Chisel++: Handling Partitioning Skew in MapReduce Framework Using Efficient Range Partitioning Technique

Prateek Dhawalia  Sriram Kailasam  D. Janakiraman
Distributed and Object Systems Lab
Dept. of Comp. Sci. and Engg., IIT Madras
OUTLINE

1 INTRODUCTION
   - MapReduce
   - Stragglers
   - Skew

2 MOTIVATION

3 SKEW

4 CHISEL

5 CHISEL++

6 CONCLUSION
MapReduce

- Parallel programming model for data-intensive applications.
- Runs tasks in parallel and provides scalability and fault tolerance.
- Map and Reduce provided as high level functions.

Hadoop

- It is an open source implementation of the MapReduce model.
MapReduce Model

\[ \text{map}(k1, v1) \rightarrow \text{list}(k2, v2) \]
\[ \text{reduce}(k2, \text{list}(v2)) \rightarrow \text{list}(k3, v3) \]
Stragglers are tasks that run disproportionately longer than other tasks in a job.

Reasons for stragglers:
- Internal: Skew
- External: Issues related with the machine, heterogeneity in the cluster and interference from other tasks
Skew occurs when individual tasks in a job have different resource requirements due to unevenness in the amount of input data and/or computation per record.

Skew may be inherent in a job or caused by bad application logic.
INTRODUCTION

MOTIVATION

SKEW

CHISEL

CHISEL++

CONCLUSION
Problem

- Presence of stragglers significantly increases the completion time of a job.

Motivation

- Most of the resources remain idle in presence of stragglers.
- These resources can be efficiently utilized to dynamically re-parallelize straggler tasks and speed up jobs progress.

Job completion depends upon the slowest running task
OUTLINE

1. INTRODUCTION
2. MOTIVATION
3. SKEW
   - Types of Skew
   - Classification of Skew Handling Techniques
4. CHISEL
5. CHISEL++
6. CONCLUSION
As discussed in \(^1\), there can be four types of skew.

- Expensive records in maps
- Variable size input to maps
- Partitioning skew in reduce
- Expensive KeyGroup in reduce

\(^1\) Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, “Skewtune: mitigating skew in mapreduce applications”
Classification of Skew Handling Techniques

**Skew Avoidance:** Steps are taken to avoid skew in tasks before starting them. Involves uniform data distribution to tasks. Gufler et al.\(^2\) and Ibrahim et al.\(^3\) uses skew avoidance for load balancing tasks.

**Skew Detection and Mitigation:** Skew is detected while the task is running. Involves task stealing. Kwon et al.\(^4\) and Guo et al.\(^5\) uses skew detection and mitigation for load balancing tasks.

\(^2\) B. Gufler, N. Augsten, A. Reiser, and A. Kemper, ”Handling data skew in mapreduce”.
\(^3\) S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, ”Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud”.
\(^4\) Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, ”Skewtune: mitigating skew in mapreduce applications”.
\(^5\) Z. Guo, M. Pierce, G. Fox, and M. Zhou, ”Automatic task re-organization in mapreduce”.

Classification of Skew Handling Techniques

Skew Avoidance: Steps are taken to avoid skew in tasks before starting them. Involves uniform data distribution to tasks. Gufler et al.\textsuperscript{2} and Ibrahim et al.\textsuperscript{3} uses skew avoidance for load balancing tasks.

Skew Detection and Mitigation: Skew is detected while the task is running. Involves task stealing. Kwon et al.\textsuperscript{4} and Guo et al.\textsuperscript{5} uses skew detection and mitigation for load balancing tasks.

\textsuperscript{2}B. Gufler, N. Augsten, A. Reiser, and A. Kemper, ”Handling data skew in mapreduce”.
\textsuperscript{3}S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, ”Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud”.
\textsuperscript{4}Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, ”Skewtune: mitigating skew in mapreduce applications”.
\textsuperscript{5}Z. Guo, M. Pierce, G. Fox, and M. Zhou, ”Automatic task re-organization in mapreduce”.
1. INTRODUCTION
2. MOTIVATION
3. SKEW
4. CHISEL
   - Handling Skew in Reduce Tasks
   - Performance
5. CHISEL++
6. CONCLUSION
CHISEL

1. Handles computation skew in map tasks and partitioning/data skew in reduce tasks.

2. Uses Skew Detection and Mitigation for map tasks and Skew Avoidance for reduce tasks.
Handling Data Skew in Reduce Tasks

- Detects data skew rather than computation skew.
- Detecting computation skew would require the expensive shuffle and sort phases to get over.
- Increases parallelism for skewed partitions right before the shuffle begins.
- Additionally, data has to be again shuffled and sorted for the dynamically created reducers.
- Each reducer (Partial Reducer) for skewed partition fetches data from a subset of maps.
- A Global Reducer processes the output of all the partial reducers.
Detects data skew rather than computation skew.

Detecting computation skew would require the expensive shuffle and sort phases to get over.

Increases parallelism for skewed partitions right before the shuffle begins.

Additionally, data has to be again shuffled and sorted for the dynamically created reducers.

Each reducer (Partial Reducer) for skewed partition fetches data from a subset of maps.

A Global Reducer processes the output of all the partial reducers.
Handling Data Skew in Reduce Tasks

- Detects data skew rather than computation skew.
- Detecting computation skew would require the expensive shuffle and sort phases to get over.
- Increases parallelism for skewed partitions right before the shuffle begins.
- Additionally, data has to be again shuffled and sorted for the dynamically created reducers.
- Each reducer (Partial Reducer) for skewed partition fetches data from a subset of maps.
- A Global Reducer processes the output of all the partial reducers.
Handling Data Skew in Reduce Tasks

- Detects data skew rather than computation skew.
- Detecting computation skew would require the expensive shuffle and sort phases to get over.
- Increases parallelism for skewed partitions right before the shuffle begins.
- Additionally, data has to be again shuffled and sorted for the dynamically created reducers.
  - Each reducer (Partial Reducer) for skewed partition fetches data from a subset of maps.
  - A Global Reducer processes the output of all the partial reducers.
Detects data skew rather than computation skew.

Detecting computation skew would require the expensive shuffle and sort phases to get over.

Increases parallelism for skewed partitions right before the shuffle begins.

Additionally, data has to be again shuffled and sorted for the dynamically created reducers.

Each reducer (Partial Reducer) for skewed partition fetches data from a subset of maps.

A Global Reducer processes the output of all the partial reducers.
Handling Data Skew in Reduce Tasks

- Detects data skew rather than computation skew.
- Detecting computation skew would require the expensive shuffle and sort phases to get over.
- Increases parallelism for skewed partitions right before the shuffle begins.
- Additionally, data has to be again shuffled and sorted for the dynamically created reducers.
- Each reducer (Partial Reducer) for skewed partition fetches data from a subset of maps.
- A Global Reducer processes the output of all the partial reducers.
This implementation requires the application reduce logic to be associative in nature.
PERFORMANCE

- 15-node cluster running modified version of Hadoop 0.23.0.
- 14 slave nodes and 1 master node.
- Hadoop was configured to use Capacity Scheduler.
- Each node had 2 AMD Opteron dual core processors running at 2GHz, 4GB of RAM, and 500GB SATA disk drive.
- Experiments were performed using WordCount and InvertedIndex on 16GB data set.
- Map skew was created by adding extra computation per record of input data in certain map tasks.
- Reduce skew was generated by writing a custom partition function.
15-node cluster running modified version of Hadoop 0.23.0.

14 slave nodes and 1 master node.

Hadoop was configured to use Capacity Scheduler.

Each node had 2 AMD Opteron dual core processors running at 2GHz, 4GB of RAM, and 500GB SATA disk drive.

Experiments were performed using WordCount and InvertedIndex on 16GB data set.

Map skew was created by adding extra computation per record of input data in certain map tasks.

Reduce skew was generated by writing a custom partition function.
OUTLINE

1 INTRODUCTION
2 MOTIVATION
3 SKEW
4 CHISEL
5 CHISEL++
   - Motivation
   - Range Partitioning
   - Range Partitioning Performance
   - Chisel++ Performance
6 CONCLUSION
Applications having **high output to input ratio** perform poorly as compared to optimal job duration in Chisel.

The **global reducer** becomes a **bottleneck** when the partial reducers generate huge amount of output.
Use Range-Partitioning among partial reducers.

- **PR**: Partial Reducer
- **GR**: Global Reducer
- **HDFS File**
- **Intermediate File**
- **Partition sizes per map**

Diagram showing range partitioning among partial reducers.
Information about keys occurring at several places in the data is needed for range-partitioning. This can be collected by scanning all the intermediate files after they are written. But scanning entire data will incur a huge overhead. Instead, meta-data is collected when partial reducers generate output, so as to avoid scanning. Data is written into smaller fixed size intermediate files called buckets. Meta-data is created for each bucket containing information like startkey, endkey, size etc.
Design for Range Partitioning

- Information about keys occurring at several places in the data is needed for range-partitioning.
- This can be collected by scanning all the intermediate files after they are written.
- But scanning entire data will incur a huge overhead.
- Instead, meta-data is collected when partial reducers generate output, so as to avoid scanning.
- Data is written into smaller fixed size intermediate files called buckets.
- Meta-data is created for each bucket containing information like startkey, endkey, size etc.
Design For Range Partitioning

- Information about keys occurring at several places in the data is needed for range-partitioning.
- This can be collected by scanning all the intermediate files after they are written.
- But scanning entire data will incur a huge overhead.
- Instead, meta-data is collected when partial reducers generate output, so as to avoid scanning.
- Data is written into smaller fixed size intermediate files called buckets.
- Meta-data is created for each bucket containing information like startkey, endkey, size etc.
Design For Range Partitioning

- Information about keys occurring at several places in the data is needed for range-partitioning.
- This can be collected by scanning all the intermediate files after they are written.
- But scanning entire data will incur a huge overhead.
- Instead, meta-data is collected when partial reducers generate output, so as to avoid scanning.
- Data is written into smaller fixed size intermediate files called buckets.
- Meta-data is created for each bucket containing information like startkey, endkey, size etc.
Information about keys occurring at several places in the data is needed for range-partitioning.

This can be collected by scanning all the intermediate files after they are written.

But scanning entire data will incur a huge overhead.

Instead, meta-data is collected when partial reducers generate output, so as to avoid scanning.

Data is written into smaller fixed size intermediate files called buckets.

Meta-data is created for each bucket containing information like startkey, endkey, size etc.
Design For Range Partitioning

- Information about keys occurring at several places in the data is needed for range-partitioning.
- This can be collected by scanning all the intermediate files after they are written.
- But scanning entire data will incur a huge overhead.
- Instead, meta-data is collected when partial reducers generate output, so as to avoid scanning.
- Data is written into smaller fixed size intermediate files called buckets.
- Meta-data is created for each bucket containing information like startkey, endkey, size etc.
Design For Range Partitioning

- An index of key and its position is also created per bucket after some fixed amount of data is written to a bucket.
- Indexing all the keys will itself generate huge amount of meta data to track.
- Since intermediate data is sorted by key, indexes are also created in a sorted order.
- Binary search is employed to get the offset of a key in the bucket during range partitioning.
- Meta data and index of buckets are sent to Application Master to create ranges for partial reducers.
Pictorial Representation of Range Partitioning
Optimizations During Range Partitioning

- The amount of data shuffled during partitioning should be minimized.
- Variance of data load distribution after range partitioning should be low.
- Balancing one may imbalance the other.
Heuristics for group assignment

- Minimize network cost: [NWO]
  - Assign a group to a reducer which has the maximum data in that group.
  - This maximizes locality of data leading to lower network overhead.
  - No consideration to load balance is given in this method.
Heuristics for group assignment

- Load balance data: [LB]
  - It follows a greedy approach wherein the next group is given to a reducer having the least data.
  - This will achieve a better load balance but will require heavy shuffling of data.
Heuristics for group assignment

- Optimize both network cost and data load balance: [NWO-LB]
  - Calculate the average data per reducer.
  - Initially assign groups to minimize network cost and if a reducer has received the average amount of data, assign groups to balance load.
Heuristics for Group Assignment

- Load balance data maintaining total order of keys: [LB-TO]
  - Calculate the average data per reducer.
  - Cut the entire list of bucket group into smaller lists having continuous ranges.
  - Assign each list to the partial reducer having the maximum data in it.
  - If a reducer has already received a list, assign it to the reducer having the second largest data in it and so on.
8.5GB data (nearly 300 million keys) having uniform initial distribution across 8 reducers.

Load distribution efficiency of various heuristics

Fraction of network transfer that takes place in all the heuristic
Chisel++ Performance

MapReduce jobs along with their o/p to i/p ratio

<table>
<thead>
<tr>
<th>Application</th>
<th>Data Set</th>
<th>o/p to i/p ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordCount(WC-1)</td>
<td>Data-1</td>
<td>0.02</td>
</tr>
<tr>
<td>InvertedIndex(II-1)</td>
<td>Data-1</td>
<td>0.5</td>
</tr>
<tr>
<td>InvertedIndex(II-2)</td>
<td>Data-2</td>
<td>1.00</td>
</tr>
<tr>
<td>MultiWordCount(MWC-1)</td>
<td>Data-1</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Time taken by various jobs using YARN, Chisel and Chisel++ when one partition was 2, 4 and 8 times skewed.
Overall Comparison

Performance gained by various MapReduce Applications having different o/p to i/p ratio and reduce skew

<table>
<thead>
<tr>
<th></th>
<th>speedup</th>
<th>slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reduce Skew-2</td>
<td>Reduce Skew-4</td>
</tr>
<tr>
<td>Chisel</td>
<td>1.18</td>
<td>2.05</td>
</tr>
<tr>
<td>Chisel++</td>
<td>1.24</td>
<td>2.10</td>
</tr>
<tr>
<td>Chisel</td>
<td>1.30</td>
<td>1.69</td>
</tr>
<tr>
<td>Chisel++</td>
<td>1.39</td>
<td>2.26</td>
</tr>
<tr>
<td>Chisel</td>
<td>0.98</td>
<td>0.99</td>
</tr>
<tr>
<td>Chisel++</td>
<td>1.37</td>
<td>1.45</td>
</tr>
<tr>
<td>Chisel</td>
<td>1.01</td>
<td>1.03</td>
</tr>
<tr>
<td>Chisel++</td>
<td>1.30</td>
<td>1.75</td>
</tr>
</tbody>
</table>
Graphical Comparison

Time taken by reduce tasks under default, Chisel and Chisel++ implementation.
Chisel used partial and global reducers to handle data skew in reduce tasks.

Global reducer becomes a bottleneck in i/o intensive workloads.

This bottleneck was removed by eliminating global reducer and using range partitioning to re-shuffle data among partial reducers.
Thank you