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 banck

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 presenese of data skew
  - Design LEEN: Locality and fairness aware key partitioning

1. X. Qiu, J. Ekanayake, S. Beason, T. Gunarathne, G. Fox, R. Barga, and D. Gannon, “Cloud technologies for bioinformatics applications”, *Proc. ACM Work. Many-Task Computing on Grids and Supercomputers (MTAGS 2009)*, ACM Press, Nov. 2009.
2. J. Lin, “The Curse of Zipf and Limits to Parallelization: A Look at the Stragglers Problem in MapReduce”, *Proc. Work. Large-Scale Distributed Systems for Information Retrieval (LSDS-IR'09)*, Jul. 2009.

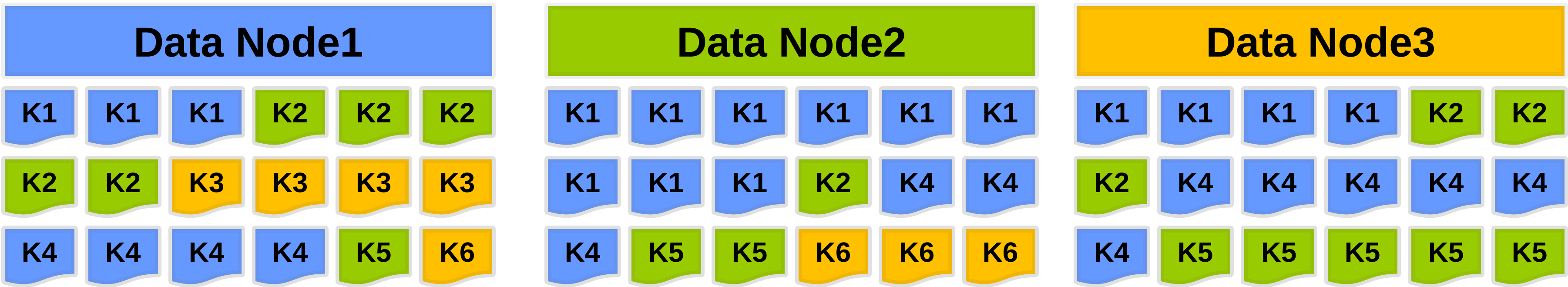
Outlines

- Motivation
- Hash partitioning in MapReduce
- LEEN: Locality and Fairness Aware Key Partitioning
- Evaluation
- Conclusion

Hash partitioning in Hadoop

- 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)



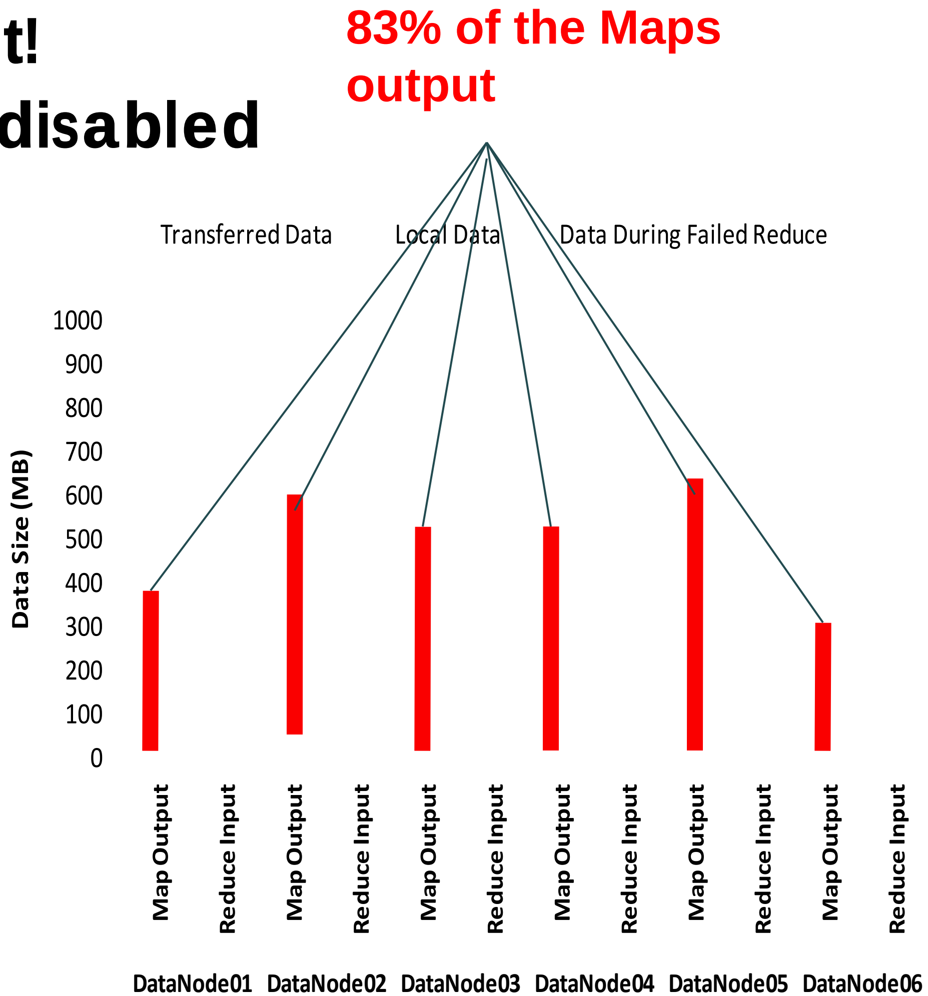
hash (Hash code (Intermediate-key) Modulo ReduceID)

	Data Node1	Data Node2	Data Node3	
Total Data Transfer	**11**	**15**	**18**	Total 44/54
Reduce Input	**29**	**17**	**8**	cv 58%

Example: Wordcount Example

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

	Data Distribution
Max-Min Ratio	20%
cv	42%



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

Minimum $\dfrac{\text{Fairness}}{\text{Locality}}$

Fairness → **[0,1]**

Locality → **[0,100]**

LEEN Partitioning Algorithm (details)

- Keys are sorted according to their $\frac{\text{Fairness}}{\text{Locality}}$ Values
  - Fairness Locality Value

$$FLK_i = \frac{\text{Fairness in distribution of } K_i \text{ amongst data node}}{\text{Node with Best Locality}}$$

- 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 (Fairness-ScoreN^j > Fairness-ScoreN^{j+1}) move to the next node
  - Else partition K_i to N_j

LEEN details (Example)

	K1	K2	k3	k4	k5	k6	
Node1	3	5	4	4	1	1	**18**
Node2	9	1	0	3	2	3	**18**
Node3	4	3	0	6	5	0	**18**
Total	**16**	**9**	**4**	**13**	**8**	**4**	
FLK	**4.66**	**2.93**	**1.88**	**2.70**	**2.71**	**1.66**	

Data Transfer = 24/54
cv = 14%

Data Node1

K1 K1 K1 K2 K2 K2
K2 K2 K3 K3 K3 K3
K4 K4 K4 K4 K5 K6

Data Node2

K1 K1 K1 K1 K1 K1
K1 K1 K1 K2 K4 K4
K4 K5 K5 K6 K6 K6

Data Node3

K1 K1 K1 K1 K2 K2
K2 K4 K4 K4 K4 K4
K4 K5 K5 K5 K5 K5

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

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

Presence of Keys' Frequencies Variation (1)

Non-uniform Key's distribution amongst Data Nodes (2)

Partitioning Skew (3–6)

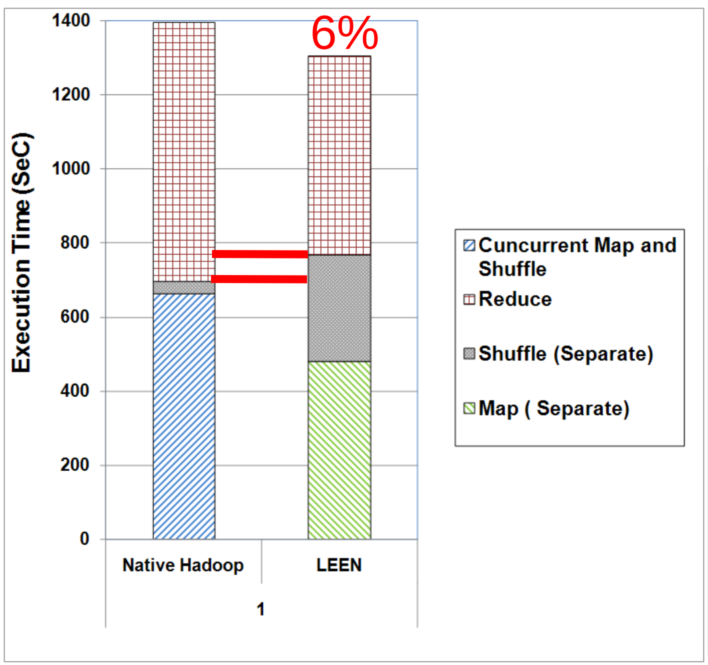
Keys' Frequencies Variation

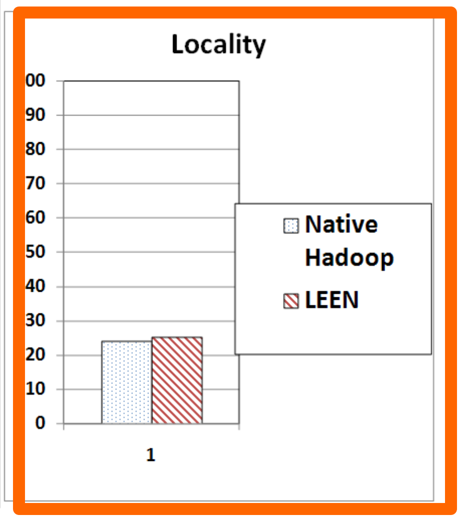
- Each key is uniformly distributed among the data nodes
- Keys frequencies are significantly varying

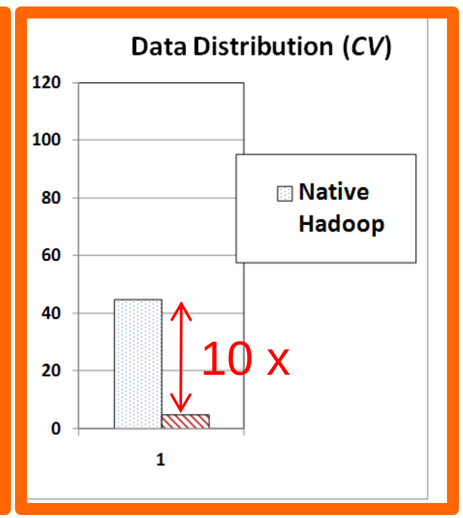
Locality Range

$$\left[\frac{\sum_{i=1}^{K} min_{1 \le j \le n}(FK_i^j)}{\sum_{i=1}^{K} FK_i} , \frac{\sum_{i=1}^{K} max_{1 \le j \le n}(FK_i^j)}{\sum_{i=1}^{K} FK_i} \right]$$

24-26%

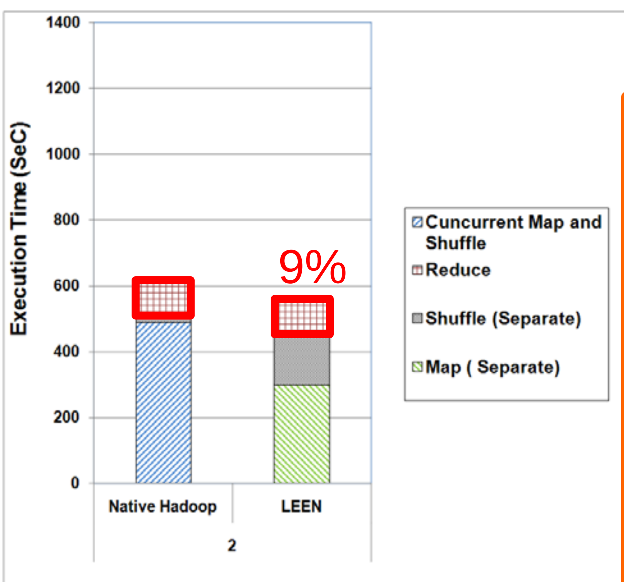


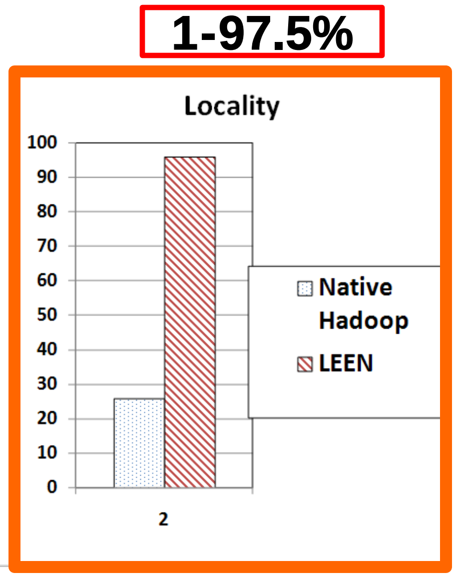


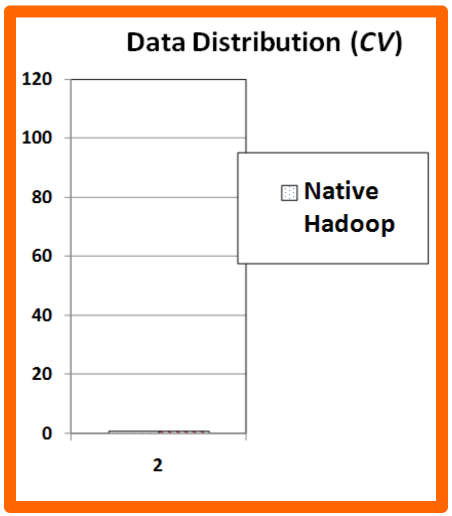


Non-Uniform Key Distribution

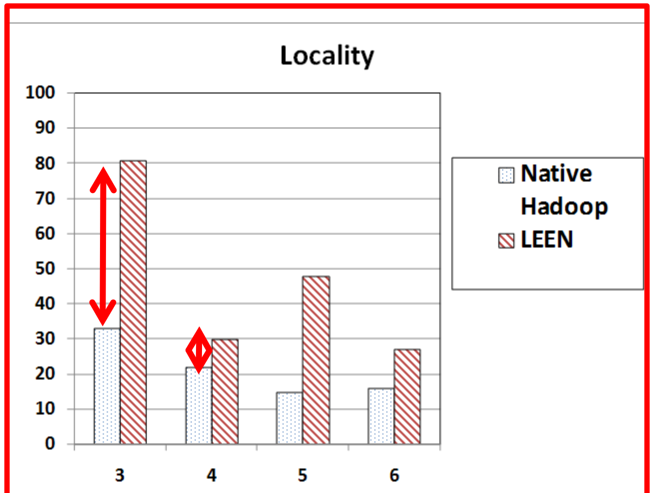
- Each key is non-uniformly distributed among the data nodes
- Keys frequencies are nearly equal

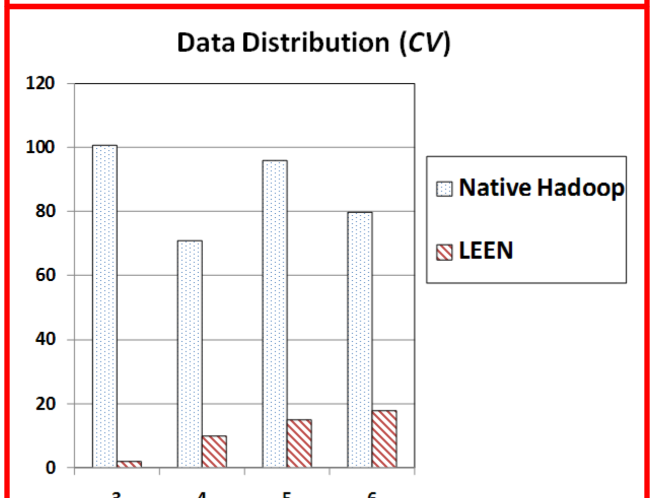




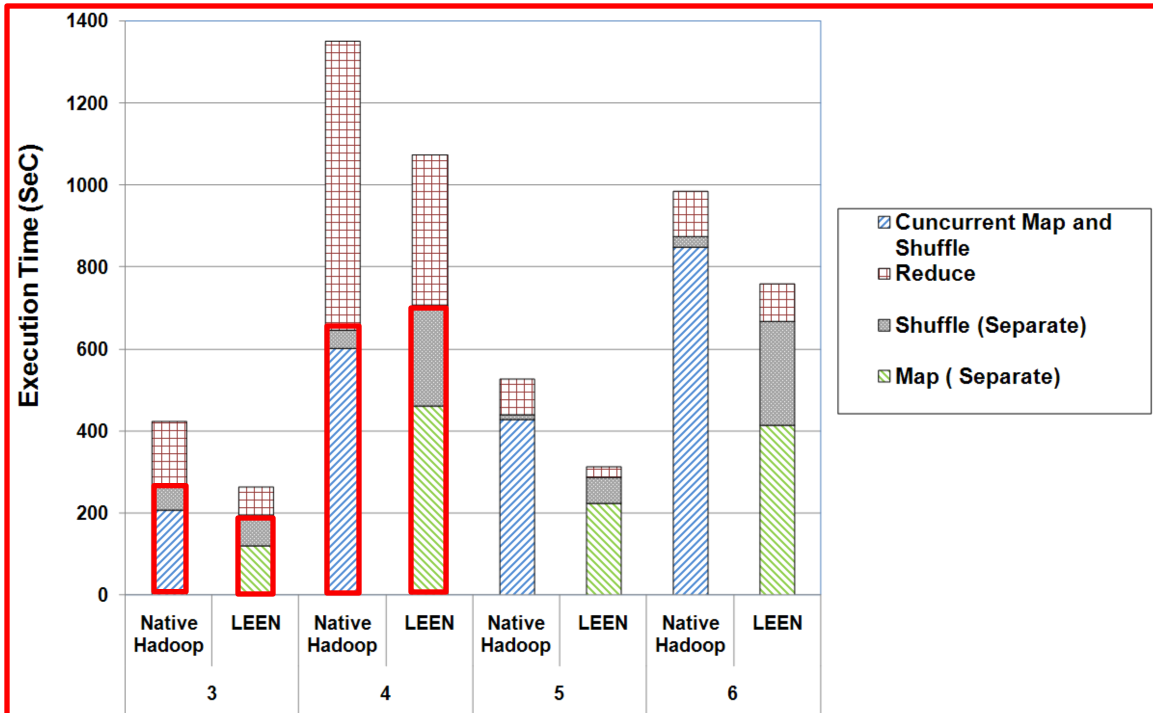


Partitioning Skew





	3	4	5	6
Locality Range	1-85%	15-35%	1-50%	1-30%



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