order of map load

Optimal schedule: $G_s$ followed by $G_m$

**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 $O(n \log n)$:

Sort jobs ascendingly by dominant workload $\max\{m, s\}$

Execute small jobs first

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 $m_1 \leq m_2$, $s_1 \geq s_2$ (non-dominance), and $m_1 + m_2 = s_1 + s_2$ (balance)

Optimal schedule $O(n \log n)$:

- Pair jobs: head to tail pairing on sorted jobs based on $m-s$
- Sort job pairs: by total workload $m+s$
- Execute sorted job pairs: smallest pair first
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 (with shuffle-heavy first).

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
Proof

$S_1$ 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

(a) Schedule $S_1$.

(b) Schedule $S_2$.

(c) Schedule $S_3$.

(d) Schedule $S_4$. 
3. General Greedy Solutions

A delicate balance for general cases

\[
\begin{array}{cc}
\text{Pairing factor} & \text{Small job factor} \\
\end{array}
\]

Map-dominant (shuffle-dominant)

more map (shuffle)-heavy than shuffle (map)-heavy
First Greedy 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!

\[ \begin{array}{c|c|c|c|c|c} \text{Group} & \text{Working order} & \text{1} & \text{2} & \text{......} & \text{k-1} \\ \end{array} \]
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: GBSP

map $\rightarrow$ 
shuffle $\rightarrow$

group jobs by workloads

pair jobs in each group

schedule
Workload Definition

Dominant workload scheduling policy (D\textsc{WSP})

Groups jobs by dominant workloads, \( \text{max} \{m, s\} \)
Performs well when jobs are simultaneously map-heavy, balanced, or shuffle-heavy

Total workload scheduling policy (T\textsc{WSP})

Groups jobs by total workloads, \( m+s \)
Performs well when jobs can be perfectly paired

Weighted workload scheduling policy (W\textsc{WSP})

A tradeoff between D\textsc{WSP} and T\textsc{WSP}
Groups jobs by weighted workloads, \( \alpha*\text{max}\{m,s\} + (1-\alpha)*(m+s) \)
Second Greedy Algorithm

Sort jobs by map-shuffle workload difference

Cut jobs into two parts
Use minimum weight maximum matching to pair jobs in the first part
Exhaust all possible cuts and pick the best cut
Sort jobs by their workloads after pairing, together with single jobs
Paired jobs are regarded as one job

Map-dominant
\[ \text{Cut} \]
\[ \text{Paired jobs} \quad \text{Single jobs} \]
Sort jobs by \( m-s \) (\( \downarrow \) or \( \uparrow \))

Shuffle-dominant
\[ \text{Cut} \]
\[ \text{Paired jobs} \quad \text{Single jobs} \]
Sort jobs by \( s-m \) (\( \downarrow \) or \( \uparrow \))
**Match-Based Scheduling Policy (MBSP)**

Pair jobs through minimum weight maximum matching

Matching weight for $J_1$ and $J_2$:

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

**Non-dominance factor:**

$$\mathbb{1}_{(m_1 - m_2)(s_1 - s_2) \geq 0}$$

**Balance factor:**

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

---

![Diagram of job pairs](image-url)
Theorem

MBSP has an approximation ratio of 2 if
(1) some jobs can be perfectly paired,
(2) all remaining jobs are map-heavy 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 $O(n^{2.5})$

(Blossom algorithm, 1961)
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**

Group $k = 20$, $\alpha = 0.5$, $\beta = 0.5$, Col 1$^{st}$ regular, 2$^{nd}$ shuffle half, 3$^{rd}$ map half

<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 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 79</td>
<td>5151 4527 2558</td>
</tr>
<tr>
<td>MBSP</td>
<td>4562 4314 2142</td>
<td>193 26 36</td>
<td>4754 4340 2178</td>
</tr>
</tbody>
</table>

The average job completion time ratio between MBSP and WWSP is 92.3%, 95.8% and 85.1%, respectively.
Impact of $k$ and $\alpha$ in WWSP

Group-based scheduling policy with $k$ groups
Sorts jobs by $\alpha \times \max(m,s) + (1-\alpha) \times (m+s)$

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 use books of different sizes
2MB, 4MB, 6MB, 8MB, 10MB, 12MB

6 TeraSort use instances of different sizes
1KB, 10KB, 100KB, 1MB, 10MB, 100MB
Completion Time

Hadoop: one master node + several data nodes

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

MBSP and WWSP have the best results
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

Optimality under certain scenarios
Pairing factor
Small jobs factor
Future Work

More simulations
Imbalanced map and shuffle
impact of k, α, and β

Multiple phases
Beyond 2-phase

Other computation paradigms
Map-collective

3-phase example

Map
Shuffle
Reduce
Future Work

Online scheduling

Batched
Batch size

Duration-based batching
Low-job rate: time out
High-job rate: probabilistic
$\Delta$: efficient, but slow; $1-\Delta$: inefficient, but fast

Counting-based batching
Low-job rate: time out
High-job rate: credit
Scheduling time vs. execution time