

An Efficient Key Partitioning Scheme for Heterogeneous MapReduce Clusters

Muhammad Hanif*, Choonhwa Lee*

* Division of Computer Science and Engineering, Hanyang University, Seoul, Republic of Korea

honeykhan@hanyang.ac.kr, lee@hanyang.ac.kr

Abstract— Hadoop is a standard implementation of MapReduce framework for running data-intensive applications on the clusters of commodity servers. By thoroughly studying the framework we find out that the shuffle phase, all-to-all input data fetching phase in reduce task significantly affect the application performance. There is a problem of variance in both the intermediate key's frequencies and their distribution among data nodes throughout the cluster in Hadoop's MapReduce system. This variance in system causes network overhead which leads to unfairness on the reduce input among different data nodes in the cluster. Because of the above problem, applications experience performance degradation due to shuffle phase of MapReduce applications. We develop a new novel algorithm; unlike previous systems our algorithm considers a node's capabilities as heuristics to decide a better available trade-off for the locality and fairness in the system. By comparing with the default Hadoop's partitioning algorithm and Leen algorithm, on the average our approach achieve performance gain of 29% and 17%, respectively.

Keywords— Hadoop; MapReduce; Cloud Computing; Context-awareness; Heterogeneous system

I. INTRODUCTION

Hadoop [1] is a standard open-source implementation of Google's MapReduce [3] programming model for processing large amount of data in parallel. Hadoop was developed predominantly by Yahoo; where it processes petabyte scale data on tens of thousands of nodes [4], and is used by other companies including Facebook, Amazon, and New York Times.

The Hadoop system runs on top of the Hadoop Distributed File System [6], within which data is loaded, partitioned into splits, and each split replicated across multiple machines. Data processing is co-located with data storage: when a file needs to be processed, the Resource Manager consults a storage metadata service to get the host node for each split, and then system schedules a task on that node, so that data locality is exploited efficiently. The map task processes a data split into key/value pairs, on which hash partitioning function is performed, on the appearance of each intermediate key produced by any running map within the MapReduce system:

hash (Hash code (Intermediate-key) Modulo Reduce-ID)

These hashing results are stored in memory buffers. In the reduce stage, a reducer takes a partition as input and performs the user defined reduce function on the partition. The storage

distribution of the hash partition among the nodes affects the network traffic and the balance of the hash partition size play a significant role in the load-balancing among the reducer nodes.

This work scrutinizes the problem of variance in both the intermediate key's frequencies and their distribution among data nodes throughout the cluster in Hadoop's MapReduce system. This variance in system causes network overhead which leads to unfairness on the reduce input among different nodes in the cluster. Because of the above problems, applications experience performance degradation due to network overhead in the shuffle phase of MapReduce kind applications. The current Hadoop's default hash partitioning and Leen [7] partitioning work well in case of the uniform distribution of the data throughout the cluster in homogeneous systems. But in case of heterogeneous machines cluster, the system performance degrades due to the lack of consideration of heterogeneity of nodes and also the random (non-uniform) data distribution of data set throughout the cluster.

To mitigate the problems of partitioning and computation skew, we develop an algorithm which considers the node heterogeneity as heuristics to manage the data locality and fairness trade-off in the system by load-balancing according to the capabilities of nodes. Our algorithm saves the network bandwidth overindulgence during the copying phase of intermediate data of MapReduce job along with balancing the reducers input. It improves the data locality in the individual nodes by asynchronous map and reduces approach, in this manner having more control on keys dissemination in each data node.

Contribution of this work includes,

- Extension of node/locality/fairness-aware execution model for the partitioning scheme for Hadoop.
- A node-aware or hardware-aware algorithm to discover data locality and fair key distribution to achieve load-balancing in cluster according to the capabilities of nodes.
- Mitigate the partitioning and computation skew and achieve reduction in network overhead in the cluster in comparison to default Hadoop's hash partitioning and Leen partitioning in heterogeneous environment.

The rest of the paper is organized as follows. Section 2 discusses the motivational background. Section 3 illuminates

the proposed scheme. The performance is evaluated in section 4 and the work is concluded in section 5.

II. MOTIVATIONAL BACKGROUND

There are different aspects of the Hadoop scheduler which should be manipulated for the improvement of existing schedulers and mitigating the problem with those existing schedulers. Our main motivations of this work are some assumptions made by existing schedulers and situations where Hadoop's existing schedulers perform worse.

A. Hadoop's Limitations

Hadoop's scheduler makes several implicit assumptions:

- Nodes can perform work at roughly the same rate i.e. Homogeneity.
- Tasks progress at a constant rate throughout time.
- A task's progress score is evocative of fraction of its total work that it has done. Specifically, in a reduce task, the copy, sort and reduce phases each take about 1/3 of the total time.
- Tasks incline to finish in waves, so a task with a low progress score is likely a straggler.
- Tasks in the same category (map or reduce) require roughly the same amount of work [3].

As we shall see, assumptions 1 and 2 break down in a virtualized data centre due to heterogeneity. Assumptions 3 and 4 can break down in a homogeneous data centre as well, and may cause Hadoop to perform poorly there too. In fact, Yahoo disables speculative execution on some jobs because it degrades performance, and monitors faulty machines through other means. Facebook disables speculation for reduce tasks [9]. Assumption 5 is intrinsic in the MapReduce paradigm, so we do not address it in this paper.

B. Leen's Limitations

Leen works well under some conditions and situations/scenarios, while there are situations where Leen cannot work properly.

- Leen assumption of uniform distribution of the keys throughout the cluster's nodes does not hold in most of the real world situations.
- It does not consider any heterogeneity, which is not the case in real world system.
- It does not consider the numbers of keys throughout the cluster in calculation of the FLK, it just consider the locality on the basis of average numbers of keys i.e. Mean.
 - This hurt the load-balancing in the system especially when best locality node is slower one.

The 2nd point of consideration of only homogeneous machines degrades the performance in both virtualized and non-virtualized situations. In a non-virtualized data centre, there may be multiple generations of hardware. In a virtualized data centre where multiple virtual machines run on each physical host, such as Amazon EC2, co-location of VMs may cause heterogeneity. In EC2, co-located VMs use a host's

full bandwidth when there is no contention and share bandwidth fairly when there is contention [10].

C. Motivational Example

For the reference, we use a similar example of nine keys like Leen [7] algorithm and use it as a comparative example between the different partitioning schemes. As shown in Fig. 1, the sum of the entire nine keys frequencies is 225 keys, distributed randomly in the cluster of three data nodes: Node1, Node2 and Node3. Also the keys occurrences are wide-ranging along with the dispersal among the data nodes.

pp	Nodes / Keys	K1	K2	K3	K4	K5	K6	K7	K8	K9	Locality
x	Node1	12	9	8	4	17	2	1	3	0	23%
2x	Node2	24	17	5	5	3	0	1	8	1	37%
3x	Node3	17	28	19	14	13	5	3	5	1	33%

Figure 1. Illustrate the current hash partitioning. The keys are ordered by appearance while each value represent the frequency of key in the data node

Fig. 1 shows that the key partitioning results using the default Hadoop Hash partitioning, which clarify that it is scant in case of the partitioning skew in terms of data size which needs to be shuffled through the system network and balance distribution of reducer's input. In this example, the percentages of keys locally partitioned on each of the three nodes are 23%, 37% and 33%. And the Total Network Traffic is 156 keys out of 225 keys. This kind of situation creates different problems like poor resource utilization and performance degradation especially in heterogeneous environment.

pp	Nodes / Keys	K1	K2	K3	K4	K5	K6	K7	K8	K9	Locality
x	Node1	12	9	8	4	17	2	1	3	0	36%
2x	Node2	24	17	5	5	3	0	1	8	1	59%
3x	Node3	17	28	19	14	13	5	3	5	1	45%

Figure 2. Using Leen, locality increases as compared to hash partitioning

Leen [7], which is an improvement to the default hash partitioning of the Hadoop system, performs well in some situations. It performs worse in some situation because it does not consider the non-uniform distribution of data throughout the data nodes and does not take into account the heterogeneity of the nodes which is the case in most of the real world scenarios. The total network traffic is to transfer 150 keys out of 225 keys, which decreases the numbers of keys transfer over the network and leads to around 2% improvement over the hash partitioning as shown in fig.2.

III. PROPOSED APPROACH

We introduce a new metric NA-FLK, which consider the node heterogeneity in the cluster. There is always a trade-off between the locality and fairness in heterogeneous systems, so we use a weightage model where users can choose the ratio between the locality and fairness. By default the locality-fairness ratio will be 60% to 40% i.e. 60% weightage to locality while 40% weightage to fairness. For this we will use two new properties, `<mapred.fairness.weightage>` and `<mapred.locality.weightage>`.

A. Measuring Heterogeneity

Afore implementing our partitioning algorithm, we need to measure the heterogeneity of Hadoop cluster in terms of data processing speed. Such processing speed highly depends on data-intensive applications. Thus, heterogeneity measurements in the cluster may change while executing different MapReduce applications. We introduce a metric “processing power ratio”, to measure each node’s processing speed in a heterogeneous cluster. Processing power ratios are determined by a sketching procedure conceded out through following steps.

- The data processing operations of a given MapReduce application are separately carrying out in each node. To fairly compare processing speed, we guarantee that all the nodes process the same amount of data. For example, experiment with the same size input file i.e. 1GB.
- Put the response time of each node performing the data processing operations in an Array-List.
- Shortest response time is used as a reference to normalize the response time measurements.
- Normalized processing powers ratios are employed by partitioning algorithm to decide the trade-off between the fairness and locality.

1) Measuring Heterogeneity Example: Suppose that there are four heterogeneous nodes (i.e., Node A, B, C and D) in a Hadoop cluster as shown in table 1. After running a Hadoop application on each node, one collects the response time of the application on node A, B, C and D is 10, 20, 30 and 40 seconds, respectively. The response time of the application on node A is the shortest. Therefore, the processing power ratio of node A with respect to this application is set to 1, which becomes a reference used to determine processing power ratios of node B, C and D. Thus, the processing power ratios of node B, C and D are 2, 3 and 4, respectively. Recall that the processing power capacity of each node is quite stable with respect to a Hadoop application. Hence, the processing power ratios are free of input file sizes. Table I shows the response times and processing power ratios for each node in a Hadoop cluster.

TABLE 1. MEASURING HETEROGENEITY EXAMPLE

Node	Execution Time (seconds)	PP-Ratio	Optimal Keys Assignment
Node A	10	1	40 %
Node B	20	2	30 %
Node C	30	3	20 %
Node D	40	4	10 %

B. Partitioning Example

Continuing with section 2.3 examples, where the total network traffic was high and the locality was lower than expected. Our new node-aware scheme is very suitable for the practical scenarios because it cover most of the drawbacks of already developed schemes. This scheme can work in case of

diverse non-uniformly distributed data over the nodes and also in case of heterogonous machines in the system. In this case the total network traffic was to transfer 138 keys out of 225 keys as shown in Fig. 3, which generate less of network traffic on the average. And achieve load-balancing according to the capabilities of the nodes in the system.

PP	Nodes / Keys	K1	K2	K3	K4	K5	K6	K7	K8	K9	Locality
x	Node1	12	9	8	4	17	2	1	3	0	2%
2x	Node2	24	17	5	5	3	0	1	8	1	59%
3x	Node3	17	28	19	14	13	5	3	5	1	60%

Figure 3. Motivational Example: Using our Node-Aware partitioning scheme ourperform both Hadoop’s hash partitioning and Leen partitioning

C. Mathematical Model

Our proposed algorithm is Node, locality and fairness-aware key partitioning in Hadoop system. For the effective partitioning of data set of K keys distributed over N nodes in cluster, there will be best possible solution in K^N solutions. To find suitable solution of all these possible ways, we use processing power or capabilities of nodes as heuristic. We keep in mind that for the best solution, we need to find a good trade-off between the locality of keys and fairness to the reduce nodes. After finding the processing power of the nodes, we need to find the optimal load for the reducers, which depend on the numbers of reducers.

To calculate the optimal load for each reducer, we need to use the optimal load percentage table with total data set,

$$OptimalLoadN_j^i = OptimalLoad_j * DataSetK$$

For the locality of keys in a specified node, we use the frequencies of keys partitioned to that node divided by the optimal load for the specified node (instead of just the mean of all which is wrong in most practical world applications).

$$Locality_j = \frac{\text{partitioned keys}}{OptimalLoadN_j^i}$$

The fairness in the system could be calculated as follows,

$$Fairness - S = \sqrt{\frac{\sum_{j=1}^n (FK_j^i - OptimalLoadN_j^i)^2}{Nos\ of\ Reducers}}$$

where FK_j^i point to the frequency of key K_i in the data node n_j , and $OptimalLoadN_j^i$ represents the optimal load on reducer according to their computing power. The best locality indicates partitioning K_i to the data node n_j which has the maximum frequencies for the key. The total network traffic in the cluster can be calculated as,

$$NW\ Traffic = \sum (TotalMapOutput_j * (1 - Locality_j))$$

IV. PERFORMANCE EVALUATION

The experimental results shows that the our approach NoLFA (Node Locality Fairness-Aware) algorithm over-run Leen and Hash partitioning by decreasing the total network traffic in the cluster as shown in the figure. 4. Hash partitioning is the default of the Hadoop system which on the average generates around 70% of the cluster’s network traffic while Leen improve it and crafts around 67% of network traffic in the cluster on the average. And our algorithm

NoLFA performs best on average and creates around 61% of the total network traffic on the average.

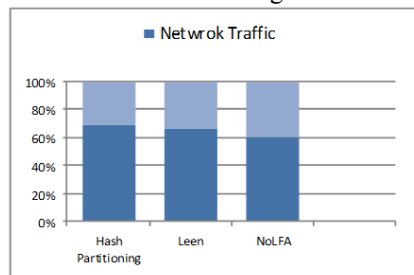


Figure 4. Percentage of total network traffic generated in the cluster

Fig. 5 shows that the optimal load-balancing for all the nodes and the load balancing achieved by other hash partitioning, Leen and NoLFA. From the results, it is clear that there is a trade-off between load balancing and locality of keys throughout the Hadoop's cluster. So the result shows that the NoLFA is better in selecting the trade-off between load balancing and locality because it considers the heterogeneity of nodes in cluster while others does not consider any heuristics and assign on the basis of static decided values.

Nodes	Processing Power	Optimal Load	Hash Load-Balance	Leen Load-Balance	NoLFA Load-Balance
Node1	X	17%	36%	20%	2%
Node2	2X	33%	46%	42%	42%
Node3	3X	50%	18%	38%	56%

Figure 5. A sample line graph using colors which contrast well both on screen and on a black-and-white hardcopy

With the better load balancing ability of NoLFA using node computing power as heuristics, it attains decrease in the execution time. Fig. 6 shows the normalized execution time of the hash, Leen and NoLFA. For the normalization, we use NoLFA as base for the calculation. Leen is about 0.22X slower on average as compare to our algorithm while Hash partitioning takes approximately 0.6X times extra time as compare to our NoLFA partitioning algorithm's execution time.

V. CONCLUSIONS

Hadoop affords simplified implementation of MapReduce framework, but its design stances challenges to attain best performance in application execution due to tightly coupled shuffle, obstinate scheduling and partitioning skew. In this paper, we developed an algorithm which takes node capabilities as heuristics to achieve better trade-off between locality and fairness in the Hadoop MapReduce system. It effectively improves the data locality and by comparing with the default Hadoop's partitioning algorithm and Leen algorithm, on the average our approach achieves performance gain of 29% and 17%, respectively.

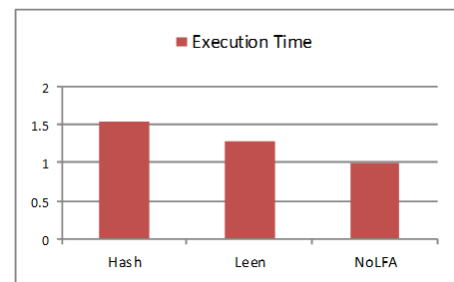


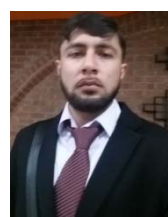
Figure 6. Normalized execution time with NoLFA as base for normalization

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2013R1A1A2007616).

REFERENCES

- [1] (2015) Apache Hadoop. [Online]. Available: <http://hadoop.apache.org/>
- [2] (2015) Amazon EC2. [Online]. Available: <http://aws.amazon.com/ec2/>
- [3] J. Dean and S. Ghemawat, "MapReduce," *Commun. ACM*, vol. 51, no. 1, p. 107, 2008.
- [4] (2015) Yahoo! Developer Network. [Online]. Available: <https://developer.yahoo.com/blogs/hadoop/yahoo-launches-world-largest-hadoop-production-application-398.html>
- [5] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," *DEC technical report TR301*, vol. cs.NI/9809, no. DEC-TR-301, pp. 1–38, 1998.
- [6] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," *2010 IEEE 26th Symp. Mass Storage Syst. Technol.*, pp. 1–10, 2010.
- [7] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and Q. Li, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in *Cloud Computing Technology and Science (CloudCom), IEEE Second International Conference on*, pp. 17–24, IEEE, 2010.
- [8] Condie, Tyson, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears, "MapReduce Online," in *NSDI'10 Proceedings of the 7th USENIX conference on Networked systems design and implementation*, vol. 10, no. 4, p. 20, 2010.
- [9] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *Ossi*, pp. 29–42, 2008.
- [10] (2015) Amazon EC2 Instance Types. [Online]. Available: <http://aws.amazon.com/ec2/instance-types/>



Muhammad Hanif received his B.S. degrees in computer software engineering from UET Peshawar, Pakistan. He is a combined Master and PhD candidate in Computer Software Engineering at Hanyang University, South Korea. His research interest includes Cloud & Distributed Computing, Big Data and Distributed Scheduling.



Choonghwa Lee is an associate professor in the Division of Computer Science and Engineering at Hanyang University, Seoul, South Korea. He received his B.S. and M.S. degrees in computer engineering from Seoul National University, South Korea, in 1990 and 1992, respectively, and his Ph.D. degree in computer engineering from the University of Florida, Gainesville, in 2003. His research interests include cloud computing, peer-to-peer and mobile networking and computing, and services computing technology.