ICPP 2015

Optimizing MapReduce based on Locality of K-V Pairs and Overlap between Shuffle and Local Reduce

Content

- **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

LELB

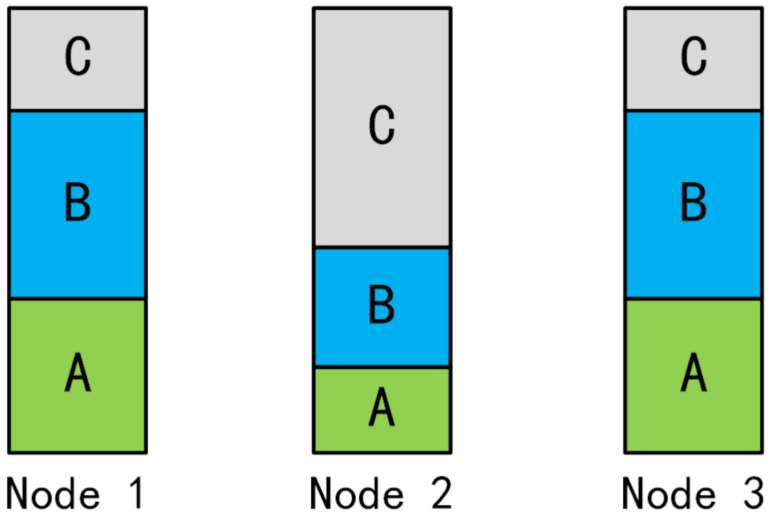

Figure 1. The distribution of keys
(A simplified example).

sum of workload(A)
= sum of workload(B)
=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?

LELB

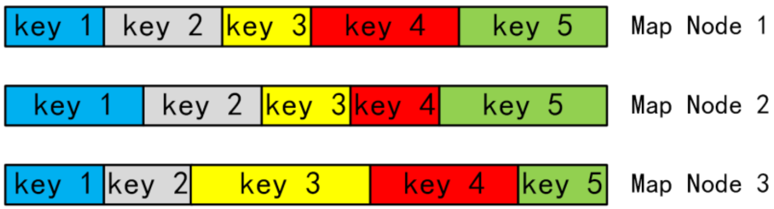

Fig.2. The distribution of keys on 3 Map nodes

$$(1) Locality1_n^k = key_n^k \Big/ \sum_{k=1}^{numKeys} key_n^k$$

$$(2) Locality2_n^k = key_n^k \Big/ \sum_{n=1}^{numMapNodes} key_n^k$$

$$(3) Locality_n^k = Locality1_n^k \times Locality2_n^k$$

$$= \frac{key_n^k}{\sum_{k=1}^{numKeys} key_n^k} \times \frac{key_n^k}{\sum_{n=1}^{numMapNodes} key_n^k}$$

$$= \frac{(key_n^k)^2}{\sum_{k=1}^{numKeys} key_n^k \times \sum_{n=1}^{numMapNodes} key_n^k}$$

(1) the proportion of the *k*th key on the nth Map Node

(2) the proportion of the *k*th key on the nth Map Node of the *k*th key on all the Map Nodes

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

LELB

Algorithm 1 LELB Algorithm

Input: key^k: the kth key

key_n^k: the number of kth key on nth nodes

key_n^k: the number of the kth key on the nth Map Node. Where,

$1 \le k \le numKeys$, $1 \le n \le numMapNodes$

$numKeys$: the number of keys

$numMapNodes$: the number of Map Nodes

$M = \{key^k, 1 \le k \le numKeys\}$

LTV: the threshold value

Output: load balance scheduling scheme during reduce phase

initialize $numMapNodes$ sets of potential reducers to schedule, $R_n = \Phi, 1 \le n \le numMapNodes$

for all $1 \le k \le numKeys$ **do**

  for all $1 \le n \le numMapNodes$ **do**

    $Locality1_n^k \leftarrow key_n^k \Big/ \sum_{k=1}^{numKeys} key_n^k$

    $Locality2_n^k \leftarrow key_n^k \Big/ \sum_{n=1}^{numMapNodes} key_n^k$

    $Locality_n^k \leftarrow Locality1_n^k \times Locality2_n^k$

  end for

end for

AverageLoad $\leftarrow \dfrac{\sum_{n=1}^{numMapNodes}\sum_{k=1}^{numKeys} key_n^k}{numMapNodes}$

$Load_n \leftarrow 0$, $1 \le n \le numMapNodes$

L1:

calculate maximum-value maxLocality=max$\{Locality_n^k, key^k \in M\}$, $mk \leftarrow k$ and $mn \leftarrow n$

$Load_{mn} \leftarrow Load_{mn} + \sum_{n=1}^{numMapNode} key_n^{mk}$

if $|Load_{mn} - averageLoad| \le LTV$ **then**

  add key^{mk} to R_{mn}, key^{mk} the mkth key will be executed reduce task on the mnth Map Node

  delete key^{mk} from M

else

  $Load_{mn} \leftarrow Load_{mn} - \sum_{n=1}^{numMapNode} key_n^{mk}$

end if

delete $Locality_{mn}^{mk}$ from $\{Locality_n^k, 1 \le k \le numKeys, 1 \le n \le numMapNodes\}$

if M is not empty **then**

  goto L1

else

  return $R_n, 1 \le n \le numMapNodes$

end if

LELB

different situations:

$$(1)\ Locality_n^k = (key_n^k)^2 / \sum_{k=1}^{numKeys} key_n^k$$

$$(2)\ Locality_n^k = (key_n^k)^2 / \sum_{n=1}^{numMapNode} key_n^k$$

Table I
THE NUMBER OF DIFFERENT KEYS ON EVERY NODE IN SITUATION(1)

	key1	key2	key3	key4	key5	key6
node1	50	100	50	40	60	80
node2	30	80	100	70	130	50
node3	120	20	50	90	10	70

Table II
THE LOCALITY OF DIFFERENT KEYS ON EVERY NODE IN SITUATION(1)

	key1	key2	key3	key4	key5	key6
node1	6.58	*26.32*③	6.58	4.21	9.47	*16.84*④
node2	1.96	13.91	*21.74*⑤	10.65	*36.74*②	5.43
node3	*40.00*①	1.11	6.94	*22.50*⑥	0.28	13.61

Table III
THE NUMBER OF DIFFERENT KEYS ON EVERY NODE IN SITUATION(2)

	key1	key2	key3	key4	key5	key6
node1	50	100	70	40	50	90
node2	20	130	100	60	80	10
node3	90	50	80	80	60	40

Table IV
THE LOCALITY OF DIFFERENT KEYS ON EVERY NODE IN SITUATION(2)

	key1	key2	key3	key4	key5	key6
node1	15.63	35.71	*19.60*⑤	8.89	13.16	*57.86*②
node2	2.50	*60.36*①	40.00	20.00	*33.68*⑥	0.71
node3	*50.63*③	8.93	25.60	*35.56*④	18.95	11.43

MLSR

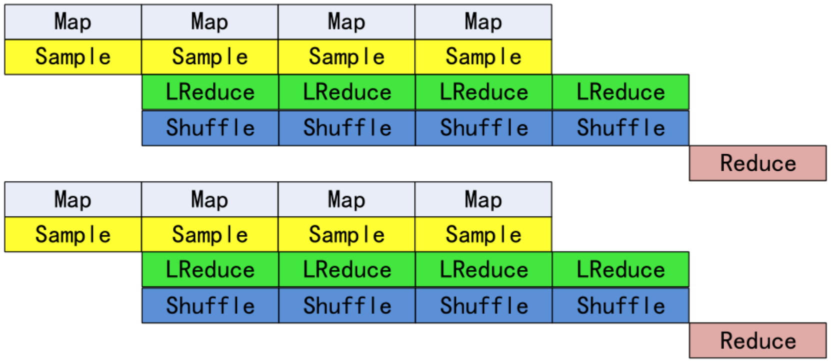

Figure 3. The execution flow of MLSR.

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:

$$Cost1 = \max\{f_C(n_1), f_C(n_2), \ldots\ldots, f_c(n_m)\}$$
$$+ \max\{f_{Ti}(n_1), f_{Ti}(n_2), \ldots\ldots, f_{Ti}(n_m)\}$$
$$+ \alpha f_C(n_1 + n_2 + \ldots\ldots + n_m),$$

where $f_{Ti}(n_i) = 0$ and $\alpha \le 1$

The cost of executing traditional Shuffle + Reduce is:

$$Cost2 = \max\{f_{Ti}(n_1), f_{Ti}(n_2), \ldots\ldots, f_{Ti}(n_m)\}$$
$$+ f_C(n_1 + n_2 + \ldots\ldots + n_m),$$

$$\therefore Cost2 - Cost1 = (1-\alpha) f_C(N) - \max_{i=1}^{m}\{f_C(\beta_i N)\}$$

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

Cost1 is smaller than *Cost2*, that is, the scheme of "**Local Reduce + Shuffle + Final Reduce**" will be applied.

MLSR

Table V
THE RELATIONSHIP BETWEEN TIME COMPLEXITY OF REDUCE AND THE SCOPE OF α

Time Complexity	The upper bound of α	Scope of α
$f_c(n) = O(1)$	0	0
$f_c(n) = O(n)$	$1 - max_{i=1}^{m}\{O(\beta_i)\}$	$[0, 1 - 1/m]$
$f_c(n) = O(n^2)$	$1 - max_{i=1}^{m}\{O(\beta_i^2)\}$	$[0, 1 - 1/m^2]$
$f_c(n) = O(n^3)$	$1 - max_{i=1}^{m}\{O(\beta_i^3)\}$	$[0, 1 - 1/m^3]$
$f_c(n) = O(n^k), k \geq 1$	$1 - max_{i=1}^{m}\{O(\beta_i^k)\}$	$[0, 1 - 1/m^k]$
$f_c(n) = O(logn)$	$-max_{i=1}^{m}\{O(log(\beta_i))\}/O(logN)$	$[0, log_N m]$
$f_c(n) = O(nlogn)$	$1 - max_{i=1}^{m}\{O(\beta_i)\} - max_{i=1}^{m}\{O(\beta_i log\beta_i)\}/O(logN)$	$[0, 1 - 1/m + log_N m/m]$

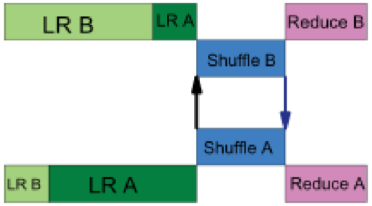

Figure 5. Case 2.

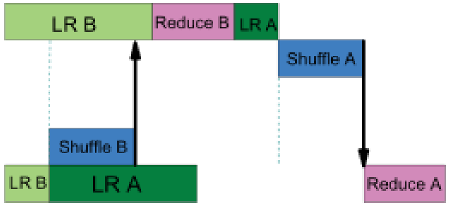

Figure 6. Case 3 & Case 4 (1).

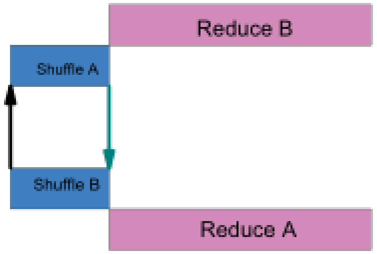

Figure 4. the execution flow of shuffle and reduce in traditional MapRedcue (Case 1).

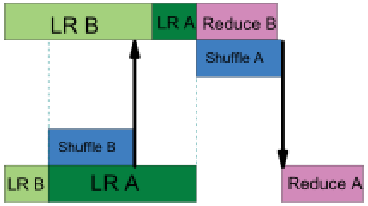

Figure 7. Case 3 & Case 4 (2).

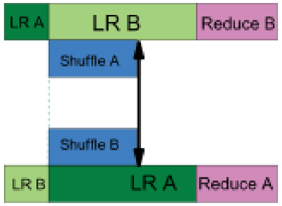

Figure 8. Case 3 & Case 4 (3).

MLSR

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

$R_n, 1 \le n \le numMapNodes$: load balance scheduling scheme during reduce phase generated by LELBA

Output: scheduling scheme generated by MLSRA

for all $1 \le n \le numMapNodes$ **do**
    for all $1 \le k \le numKeys$ **do**
        if $key^k \notin R_n$ **then**
            if *Cost(Local Reduce+Shuffle+Reduce)* of key^k is less than *Cost(Shuffle+Reduce)* of key^k **then**
                local reduce for key^k on the *n*th nodes
            end if
            shuffle for key^k
        end if
    end for
end for
for all $1 \le n \le numMapNodes$ **do**
    for all $1 \le k \le numKeys$ **do**
        if $key^k \in R_n$ **then**
            local reduce for key^k on the *n*th nodes
        end if
    end for
end for
final reduce for key^k, $1 \le k \le numKeys$

Experiments

examples
- Word count
- Merge sort

different factors
- data sizes
- map tasks' number

Table VI
THE HARDWARE TEST ENVIRONMENT

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

Table VII
THE SOFTWARE TEST ENVIRONMENT

NameNode	DataNode
Redhat Enterprise Linux Server Release 5.2	Fedora 3
hadoop: 0.20.2	
Eclipse: Europa 3.3	

Experiments

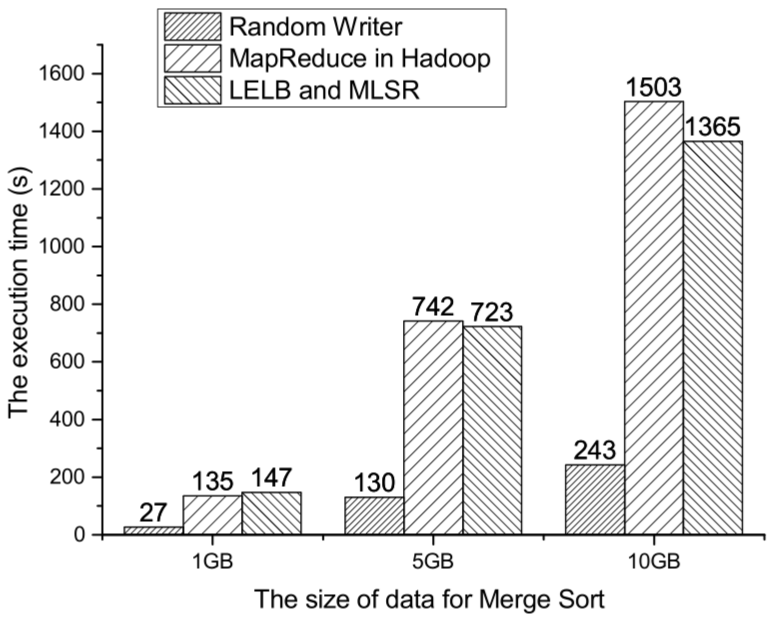

Figure 9. The relationship between the computing performance and the size of data for Merge Sort.

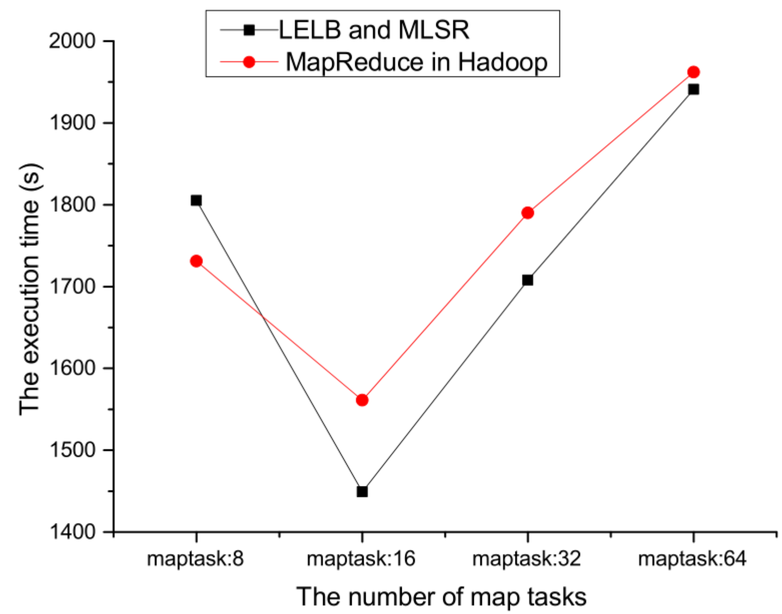

Figure 10. The relationship between the computing performance and the number of map tasks for Merge Sort.

Experiments

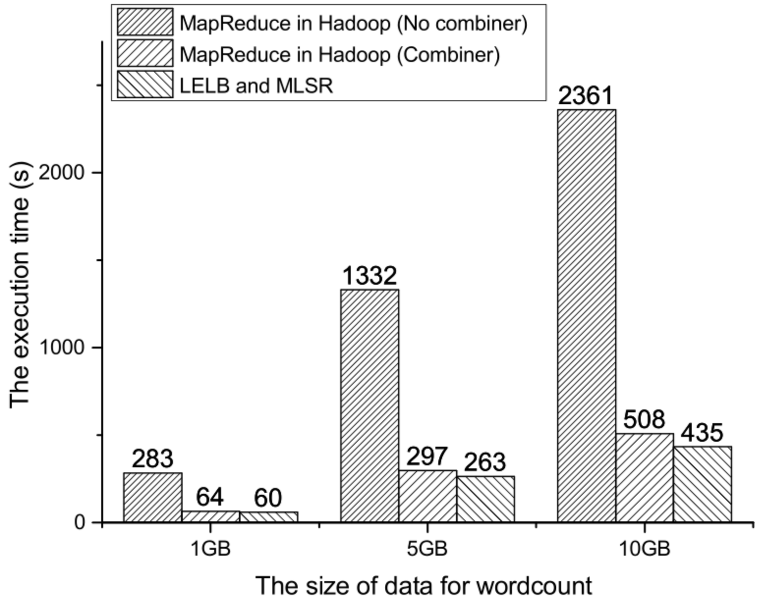

Figure 9. The relationship between the computing performance and the size of data for word count.

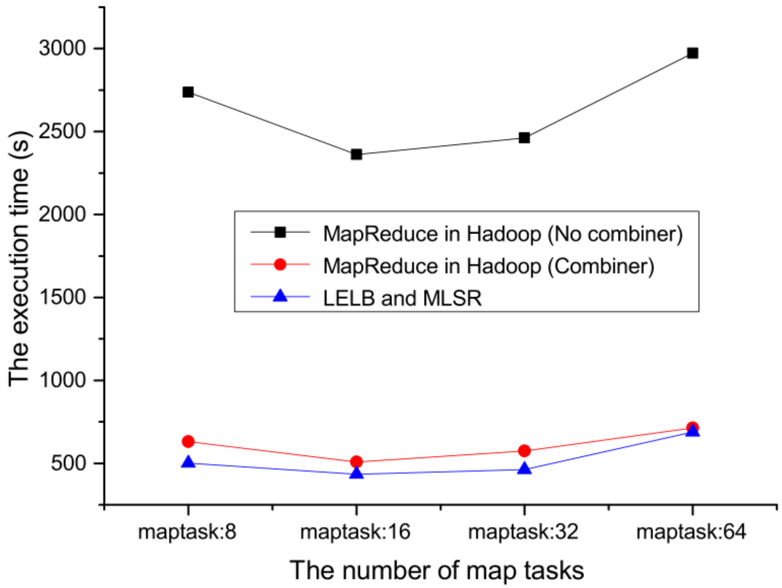

Figure 10. The relationship between the computing performance and the number of map tasks for word count.

University of Science and Technology Beijing

TEMPLE UNIVERSITY

Conclusion

- 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).

Reference

- S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen:Locality/fairness-aware key partitioning for mapreduce in the cloud," in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on.IEEE, 2010, pp. 17–24.
- S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, "Handling partitioning skew in mapreduce using leen," Peer-to-Peer Networking and Applications, vol. 6, no. 4, pp. 409–424, 2013.
- M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. S-toica, "Improving mapreduce performance in heterogeneous environments," in OSDI, vol. 8, 2008, p. 7.
- Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality in mapreduce," in Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Computer Society, 2012, pp. 419–426.

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**