ICPP 2015

Optimizing MapReduce based on Locality of K-V Pairs and Overlap between Shuffle and Local Reduce
• Background
• LELB and MLSR
• Experiments and Conclusion
Background

- high throughput
- low-cost
- scalability
- large data set
Background

Performance degradation
- Communication cost in the shuffling phase
- Load imbalance in the reduction phase
Figure 1. The distribution of keys (A simplified example).

\[
\text{sum of workload}(A) = \text{sum of workload}(B) = \text{sum of workload}(C)
\]

**Key issues**
- Which node will keys A, B and C be executed on?
- Which job will be executed first on every node?
- For each node, will it send/receive data using a network source, or will it reduce using the computing source?
Based on (1) & (2), take into account both the internal node locality and locality between all the nodes.

(1) the proportion of the \( k \)th key on the \( n \)th Map Node

\[
(1) \text{Locality}^1_{n} = \frac{\text{Key}^k_n}{\sum_{k=1}^{\text{numKeys}} \text{Key}^k_n}
\]

(2) the proportion of the \( k \)th key on the \( n \)th Map Node of the \( k \)th key on all the Map Nodes

\[
(2) \text{Locality}^2_{n} = \frac{\text{Key}^k_n}{\sum_{n=1}^{\text{numMapNodes}} \text{Key}^k_n}
\]

(3) Based on (1) & (2), take into account both the internal node locality and locality between all the nodes

\[
(3) \text{Locality}^k_{n} = \text{Locality}^1_{n} \times \text{Locality}^2_{n} = \frac{\left(\sum_{k=1}^{\text{numKeys}} \text{Key}^k_n\right) \times \left(\sum_{n=1}^{\text{numMapNodes}} \text{Key}^k_n\right)}{\text{numKeys} \times \text{numMapNodes}}
\]

Fig. 2. The distribution of keys on 3 Map nodes
Algorithm 1  LELB Algorithm

Input: \(key^k\): the \(k\)th key
\(key^k_n\): the number of \(k\)th key on \(n\)th nodes
\(key^k_n\): the number of the \(k\)th key on the \(n\)th Map Node. Where,
\(1 \leq k \leq numKeys\), \(1 \leq n \leq numMapNodes\)
numKeys: the number of keys
numMapNodes: the number of Map Nodes
\(M = \{key^k, 1 \leq k \leq numKeys\}\)
LTV: the threshold value

Output: load balance scheduling scheme during reduce phase

1: initialize numMapNodes sets of potential reducers to schedule,
\(R_n = \emptyset, 1 \leq n \leq numMapNode s\)
2: for all \(1 \leq k \leq numKeys\) do
3:   for all \(1 \leq n \leq numMapNode s\) do
4:     \(Locality^1_n \leftarrow key^k_n / \sum_{n'=1}^{numMapNode s} key^k_n\)
5:     \(Locality^2_n \leftarrow key^k_n / \sum_{n'=1}^{numMapNode s} key^k_n\)
6:     \(Locality^3_n \leftarrow Locality^1_n \times Locality^2_n\)
7: end for
8: AverageLoad \leftarrow \frac{\sum_{n=1}^{numMapNode s} \sum_{k=1}^{numKeys} key^k_n}{numMapNode s}\)
9: \(Load_n \leftarrow 0, 1 \leq n \leq numMapNode s\)
10: calculate maximum-value
11: \(maxLocality = \max\{Locality^3_n: key^k \in M\}\),
12: \(mk \leftarrow k\) and \(mn \leftarrow n\)
13: \(Load_{mn} \leftarrow Load_{mn} + \sum_{n=1}^{numMapNode s} key^k_{mn}\)
14: if \(|Load_{mn} - \text{AverageLoad}| \leq LTV\) then
15:   add \(key^m\) to \(R_{mn}\), \(key^m\) the \(m\)th key will be executed reduce task on the \(mn\)th Map Node
16:   delete \(key^m\) from \(M\)
17: end if
18: delete \(Locality^3_n\) from
19: \(\{Locality^3_n, 1 \leq k \leq numKeys, 1 \leq n \leq numMapNode s\}\)
20: if \(M\) is not empty then
21:   goto L1
22: else
23:   return \(R_n, 1 \leq n \leq numMapNode s\)
different situations:

(1) \( \text{Locality}_n^k = \frac{(k^{\text{key}_n})^2}{\sum_{k=1}^{\text{numKeys}} k^{\text{key}_n}} \)

(2) \( \text{Locality}_n^k = \frac{(k^{\text{key}_n})^2}{\sum_{n=1}^{\text{numMapNode}} k^{\text{key}_n}} \)

Table I

<table>
<thead>
<tr>
<th>node1</th>
<th>key1</th>
<th>key2</th>
<th>key3</th>
<th>key4</th>
<th>key5</th>
<th>key6</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>100</td>
<td>50</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>80</td>
<td>100</td>
<td>70</td>
<td>130</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>20</td>
<td>50</td>
<td>90</td>
<td>10</td>
<td>70</td>
<td></td>
</tr>
</tbody>
</table>

Table II

<table>
<thead>
<tr>
<th>node1</th>
<th>key1</th>
<th>key2</th>
<th>key3</th>
<th>key4</th>
<th>key5</th>
<th>key6</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.58</td>
<td>26.32</td>
<td>6.58</td>
<td>4.21</td>
<td>9.47</td>
<td>16.84</td>
<td></td>
</tr>
<tr>
<td>1.96</td>
<td>13.91</td>
<td>21.74</td>
<td>10.65</td>
<td>36.74</td>
<td>5.43</td>
<td></td>
</tr>
<tr>
<td>40.00</td>
<td>1.11</td>
<td>6.94</td>
<td>22.50</td>
<td>0.28</td>
<td>13.61</td>
<td></td>
</tr>
</tbody>
</table>

Table III

<table>
<thead>
<tr>
<th>node1</th>
<th>key1</th>
<th>key2</th>
<th>key3</th>
<th>key4</th>
<th>key5</th>
<th>key6</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>100</td>
<td>70</td>
<td>40</td>
<td>50</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>130</td>
<td>100</td>
<td>60</td>
<td>80</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>50</td>
<td>80</td>
<td>80</td>
<td>60</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

Table IV

<table>
<thead>
<tr>
<th>node1</th>
<th>key1</th>
<th>key2</th>
<th>key3</th>
<th>key4</th>
<th>key5</th>
<th>key6</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.63</td>
<td>35.71</td>
<td>19.60</td>
<td>8.89</td>
<td>13.16</td>
<td>57.86</td>
<td></td>
</tr>
<tr>
<td>2.50</td>
<td>60.38</td>
<td>40.00</td>
<td>20.00</td>
<td>35.68</td>
<td>0.71</td>
<td></td>
</tr>
<tr>
<td>30.65</td>
<td>8.93</td>
<td>25.60</td>
<td>35.56</td>
<td>18.95</td>
<td>11.43</td>
<td></td>
</tr>
</tbody>
</table>
Supposed that the time complexity of computation for \( n \) keys is \( f_C(n) \) the time complexity of communication for \( n \) keys is \( f_T(n) \)

The Cost of executing **Local Reduce + Shuffle + Final Reduce** is:

\[
\begin{align*}
\text{Cost}_1 &= \max \{ f_C(n_1), f_C(n_2), \ldots, f_C(n_m) \} \\
&+ \max \{ f_{T_1}(n_1), f_{T_1}(n_2), \ldots, f_{T_1}(n_m) \} \\
&+ \alpha f_C(n_1 + n_2 + \ldots + n_m),
\end{align*}
\]

where \( f_{T_i}(n_i) = 0 \) and \( \alpha \leq 1 \)

The cost of executing traditional Shuffle + Reduce is:

\[
\begin{align*}
\text{Cost}_2 &= \max \{ f_{T_1}(n_1), f_{T_1}(n_2), \ldots, f_{T_1}(n_m) \} \\
&+ f_C(n_1 + n_2 + \ldots + n_m),
\end{align*}
\]

\[
\therefore \text{Cost}_2 - \text{Cost}_1 = (1 - \alpha) f_C(N) - \max_{i=1}^{m} \{ f_C(\beta_i N) \}
\]

if \( \alpha \leq 1 - \frac{\max_{i=1}^{m} \{ f_C(\beta_i N) \}}{f_C(N)} \),

\( \text{Cost}_1 \) is smaller than \( \text{Cost}_2 \) that is, the scheme of “**Local Reduce + Shuffle + Final Reduce**” will be applied.
Table V

<table>
<thead>
<tr>
<th>Time Complexity</th>
<th>The upper bound of $\alpha$</th>
<th>Scope of $\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_a(n) = O(1)$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$f_a(n) = O(n)$</td>
<td>$1 - \max_{i=1}^{m} O(\beta_i)$</td>
<td>$[0, 1 - 1/m]$</td>
</tr>
<tr>
<td>$f_a(n) = O(n^2)$</td>
<td>$1 - \max_{i=1}^{m} O(\beta_i^2)$</td>
<td>$[0, 1 - 1/m^2]$</td>
</tr>
<tr>
<td>$f_a(n) = O(n^3)$</td>
<td>$1 - \max_{i=1}^{m} O(\beta_i^3)$</td>
<td>$[0, 1 - 1/m^3]$</td>
</tr>
<tr>
<td>$f_a(n) = O(n^k), k \geq 1$</td>
<td>$1 - \max_{i=1}^{m} O(\beta_i^k)$</td>
<td>$[0, 1 - 1/m^k]$</td>
</tr>
<tr>
<td>$f_a(n) = O(\log n)$</td>
<td>$-\max_{i=1}^{m} O(\log(\log(n))) / O(\log(N))$</td>
<td>$[0, \log_N m]$</td>
</tr>
<tr>
<td>$f_a(n) = O(n \log n)$</td>
<td>$1 - \max_{i=1}^{m} O(\beta_i) - \max_{i=1}^{m} O(\beta_i \log(\log n)) / O(\log(N))$</td>
<td>$[0, 1 - 1/m + \log_2 m/m]$</td>
</tr>
</tbody>
</table>

Figure 4. The execution flow of shuffle and reduce in traditional MapReduce (Case 1).

Figure 5. Case 2.

Figure 6. Case 3 & Case 4 (1).

Figure 7. Case 3 & Case 4 (2).

Figure 8. Case 3 & Case 4 (3).
Algorithm 2  MLSR (Map + Local Reduce + Shuffle + Reduce) Algorithm

**Input:**
- \( key^k \): the \( k \)th key
- \( numKeys \): the number of keys
- \( numMapNodes \): the number of Map Nodes
- \( R_n, 1 \leq n \leq numMapNodes \): load balance scheduling scheme during reduce phase generated by LELBA

**Output:** scheduling scheme generated by MLSRA

1:    for all \( 1 \leq n \leq numMapNodes \) do
2:        for all \( 1 \leq k \leq numKeys \) do
3:            if \( key^k \not\in R_n \) then
4:                if Cost(Local Reduce+Shuffle+Reduce) of \( key^k \) is less than Cost(Shuffle+Reduce) of \( key^k \) then
5:                    local reduce for \( key^k \) on the \( n \)th nodes
6:                end if
7:            end if
8:        end for
9:    end for
10:   for all \( 1 \leq n \leq numMapNodes \) do
11:       for all \( 1 \leq k \leq numKeys \) do
12:           if \( key^k \in R_n \) then
13:               local reduce for \( key^k \) on the \( n \)th nodes
14:           end if
15:       end for
16:    end for
17:    final reduce for \( key^k \cdot 1 \leq k \leq numKeys \)
Experiments

examples
• Word count
• Merge sort

different factors
• data sizes
• map tasks’ number

Table VI
THE HARDWARE TEST ENVIRONMENT

<table>
<thead>
<tr>
<th>NameNode</th>
<th>DataNode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Intel multi-core server</td>
<td>3 SMP Intel Servers</td>
</tr>
<tr>
<td>4-way 4-core Intel Xeon 2.13 GHz</td>
<td>2-core Intel Xeon 3.0GHz</td>
</tr>
<tr>
<td>2 x 2M L2 Cache</td>
<td>1M L2 Cache</td>
</tr>
<tr>
<td>2GB Memory</td>
<td></td>
</tr>
<tr>
<td>36GB Hard Disk</td>
<td></td>
</tr>
<tr>
<td>2 x Intel EtherExpress/1000 network cards</td>
<td></td>
</tr>
</tbody>
</table>

Table VII
THE SOFTWARE TEST ENVIRONMENT

<table>
<thead>
<tr>
<th>NameNode</th>
<th>DataNode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redhat Enterprise Linux Server Release 5.2</td>
<td>Fedora 3</td>
</tr>
<tr>
<td>hadoop: 0.20.2</td>
<td></td>
</tr>
<tr>
<td>Eclipse: Europa 3.3</td>
<td></td>
</tr>
</tbody>
</table>
Figure 9. The relationship between the computing performance and the size of data for Merge Sort.

Figure 10. The relationship between the computing performance and the number of map tasks for Merge Sort.
Experiments

Figure 9. The relationship between the computing performance and the size of data for word count.

Figure 10. The relationship between the computing performance and the number of map tasks for word count.
• This paper proposes a Locality-Enhanced Load Balance (LELB) algorithm

• And extends the execution flow of MapReduce to Map, Local reduce, Shuffle and final Reduce (MLSR), then proposes a corresponding MLSR algorithm.

• Use of the novel algorithms can share the computation of reduce and overlap with shuffle in order to take full advantage of CPU and I/O resources.

• The actual test results demonstrate that the execution performance outperforms the execution performance using hadoop by up to 9.2% (for Merge Sort) and 14.4% (for WordCount).


THANKS FOR LISTENING!

• **Contact:**
  – Jianjiang Li
    • lijianjiang@ustb.edu.cn
  – Jie Wu
    • jiewu@temple.edu
  – Xiaolei Yang
    • chinayangxiaolei@163.com
  – Shiqi Zhong
    • zhongshiqi1991@163.com