iShuffle: Improving Hadoop Performance with Shuffle-on-Write

YANFEI GUO, JIA RAO, XIAOBO ZHOU

CS DEPARTMENT, UNIVERSITY OF COLORADO, COLORADO SPRINGS

PRESENTED BY YANFEI GUO
MapReduce

A framework for processing parallelizable problems across huge data sets using a large number of machines

- Invented and used by Google [OSDI’04]
- Many implementations
  - Apache Hadoop, Dryad
- From interactive query to massive/batch computation
  - Nutch, Hive, HBase
MapReduce Model

Apply a common function to the problem’s input

Generate intermediate data

Process intermediate data for answer

Map
\[ Map(k_1, v_1) \rightarrow list(k_2, k_2) \]

Reduce
\[ Reduce(k_2, list(v_2)) \rightarrow list(v_3) \]
MapReduce

Programming and Execution Model

\[ \text{Map} (k_1, v_1) \rightarrow \text{list} (k_2, k_2) \]

\[ \text{Reduce} (k_2, \text{list} (v_2)) \rightarrow \text{list} (v_3) \]
Hadoop Implementation

Map
- Buffered output
- Spill to disk

Reduce

Input Split

Mapper Map() → Partitioner → Output Buffer

SpillThread → Combiner

OutputCollector Collect() → Output

<k, v> → Map Task

<table>
<thead>
<tr>
<th>P1</th>
<th>k1</th>
<th>v1, v2, v3 ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>k7</td>
<td>v1, v2, v3 ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>k8</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>k9</td>
<td>v1, v2, v3 ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>k10</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

7/3/2013 ICAC'13 ISHUFFLE
Hadoop Key Designs

**Shuffle**
- All-to-all input data fetching phase in a reduce task
- The reduce function will not be performed until its completion
- Disk I/O and network intensive

**Overlapping shuffle with map tasks**
- Hadoop allows an early start of the shuffle phase as soon as part of the reduce input is available
- By default, shuffle is started when 5% of map tasks finished

**Fair sharing**
- Hadoop enforces fairness among users/jobs
- Fair share of map and reduce slots
Issues

Input data skew among reduce tasks
- Non-uniform key distribution $\Rightarrow$ Different partition size
- Lead to disparity in reduce completion time

Inflexible Scheduling of Reduce Tasks
- Reduce tasks are created during job initialization
- Tasks are scheduled in ascending order of their ID
- Reduce tasks can not start even if all their input partitions are available

Tight coupling of shuffle and reduce
- Shuffle starts only when the corresponding reduce is scheduled
- Leaving parallelism within and between jobs unexploited
A Motivating Example

**Workload:** tera-sort with 4GB dataset  
**Platform:** 10-node Hadoop cluster  
1 map and 1 reduce slots per node
Related Work

Map Scheduling in Hadoop
- Accelerating straggler Task: [OSDI’08]
- Enforcing Fairness: [Middleware’10], [EuroSys’10]

Improving reduce performance
- Push-based shuffling: [NSDI’10]
- RDMA-based acceleration: [SC’11]
- Specially designed partitioner: [TPDS’12]

Not applicable to reduce tasks

Requiring hardware support or not effective in multiple wave execution
Our Approach

Decouple shuffle phase from reduce tasks
- Shuffle as a platform service provided by Hadoop
- Pro-actively and deterministically push map output to different slave nodes

Balancing the partition placement
- Predict partition sizes during task execution
- Determine which node should a partition been shuffled to
- Mitigate data skew

Flexible reduce task scheduling
- Assign partitions to reduce tasks only when scheduled
iShuffle Design

iShuffle
- Shuffler
- Shuffle Manager
- Task Scheduler

Features
- User-Transparent Shuffle Service
- Shuffle-on-Write
- Automated Map Output Placement
- Flexible Reduce Task Scheduling
Shuffle-on-Write

“shuffle” when Hadoop stores intermediate results

Map output collection
- MapOutputCollector
- DataSpillHandler

Data shuffling
- Queuing and Dispatching
- Data Size Predictor
- Shuffle Manager

Map output merging
- Merger
- Priority-Queue merge sort
Partition Placement

Prediction of Partition Sizes

- Task characteristics: input data size, map selectivity
- Linear model between partition size and input data size
- Metrics measured during the task execution

\[ p_{i,j} = a_j + b_j D_i \]

Partition Placement Problem

- Minimizes the difference of total partition size on different nodes

\[ \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\mu - \sum_{j \in s_i} p_j)} \]
Heuristic Placement Algorithm

Largest Partition First (LPF)

◦ Pick the largest partition first
◦ Place it to node with the least total partition size

```
Data: p: list of partition
Data: S: list of nodes, has the size of all allocated partitions
Result: Balanced partition placement
sort list p in descending order of partition sizes;
for i ← 1 to m do
    min_node ← S[1];
    for j ← 1 to n do
        if S[j].size < min_node.size then
            min_node ← S[j];
        end
    end
    min_node.place(p[i]);
end
```
Flexible Reduce Scheduling

Assign Partitions to Reduce Tasks at Runtime
- Override the partition assignment at job initialization
- Allow tasks to run on any node

Multiple Job Scheduling
- Fair scheduling for map tasks
- Disabled fair share for reduce tasks
- Prevent wasted cluster cycles for waiting unfinished maps
Experiments

32-node Hadoop Cluster
  ◦ 1 namenode, 1 jobtracker, 30 slave nodes
  ◦ 4 map slots and 2 reduce slots per slave
  ◦ HDFS Block size = 64 MB
  ◦ Hadoop version 1.1.1

Hardware
  ◦ Intel Xeon E5530, 4-core, 2.4 GHz
  ◦ 4 GB Memory
  ◦ 1 Gbps Ethernet
## Benchmark

Purdue MapReduce Benchmark Suite (PUMA)
- Real data set from Wikipedia, Netflix
- Shuffle-heavy and shuffle-light

<table>
<thead>
<tr>
<th>Job</th>
<th>Input Size (GB)</th>
<th># Map</th>
<th># Reduce</th>
<th>Shuffle Vol (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-join</td>
<td>250</td>
<td>4000</td>
<td>180</td>
<td>246</td>
</tr>
<tr>
<td>Tera-sort</td>
<td>300</td>
<td>4800</td>
<td>180</td>
<td>300</td>
</tr>
<tr>
<td>Ranked-inverted-index</td>
<td>220</td>
<td>3520</td>
<td>180</td>
<td>235</td>
</tr>
<tr>
<td>K-means</td>
<td>30</td>
<td>480</td>
<td>6</td>
<td>43</td>
</tr>
<tr>
<td>Inverted-index</td>
<td>250</td>
<td>4000</td>
<td>180</td>
<td>57</td>
</tr>
<tr>
<td>Term-vector</td>
<td>250</td>
<td>4000</td>
<td>180</td>
<td>59</td>
</tr>
<tr>
<td>wordcount</td>
<td>250</td>
<td>4000</td>
<td>180</td>
<td>49</td>
</tr>
<tr>
<td>Histogram-movies</td>
<td>200</td>
<td>3200</td>
<td>180</td>
<td>0.002</td>
</tr>
<tr>
<td>Histogram-ratings</td>
<td>200</td>
<td>3200</td>
<td>180</td>
<td>0.0012</td>
</tr>
<tr>
<td>Grep</td>
<td>250</td>
<td>4000</td>
<td>180</td>
<td>0.0013</td>
</tr>
</tbody>
</table>
iShuffle Performance

Execution Trace

- Slow start of Hadoop does not eliminate shuffle delay for multiple reduce wave
- Overhead of remote disk access of Hadoop-A [SC’11]
- iShuffle has almost no shuffle delay
iShuffle Performance (cont’d)

Reducing Job Completion Time
- **30%** and **21%** less than vanilla Hadoop and Hadoop-A

Reducing Shuffle Delay
- **10x** less than vanilla Hadoop in job’s with large shuffle volume
- **2x** to **3x** less than Hadoop-A
Balanced Partition Placement

Performance improvement by a Balanced Partition Placement

- **8-12%** shorter job completion time

![Normalized Job Execution Time Graph](image-url)
Multiple Job Performance

Shuffle-heavy + Shuffle-heavy
- 8% and 23% improvement on tera_sort and inverted-index

Shuffle-heavy + Shuffle-light
- 16% and 25% improvement on tera_sort and histogram-movies
Conclusions

Motivations
- Tight coupling of shuffle of reduce
- Inefficient reduce scheduling
- Parallelism unexploited

iShuffle
- Proactively push shuffle data
- Balancing map output to mitigate data skew
- Flexible reduce scheduling

Results
- Significantly reducing completion time for shuffle-heavy jobs
Questions?
Backup Slides
iShuffle v.s. Random Placement

iShuffle outperforms randomization-based placement algorithms