Preemptive ReduceTask Scheduling for Fair and Fast Job Completion

Yandong Wang, Jian Tan, Weikuan Yu,

Li Zhang, Xiaoqiao Meng

IBM Research
Outline

- **Background & Motivation**
- Issues in Hadoop Scheduler
- Preemptive ReduceTask
- Fair Completion Scheduler
- Performance Evaluation
- Conclusion and Future Work
Overview

- **MapReduce** is a programming model for processing massive-scale data.
  - Hadoop: Open-source implementation of MapReduce

- **Hadoop has been widely adopted by leading companies.**
  - Providing high scalability and strong fault tolerance.

- **Data consolidation can be highly beneficial.**
  - Co-location of disparate data sets and avoiding data replication cost.

- **Mixed workloads of long batch jobs and small interactive queries.**
  - Interactive queries are expected to return quickly.
  - Hadoop Fair Scheduler was introduced to allow fair sharing among concurrent jobs.
Hadoop schedulers strive to overlap the map and shuffle phases to accelerate data processing pipeline.
Hadoop Fair Scheduler

• A widely used Hadoop scheduler for sharing a Hadoop cluster.

• Providing fairness among concurrently running jobs via max-min fair sharing.
  • Delay scheduling policy are used to provide data locality awareness.

• Tasks occupy slots until successful completion or failure.
Outline

- Background & Motivation
- Issues in Hadoop Scheduler
- Preemptive ReduceTask
- Fair Completion Scheduler
- Performance Evaluation
- Conclusion and Future Work
Unfair Reduce Slots Allocation

- Monopolizing behavior of long ReduceTasks from the large job (Job3).
- On average, last 5 small jobs are severely slowed down by $15\times$.

![Diagram showing Map and Reduce slot usage over time, with Job4 suffering from slowed down by $19\times$.](diagram.png)
**Distinct Execution Pattern between Map and Reduce Tasks**

- Current Hadoop schedulers treat map and reduce tasks similarly.

<table>
<thead>
<tr>
<th>Distinctions</th>
<th>MapTask</th>
<th>ReduceTask</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution Time</strong></td>
<td>Short-lived</td>
<td>Long-lived</td>
</tr>
<tr>
<td><strong>Execution Phase</strong></td>
<td>Single-phase</td>
<td>Multi-phase</td>
</tr>
<tr>
<td><strong>Execution Dependency</strong></td>
<td>None</td>
<td>Map phase</td>
</tr>
</tbody>
</table>
## Distinct Execution Pattern between Map and Reduce Tasks

- Current Hadoop schedulers treat map and reduce tasks similarly.

<table>
<thead>
<tr>
<th>Distinctions</th>
<th>MapTask</th>
<th>ReduceTask</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Phase</td>
<td>Single phase</td>
<td>Multi-phase</td>
</tr>
<tr>
<td>Execution Dependency</td>
<td>None</td>
<td>Map phase</td>
</tr>
</tbody>
</table>

*It is critical for Hadoop schedulers to be aware of these different patterns.*
Existing Efforts

- **Hadoop introduces slow start** [1]
  - Mitigating the starvation but at the cost of slowing down the data processing pipeline.
  - Impacting the execution time of small jobs.
- **Coupling scheduling policy from IBM** [2]
  - Similar to slow start which let monopolization progressively happen
- **Copy-Compute Splitting** [3]
  - Performance is unknown, no results was reported.

---

[1]: “mapred.reduce.slowstart.completed.maps”.

[2]: Jian Tan, Xiaoqiao Meng, Li Zhang, “Coupling scheduler for MapReduce/Hadoop”, HPDC’12.

Fundamental Solutions

**How to achieve both high Efficiency and Fairness?**

- **How to tackle monopolizing behavior of long running ReduceTasks?**
  - Existing schedulers ignore long-lasting ReduceTasks, once they are launched, they occupy resource until completion or failure.
  - Introducing a new mechanism: Preemptive ReduceTask.

- **How to coordinate two-phase job scheduling?**
  - MapReduce adopts two-phase scheme (map and reduce) to schedule tasks. However less contemplation has been given to coordinate them.
  - A new scheduler: Fair Completion Scheduler.
Outline

• Background & Motivation
• Issues in Hadoop Scheduler
• Preemptive ReduceTask
• Fair Completion Scheduler
• Performance Evaluation
• Conclusion and Future Work
Preemptive ReduceTask

- **Lightweight work-conserving preemption mechanism.**
  - Preserving previous computation and I/O.
  - Providing lightweight preemption with no noticeable performance impact.

- **Different from Linux process suspend commend ("Kill -STOP $PID").**
  - Preemptive ReduceTask releases the reduce slot.

- **Superior to current killing preemption mechanism.**
  - Killing can lead to significant waste of computation and I/O.
Preemption During Shuffle Phase

- Only merging the in-memory intermediate data, while maintaining on-disk intermediate data untouched.

\[ R_1: \text{Before Preempt} \]

\[ R_1: \text{After Resume} \]
Preemption During Reduce Phase

- Recording the current offset of each segment and minimum priority queue
- Preemption occurs at the boundary of intermediate <key,value> pairs.

**R1: Before Preempt**

**R1: After Resume**

- 
- Index
- TaskTracker

**Offset**

**Flush**

**Retrieve**

**MPQ**

**DFS**
Evaluation of Preemptive ReduceTask

- Terasort benchmark with 512GB input data on a cluster of 20 worker nodes.
Outline

• Background & Motivation
• Issues in Hadoop Scheduler
• Preemptive ReduceTask
• Fair Completion Scheduler
• Performance Evaluation
• Conclusion and Future Work
Fair Completion Scheduler

- Prioritizing ReduceTasks from jobs with the shortest remaining map phases.
  - Allowing small jobs to preempt long-running ReduceTasks from large jobs.
  - MapTask scheduling follows max-min fair sharing policy.

- When remaining map phases are equal, prioritizing ReduceTasks from jobs with least remaining reduce data.

- Detecting the job execution slowdown caused by preemptions.
  - Preventing ReduceTasks of large jobs from being preempted for too long and too many times.
Fair Completion Scheduling Details

<table>
<thead>
<tr>
<th></th>
<th>Remaining Map Phase</th>
<th>Remaining Reduce Data</th>
<th>Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>1000 s</td>
<td>100GB</td>
<td>6</td>
</tr>
<tr>
<td>$J_2$</td>
<td>200 s</td>
<td>10GB</td>
<td>2</td>
</tr>
</tbody>
</table>

Sort Running Jobs:
(1): According to remaining map time
(2): According to remaining reduce data

Slave Node 1
- $R_1$ of $J_1$
- $R_2$ of $J_1$

Slave Node 2
- $R_3$ of $J_1$
- $R_1$ of $J_2$

Slave Node 3
- $R_4$ of $J_1$
- $R_5$ of $J_1$

Slave Node 4
- $R_6$ of $J_1$
- $R_2$ of $J_2$
## Fair Completion Scheduling Details

<table>
<thead>
<tr>
<th>Job</th>
<th>Remaining Map Phase</th>
<th>Remaining Reduce Data</th>
<th>Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>1000 s</td>
<td>100GB</td>
<td>6</td>
</tr>
<tr>
<td>$J_2$</td>
<td>200 s</td>
<td>10GB</td>
<td>2</td>
</tr>
<tr>
<td>$J_3$</td>
<td>80 s</td>
<td>8GB</td>
<td>4</td>
</tr>
</tbody>
</table>

### Sort Running Jobs:
(1): According to remaining map time
(2): According to remaining reduce data

---

- **Slave Node 1**
  - $R_1$ of $J_1$
  - $R_2$ of $J_1$

- **Slave Node 2**
  - $R_3$ of $J_1$
  - $R_1$ of $J_2$

- **Slave Node 3**
  - $R_4$ of $J_1$
  - $R_5$ of $J_1$

- **Slave Node 4**
  - $R_6$ of $J_1$
  - $R_2$ of $J_2$
Fair Completion Scheduling Details

Sort Running Jobs:
(1): According to remaining map time
(2): According to remaining reduce data

<table>
<thead>
<tr>
<th>Job</th>
<th>Remaining Map Phase</th>
<th>Remaining Reduce Data</th>
<th>Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>1000 s</td>
<td>100GB</td>
<td>6</td>
</tr>
<tr>
<td>$J_2$</td>
<td>200 s</td>
<td>10GB</td>
<td>2</td>
</tr>
<tr>
<td>$J_3$</td>
<td>80 s</td>
<td>8GB</td>
<td>4</td>
</tr>
</tbody>
</table>

Slave Node 1: $R_1$ of $J_1$, $R_2$ of $J_1$
Slave Node 2: $R_2$ of $J_1$, $R_3$ of $J_1$
Slave Node 3: $R_4$ of $J_1$, $R_5$ of $J_1$
Slave Node 4: $R_6$ of $J_1$, $R_2$ of $J_2$

Preempt ReduceTask of Job1
Fair Completion Scheduling Details

<table>
<thead>
<tr>
<th></th>
<th>Remaining Map Phase</th>
<th>Remaining Reduce Data</th>
<th>Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>1000 s</td>
<td>100GB</td>
<td>6</td>
</tr>
<tr>
<td>$J_2$</td>
<td>200 s</td>
<td>10GB</td>
<td>2</td>
</tr>
<tr>
<td>$J_3$</td>
<td>80 s</td>
<td>8GB</td>
<td>4</td>
</tr>
</tbody>
</table>

Sort Running Jobs:
1. According to remaining map time
2. According to remaining reduce data

Launch Reduce Tasks of Job 3

Slave Node 1
- $R_1$ of $J_1$
- $R_2$ of $J_1$
- $R_1$ of $J_3$

Slave Node 2
- $R_2$ of $J_3$

Slave Node 3
- $R_1$ of $J_2$
- $R_4$ of $J_3$

Slave Node 4
- $R_6$ of $J_1$
- $R_2$ of $J_2$

Job 1
- Job $J_1$

Job 2
- Job $J_2$

Job 3
- Job $J_3$

Launch Reduce Tasks of Job 3
Fair Completion Scheduling Details

<table>
<thead>
<tr>
<th>Job</th>
<th>Remaining Map Phase</th>
<th>Remaining Reduce Data</th>
<th>Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>800 s</td>
<td>100GB</td>
<td>6</td>
</tr>
<tr>
<td>$J_2$</td>
<td>120 s</td>
<td>10GB</td>
<td>2</td>
</tr>
</tbody>
</table>

Sort Running Jobs:
(1): According to remaining map time
(2): According to remaining reduce data

Job $J_1$ completes

Resume ReduceTasks of Job 1

Slave Node 1
- $R_1$ of $J_1$
- $R_2$ of $J_1$

Slave Node 2
- $R_3$ of $J_1$
- $R_1$ of $J_2$

Slave Node 3
- $R_4$ of $J_1$
- $R_5$ of $J_1$

Slave Node 4
- $R_6$ of $J_1$
- $R_2$ of $J_2$
Outline

• Background & Motivation
• Issues in Hadoop Scheduler
• Preemptive ReduceTask
• Fair Completion Scheduler
• Performance Evaluation
• Conclusion and Future Work
Testbed and Benchmarks/Metrics

- **Hardware configuration**
  - A cluster of 46 nodes. 4 2.67GHz hex-core Intel Xeon CPUs, 24GB memory and two hard disks.

- **Software configuration:**
  - Hadoop 1.0.0 and its Fair Scheduler. 8 map slots and 4 reduce slots on each node.

- **Gridmix2 and Tarazu benchmarks:**
  - Map-heavy workload
  - Reduce-heavy workload
  - Scalability evaluation
Results for Map-heavy Workload

- FCS reduces average execution time by 31% (171 jobs).

- Significantly speeds up small jobs, slightly slow down large jobs.
Small jobs are benefited from significantly shortened reduce wait time.

Waiting time are reduced by 22× for the jobs in the first 6 groups.
Preemption Frequency

- FCS controls the preemption frequency to avoid excessive preemptions.
FCS improves the fairness by 66.7% on average.

Achieving nearly uniform maximum slowdown for all groups of jobs.
Results for Reduce-heavy Workload

- FCS reduces average execution time by 28% (171 jobs).
- FCS accelerates all types of jobs in the reduce-heavy workload.
  - Impact of preemption on large job is not heavy due to they are still in map phases.
Fairness of Reduce-heavy Workload

• FCS improves the fairness by 35.2% on average.

Maximum Slowdown

10 Groups of Jobs

- Fair Completion
- Hadoop Fair
Scalability Evaluation with GridMix-2

- FCS reduces the average execution time by 39.7%.

- Small improvement at 60 due to dominant number of small jobs.
Outline

- Background & Motivation
- Issues in Hadoop Scheduler
- Preemptive ReduceTask
- Fair Completion Scheduler
- System Evaluation
- Conclusion and Future Work
Conclusion and Future Work

• Identify the inefficiencies in existing Hadoop schedulers.

• Preemptive ReduceTask provides an efficient preemption approach.

• Fair Completion Scheduler is introduced to improve the efficiency and fairness of the concurrently running jobs.

• Preemptive ReduceTask provides opportunities to improve the fault tolerance mechanism.

• More preemptive scheduling policy can be implemented based on Preemptive ReduceTask.
Sponsors of Our Research
Thank You and Questions?