LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud

Shadi Ibrahim, Hai Jin, Lu Lu, Song Wu, Bingsheng He*, Qi Li#

Huazhong University of Science and Technology
*Nanyang Technological University
#China Development Bank
Motivation

MapReduce is Becoming very Popular
  × Hadoop is widely used by Enterprise and Academia
    ✓ Yahoo!, Facebook, Baidu, ....
    ✓ Cornell, Maryland, HUST, ..... 

The wide diversity of Today’s Data Intensive applications:
  × Search Engine
  × Social networks
  × Scientific Application
Motivation

Some applications experienced Data Skew in the shuffle phase [1,2]
the current MapReduce implementations have overlooked the skew issue

Results:
- Hash partitioning is inadequate in the presence of data skew
- Design LEEN: Locality and fairness aware key partitioning

Outlines

- Motivation
- Hash partitioning in MapReduce
- LEEN: Locality and Fairness Aware Key Partitioning
- Evaluation
- Conclusion
The current Hadoop’s hash partitioning works well when the keys are equally appeared and uniformly stored in the data nodes.

In the presence of Partitioning Skew:

- Variation in Intermediate Keys’ frequencies
- Variation in Intermediate Key’s distribution amongst different data node

Native blindly hash-partitioning is to be inadequate and will lead to:

- Network congestion
- Unfairness in reducers’ inputs → Reduce computation Skew
- Performance degradation
The Problem (Motivational Example)

Data Node1: K1, K1, K1, K2, K2, K2, K2, K3, K3, K4, K4, K5, K6

Data Node2: K1, K1, K1, K1, K1, K1, K1, K2, K4, K4, K4, K4, K5, K5, K5, K6, K6, K6

Data Node3: K1, K1, K1, K1, K1, K1, K1, K2, K4, K4, K4, K4, K5, K5, K5, K5, K5, K5

hash (Hash code (Intermediate-key) Modulo ReduceID)

<table>
<thead>
<tr>
<th></th>
<th>Data Node1</th>
<th>Data Node2</th>
<th>Data Node3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Data Transfer</td>
<td>11</td>
<td>15</td>
<td>18</td>
<td>44/54</td>
</tr>
<tr>
<td>Reduce Input</td>
<td>29</td>
<td>17</td>
<td>8</td>
<td>CV 58%</td>
</tr>
</tbody>
</table>
Example: Wordcount Example

- **6-node, 2 GB data set!**
- **Combine Function is disabled**
- **Transferred Data is relatively Large**
- **Data Distribution is Imbalanced**

<table>
<thead>
<tr>
<th></th>
<th>Data Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max-Min Ratio</td>
<td>20%</td>
</tr>
<tr>
<td>cv</td>
<td>42%</td>
</tr>
</tbody>
</table>

83% of the Maps output

83% of the Maps output

Data Distribution

Max-Min Ratio

cv

Data Node Distribution

Transferred Data

Local Data

Data During Failed Reduce

Data Node Distribution

Data Node01 Data Node02 Data Node03 Data Node04 Data Node05 Data Node06
Our Work

❯ Asynchronous Map and Reduce execution

❯ Locality-Aware and Fairness-Aware Key Partitioning

LEEN
Asynchronous Map and Reduce execution

Default Hadoop:
- Several maps and reduces are concurrently running on each data
- Overlap computation and data transfer

Our Approach:
- Keep a track on all the intermediate keys’ frequencies and key’s distributions (using DataNode-Keys Frequency Table)
  - Could bring a little overhead due to the unutilized network during the map phase
  - It can fasten the map execution because the complete I/O disk resources will be reserved to the map tasks.
  - For example, the average execution time of map tasks (32 in default Hadoop, 26 Using our approach)
LEEN Partitioning Algorithm

- Extend the Locality-aware concept to the Reduce Tasks
- Consider fair distribution of reducers’ inputs

Results:
- Balanced distribution of reducers’ input
- Minimize the data transfer during shuffle phase
- Improve the response time

Close to optimal tradeoff between Data Locality and reducers’ input Fairness

\[
\begin{align*}
\text{Minimum} & \quad \rightarrow [0,1] \\
\text{Locality} & \quad \rightarrow [0,100] \\
\text{Fairness} & \quad \rightarrow [0,1]
\end{align*}
\]
LEEN Partitioning Algorithm (details)

- Keys are sorted according to their Fairness Values
  - Fairness Locality Value

\[ \text{FLK}_i = \frac{\text{Fairness in distribution of } K_i \text{ amongst data node}}{\text{Node with Best Locality}} \]

- For each key, nodes are sorted in descending order according to the frequency of the specific Key
- Partition a key to a node using Fairness-Score Value
- For a specific Key \( K_i \)
  - If \((\text{Fairness-ScoreN}_i > \text{Fairness-ScoreN}_{i+1})\) move to the next node
  - Else partition \( K_i \) to \( N_j \)
LEEN details (Example)

<table>
<thead>
<tr>
<th></th>
<th>K1</th>
<th>K2</th>
<th>k3</th>
<th>k4</th>
<th>k5</th>
<th>k6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Node2</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Node3</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>16</td>
<td>9</td>
<td>4</td>
<td>13</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>FLK</td>
<td>4.66</td>
<td>2.93</td>
<td>1.88</td>
<td>2.70</td>
<td>2.71</td>
<td>1.66</td>
</tr>
</tbody>
</table>

Data Transfer = 24/54
cv = 14%
Evaluation

Cluster of 7 Nodes
- Intel Xeon two quad-core 2.33GHz
- 8 GB Memory
- 1 TB Disk

Each node runs RHEL5 with kernel 2.6.22
Xen 3.2
Hadoop version 0.18.0

Designed 6 test sets
- Manipulate the Partitioning Skew Degree By modifying the existing textwriter code in Hadoop for generating the input data into the HDFS
## Test sets

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nodes number</strong></td>
<td><strong>6PMs</strong></td>
<td><strong>6PMs</strong></td>
<td><strong>6PMs</strong></td>
<td><strong>6PMs</strong></td>
<td><strong>24VMs</strong></td>
<td><strong>24VMs</strong></td>
</tr>
<tr>
<td><strong>Data Size</strong></td>
<td><strong>14GB</strong></td>
<td><strong>8GB</strong></td>
<td><strong>4.6GB</strong></td>
<td><strong>12.8GB</strong></td>
<td><strong>6GB</strong></td>
<td><strong>10.5GB</strong></td>
</tr>
<tr>
<td><strong>Keys frequencies variation</strong></td>
<td>230%</td>
<td>1%</td>
<td>117%</td>
<td>230%</td>
<td>25%</td>
<td>85%</td>
</tr>
<tr>
<td><strong>Key distribution variation (average)</strong></td>
<td>1%</td>
<td>195%</td>
<td>150%</td>
<td>20%</td>
<td>180%</td>
<td>170%</td>
</tr>
<tr>
<td><strong>Locality Range</strong></td>
<td><strong>24-26%</strong></td>
<td><strong>1-97.5%</strong></td>
<td><strong>1-85%</strong></td>
<td><strong>15-35%</strong></td>
<td><strong>1-50%</strong></td>
<td><strong>1-30%</strong></td>
</tr>
</tbody>
</table>

- **Presence of Keys’ Frequencies Variation**
- **Non-uniform Key’s distribution amongst Data Nodes**
- **Partitioning Skew**
Keys’ Frequencies Variation

- Each key is uniformly distributed among the data nodes.
- Keys frequencies are significantly varying.

\[
\text{Locality Range} = \left[ \frac{\sum_{i=1}^{K} \min_{1\leq j \leq n} (FK_j)}{\sum_{i=1}^{K} FK_i}, \frac{\sum_{i=1}^{K} \max_{1\leq j \leq n} (FK_j)}{\sum_{i=1}^{K} FK_i} \right]
\]

- Execution Time (Sec)
- Native Hadoop vs LEEN
- Cuncurrent Map and Shuffle
- Reduce
- Shuffle (Separate)
- Map (Separate)

- Data Distribution (CV)
- Native Hadoop
- LEEN

10 x

6%
Non-Uniform Key Distribution

- Each key is non-uniformly distributed among the data nodes
- Keys frequencies are nearly equal
Partitioning Skew

Locality Range

<table>
<thead>
<tr>
<th>Locality</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Range</td>
<td>1-85%</td>
<td>15-35%</td>
<td>1-50%</td>
<td>1-30%</td>
</tr>
</tbody>
</table>

Data Distribution (CV)

Execution Time (Sec)

- Native Hadoop
- LEEN

- Concurrent Map and Shuffle
- Reduce
- Shuffle (Separate)
- Map (Separate)
Conclusion

Partitioning Skew is a challenge for MapReduce-based applications:

- Today, diversity of Data-intensive applications
  - Social Network, Search engine, Scientific Analysis, etc
- Partitioning Skew is due to two factors:
  - Significant variance in intermediate keys’ frequencies
  - Significant variance in intermediate key’s distributions among the different data.
- Our solution is to extend the Locality concept to the reduce phase
  - Partition the Keys according to
    - their high frequencies
    - Fairness in data distribution among different data nodes

Up to 40% improvement using simple application example!

Future work

- Apply LEEN to different key and values size
Thank you!

Questions?

shadi@hust.edu.cn
http://grid.hust.edu.cn/shadi