Capacity-aware Key Partitioning Scheme for Heterogeneous Big Data Analytic Engines

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*—Big data and cloud computing became the centre of interest for the past decade. With the increase of data size and different cloud application, the idea of big data analytics become very popular both in industry and academia. The research communities in industry and academia never stopped trying to come up with the fast, robust, and fault tolerant analytic engines. MapReduce becomes one of the popular big data analytic engine over the past few years. 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 problems, applications experience performance degradation due to shuffle phase of MapReduce applications. We develop a new novel algorithm; unlike previous systems our algorithm considers each 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 partitioning algorithm: a). In case of 2 million key-value pairs to process, on the average our approach achieve better resource utilization by about 19%, and 9%, in that order; b). In case of 3 million key-value pairs to process, our approach achieve near optimal resource utilization by about 15%, and 7%, respectively.**

***Keyword*—Cloud and Distributed Computing, Context-aware Partitioning, Hadoop MapReduce, Heterogeneous Systems**

I. INTRODUCTION

BIG DATA [1] is getting bigger day by day with the information coming from instrumented, steady supply chains transmitting real-time data about the variabilities in everything from e-trading to weather. Furthermore, cautious information has in full swing through amorphous digital channels like social media, smart phones applications and different IoT devices. This big amount of data has challenges involve with it: data is big, it is fast, unstructured, has enormous amount of sources, and contains graphics. Cloud computing [2] becomes the interest point for both industry and academia due to its scalable, distributed and fault tolerant storage services and applications which have the aptitude to handle the challenges associated with big data. The data processing of big data in cloud and distributed computing environment is one of the core delinquents and under the spot light in research community for a while. MapReduce has proven to be the most popular implementation of computational processing framework which has the capability of supporting distributed storage holding large scale data over the distributed infrastructure like cloud computing.

Google's MapReduce [3] programming model is an emerging data intensive programming model for large scale data parallel applications including data mining, web indexing, multilinear subspace learning, business intelligence, and scientific simulations. MapReduce facilitates users with an easy parallel programming interface in distributed computing paradigm. It is used for distributed fault tolerance for supervision of multiple processing nodes in the clusters. One of the most significant feature of MapReduce is its high scalability that permits users to process massive amount of data in short time. There are numerous fields that benefit from MapReduce including bioinformatics [4], scientific analysis [5], web data analytics, security [6], and machine learning [7]. Hadoop [8] [9]is a standard open-source implementation of Google's MapReduce 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 [10] [11], and has been successfully adopted by several companies including Amazon, AOL, Facebook, and New York Times. For example, AOL uses it for running behavioural pattern analytics application which analyses the behavioural pattern of their users so as to targeted services on the basis of their location, interest and so on.

The Hadoop system runs on top of the Hadoop Distributed File System [12], within which data is loaded, partitioned into splits, and each split replicated across multiple nodes. Data processing is co-located with data storage: when a file needs to be processed, the Resource Manager consults a storage

Manuscript received April 26, 2017. This research work is follow-up of the invited journal to an accepted eminent conference paper of the 18th International Conference on Advance Communication Technology (ICACT2016). This work is supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2017R1A2B4010395).

Muhammad Hanif is with the Department of Computer and Software Engineering, Hanyang University, Wangsimni-ro 222, Seoul, Republic of Korea. (E-mail: honeykhan@ hanyang.ac.kr).

Choonhwa Lee is with the Division of Computer Science and Engineering, Hanyang University, Wangsimni-ro 222, Seoul, Republic of Korea. (Corresponding author, Phone: +82-2-2220-1268; fax: 82-2-2220-1723; e-mail: lee@ hanyang.ac.kr).

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 [13] 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-ness (non-uniform) in data distribution of data set throughout the cluster.

To alleviate the problems of partitioning and computation skew, we develop an algorithm which considers the node heterogeneity (i.e. the capacity of each node in the cluster) as heuristics to manage the data locality and fairness trade-off in the system by load-balancing according to the capabilities of nodes in the cluster. 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 decoupling mappers and reducer tasks, in this manner having more control on keys dissemination in each data node of the cluster.

Contribution of this work includes,

- Extension of node/locality/fairness-aware execution model for the partitioning scheme for Hadoop.
- A node-aware or capacity-aware algorithm to ascertain data locality and fair key distribution to achieve load-balancing in cluster according to the capabilities of nodes.
- Automatize the suboptimal trade-off between locality, load balancing, and fairness.
- Mitigate the partitioning and computation skew and achieve reduction in network overhead in the cluster in comparison to default state of the art partitioning schemes in heterogeneous environment.

The rest of the paper is organized as follows. Section 2 discusses the motivational background. Section 3 illuminates the system architecture, while the proposed scheme is discussed in section 4. The performance is evaluated in section 5. Section 6 discusses the related work and the paper is concluded in section 7.

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. In this section, we will justify our motivation of the work by going through the limitations of the previous state of the art approaches and demonstrating through motivational example by carrying out a series of experiments to validate the aforementioned problems in the current Hadoop implementation.

Hadoop's Limitations

Default Hadoop's system makes several implicit assumptions:

i. All nodes in the cluster can perform work at roughly the same rate i.e. the cluster is consist of Homogeneous machines.
ii. Tasks progress at a constant rate throughout time.
iii. 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. Which is not the case in real life examples, i.e. jobs and tasks can be of different types such as CPU intensive, IO intensive, or Memory intensive.
iv. Tasks incline to finish in waves, so a task with a low progress score is likely a straggler.
v. 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 of the resources. 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 in order to achieve better performance [14] [15]. Assumption 5 is intrinsic in the MapReduce paradigm, so we do not address it in this paper.

Leen's Limitations

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

i. Leen assumption of uniform distribution of the keys throughout the cluster's nodes does not hold in most of the real world situations, as the real world input data set are usually distributed and non-uniform.
ii. It does not consider any heterogeneity, which is not the case in real world system. Almost all the data centres in the industry consists of heterogeneous machines such as Amazon EC2 [16], Microsoft Azure [17]
iii. 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.

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

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

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

Fig. 2. Using Leen partitioning scheme, it increases locality as compared to the default Hadoop's hash partitioning.

mathematical mean value of it.

a. 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 at the same data centre as in case of upgrading some system to the new generation whereas other remain intact. 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 [16].

Motivational Example

As shown in Fig. 1, there are three nodes: Node1, Node2, and Node3, with nine intermediate keys, ordered by their influx during the map tasks execution. For the reference, we use a similar example of nine keys like Leen [13] algorithm and use it as a comparative example among the different partitioning schemes. The sum of the entire nine keys frequencies is 225 keys, distributed randomly in the cluster of three data nodes, which is usually the case in distributed infrastructure. Also the keys occurrences are wide-ranging along with the dispersal among the data nodes.

Fig. 1 shows that the key partitioning results using the default Hadoop Hash partitioning, which is assigning K1, K4, and K7 to Node1; K2, K5, and K8 to Node2; whereas K3, K6, and K9 to Node3. So despite the fact that Node3 has the highest processing capability, Node1 needs to process 81 out of 225, Node2 needs to process 103 out of 225, and finally Node3 needs to process 41 out of 225 key-value pairs leading to non-optimal utilization of the resources. This 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. We discern that the data size needs to be transmitted through the network in the shuffle phase is enormous, and the hash partitioning is inadequate in the presence of partitioning skew. In this example, the percentages of keys locally partitioned on each of the three nodes are 23%, 37% and 33%, respectively. And the Total Network Traffic is 156 keys out of 225 keys. According to the processing power of the Nodes in the given example shown in the Table.1, Node1 process 81 keys-value pairs in 36 units of time, Node2 process its 103 key-value pairs in 23 units of time, and Node3 process its 41 assigned key-value pairs in 6 units of time, which prove the hypothesis of non-optimal utilization resources that Node3 stay idle for about 30 units of time (36 units for Node1 – 6 units for Node3). This kind of situation creates different problems like poor resource utilization and performance degradation especially in heterogeneous environment.

Leen [13], which is an improvement to the default hash partitioning of the Hadoop system, performs well in some situations, specifically, in case of homogeneous cluster. It performs worse in some situation because it does not consider the non-uniform distribution of data throughout the data nodes in the cluster, as well as does not take into account the heterogeneity of the nodes which is the case in most of the real world scenarios. Continuing the example above, Leen assigns K5, K6, and K7 to Node1; K1, K4, K8, and K9 to Node2; whereas K2, and K3 to Node3. Leading to the fact that Node1 needs to process 45 out of 225, Node2 needs to process 94 out of 225, and finally Node3 needs to process 86 out of 225 key-value pairs leading to a sub-optimal solution which is an enhanced assignment of key-value pairs than the one by default hash partitioning scheme. The percentage of keys locality in the three nodes are 36%, 59%, and 45%, respectively. 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. Bestowing to the previous calculations, Node1 process 45 keys-value pairs in 20 units of time, Node2 process its 94 key-value pairs in 21 units of time, and Node3 process its 86 assigned key-value

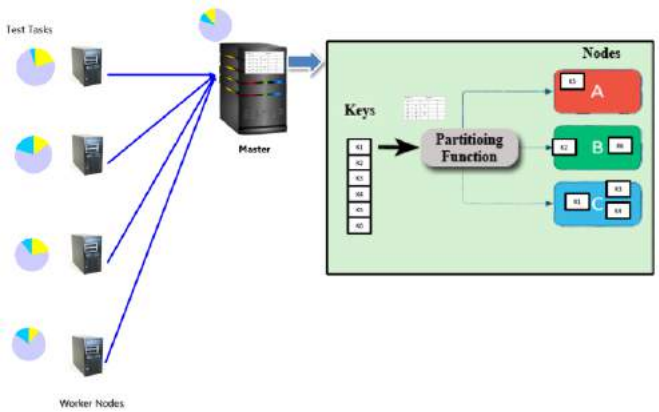

Fig. 3. Proposed scheme system architecture.

pairs in 13 units of time, in which case Node3 stay idle only for about 8 units of time instead of 30 units as in the case of default hash partitioning (i.e. 21 units for Node2 – 13 units for Node3). This example shows that the Leen partitioning algorithm help the system to improve the utilization of the whole cluster eventually.

By the above reasoning, we have to conclude that the previous work lack of contemplation of the capacity awareness of the nodes superintends any opportunities of the reduction of the network traffic during the shuffle phase of the MapReduce application execution, in case of heterogeneity in the cluster. Also the load misbalancing data distribution of reducer nodes occurs, i.e. 1). Nodes with higher capacity get less amount of data leading to non-optimal utilization of resources and under loading, 2). Lower capacity nodes getting more data to process leading to performance degradation, overloading, and straggler effect.

III. System Architecture

In this section, we will introduce the system architecture and how the proposed system work in the specified environment. As mentioned earlier, we decouple the mappers and reducers in order to achieve more parallelism and keep track of all the intermediate data keys frequencies and distribution in the form of capacity-keys frequency table. In order to meritoriously partition certain input data set of K keys distributed over N nodes in a cluster, the system need to find the best available solution in a space of K^N possible combinations. The system achieve it through the proposed approach which will be explained in the forthcoming section.

The system architecture is consists of a master node and a number of worker nodes as shown in the Fig. 3, and it works as subsequent way. The system first run some test tasks on the worker nodes over the cluster which send the results back to the master node. The master node uses the gather information of each worker node in the cluster and keep track of the execution time of each node in the cluster for the jobs run by the specified node. The master node then estimates the processing power ratio using the sample task run results. Then master node constructs node-capacity table which is further used in the edifice of capacity-keys frequency table. As the master node already knows the input keys data distribution over the cluster, the formation of capacity-keys frequency table take place using the perceptibly known information required. Then this table is being forwarded to the task scheduler. The task scheduler schedule different keys to different node while taking all the available information into account, in order to get the sub-optimal trade-off between fairness, locality and load-balancing. And as a result of this procedure, every node in the cluster get the numbers of tasks and partitions of the input data suitable to their processing power.

TABLE I
EXAMPLE OF MEASURING HETEROGENEITY

Node	Execution Time	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%

*Execution units used are seconds here.

IV. Proposed Approach

In this section, we will thoroughly explain the proposed approach in three different subsections. First, we will explain the details about how to measure the heterogeneity of different nodes exists within the cluster. Then will move to demonstrating the effectiveness of the proposed approach with the help of continuing the same example from section 2. Finally, the details of the mathematical model used in the system will be elucidated in the last subsection of this section.

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>. With these properties, we gives the administrator the power to decide which of the metric is more valuable to their organization according to their SLA with users.

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 data processing applications. We introduce a metric “processing power ratio”, to measure each node’s processing speed in a heterogeneous cluster upon execution of new application execution, insertion of new node, or failure of node in the 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 input data. For example, experiment with the same size input file of 1GB to process by the specified node.

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

Fig. 4. Motivational Example: Using our Capacity-Aware partitioning scheme outperform both Hadoop's hash partitioning and Leen partitioning.

- The response time of each node performing the data processing operations will be recorded in an Array-List data structure.
- Shortest response time is used as a reference to normalize the response time measurements.
- The normalized processing powers ratios are employed by the partitioning algorithm of the system while taking decision of the trade-off between the fairness, load-balancing, and locality.

Measuring Heterogeneity Example: Suppose that there are four heterogeneous nodes: Node A, B, C and D, in a Hadoop MapReduce 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 any specified Hadoop analytic application. Hence, the processing power ratios are free of input file sizes. Table I shows the response times, processing power ratios, and optimal keys assignment percentage for each node in the cluster. As we can grasp, the optimal keys assignment percentage with the value of 40% for Node A is the highest, 30% for Node B, 20% for Node C, and 10% for Node D, so the scheduler with the suboptimal solution will get the nearest possible values for each node in the cluster leading to a load-balanced cluster.

B. Partitioning Example

Continuing with section 2-C motivational examples, where the total network traffic was high and the locality was lower than expected. Our proposed Capacity-aware scheme is very much appropriate for the practical scenarios because it cover most of the drawbacks of previously 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. Continuing with the motivational example, as shown in Fig. 4, the proposed scheme NoLFA assigns K7 to Node1; K1, K4, K8, and K9 to Node2; whereas K2, K3, K5, and K6 to Node3. Prominently leading towards the datum that Node1 needs to process 5 out of 225, Node2 needs to process 94 out of 225, and Node3 needs to process 126 out of 225 key-value pairs, which give us a near-optimal solution of assignment of key-value pairs. The percentage of keys locality in the three nodes are 2%, 59%, and 60%, respectively. The total network traffic is to transfer 138 keys out of 225 keys, which decreases the numbers of keys transfer over the network and leads to around 8% improvement over the hash partitioning scheme. According to the processing power of the Nodes in the given example, Node1 process 5 keys-value pairs in 2 units of time, Node2 process its 94 key-value pairs in 21 units of time, and Node3 process its 126 assigned key-value pairs in 18 units of time. Thus, NoLFA achieves better load-balancing according to the capabilities of the nodes in the system.

C. Mathematical Model

Our proposed algorithm introduces heterogeneity awareness into the default Hadoop scheduling system by altering the hash key partitioning scheme while taking locality of data and fairness into account. 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 in the proposed scheme. We keep in mind that for the best solution, we need to find a good trade-off between the locality of keys-value pairs, load-balancing, and fairness to the reduce nodes throughout the cluster. After estimating the processing power of the nodes, we need to find the optimal load for the reducers depending on the numbers of reducers according to the SLA based configuration of the cluster.

To calculate the suboptimal load for each reducer in the cluster, 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 athematic mean of all, which is wrong in most practical world applications).

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

As we can see in the above equation, the locality of each node j is the ration of partitioned keys to the Optimal Load from the already calculated table. The best locality node is usually the node which contains maximum frequencies of a key, and that key is partitioned to that node. The fairness in the system could be calculated as follows,

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

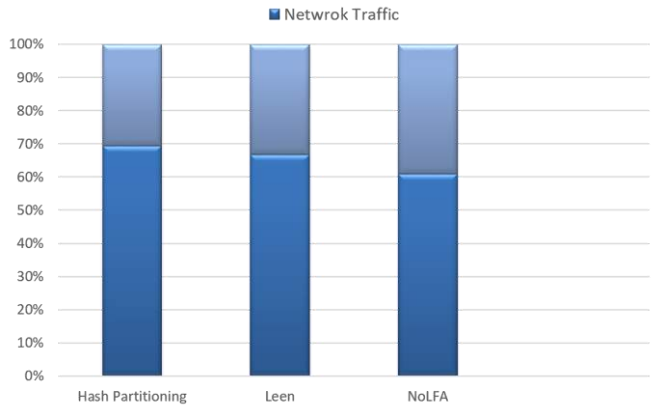

Fig. 5. Percentage of total network traffic generated in the cluster.

where FK_j^i points 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,

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

With this formula, we can get an educated guess of the network overhead in the cluster with the combined effect of both the total intermediate data and each node assessed locality.

V. PERFORMANCE EVALUATION

To evaluate the performance of the proposed algorithm, we have designed and execute certain set of experiments with different variations of keys and frequencies distribution. The experimental results shows that the our proposed approach NoLFA algorithm over-run Leen and Hash partitioning by decreasing the total network traffic in the cluster as shown in the Fig. 5. Hash partitioning is the default partitioning scheme of the Hadoop data processing framework, which on the average generates around 70% of the cluster's network traffic, whereas Leen improve it and crafts around 67% of network traffic in the cluster on the average. And our algorithm NoLFA outperforms both of the above partitioning scheme by achieving on average better results, and creates around 61% of the total network traffic. This is because of the fact that NoLFA considers the capacity of each machine/node in the cluster while taking the decision about the partitioning of different keys to different nodes in the cluster.

The second set of experiments focus on the load-balancing problem in the Hadoop scheduling systems. As Fig. 6 shows that the processing power of three different nodes, the desired optimal load-balancing for all three nodes, and the load balancing achieved by each partitioning scheme including hash partitioning, Leen, and NoLFA. From the domino effect, it is clear that there is a trade-off between load balancing and locality of key-values pairs throughout the Hadoop's cluster. So the outcome shows that the NoLFA perform better in selecting the trade-off between load balancing and locality because it considers the heterogeneity of nodes in cluster whereas others does not consider any such heuristics and assign on the basis of static decided values.

Through better load balancing ability of NoLFA using node computing power as heuristics, it attains decrease in the execution time of the overall application. Fig. 7 shows the normalized execution time of each partitioning scheme designed for this set of experiments i.e. Hadoop's default hash partitioning scheme, Leen, and NoLFA. For the normalization effect, we use NoLFA as base for the calculation. Leen is about 0.22X slower on average as compare to our algorithm whereas Hash partitioning takes approximately 0.6X times extra time as compare to our NoLFA partitioning algorithm's execution time. With the elucidation of Fig. 8, we illustrate the average resource utilization of cluster resources by Hadoop, Leen, and NoLFA, respectively. X-Axis shows different schemes such as Hadoop, Leen, and NoLFA. Y-Axis shows the average percentage of cluster resource utilization by different schemes. Whereas Z-Axis shows the change in the data size in units of numbers of key-values pairs processed in each case study. As we can surmise from Fig. 8, the blue bars at the front represents the state of affairs when the numbers of key-values pairs are 2 million. The red bars in the back represents the results in case of 3 million key-value pairs been processed by each scheme. We can deduct from these case studies that Leen and NoLFA keep the trend of outperforming the default Hadoop partitioning scheme in both cases, as the numbers of key-values pairs increased in the experiment. The utilization increases with the number of key-value pair increases until the saturation of data to the nodes in the cluster.

Finally, Fig. 9 signposts the network traffic overhead in all six instance of the experiments for the Hadoop with number of key-value pairs to process set to 2 million and three million;

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

Fig. 6. Trade-off between Load balancing and Fairness.

Fig. 7. Normalized execution time with NoFLA as base for normalization.

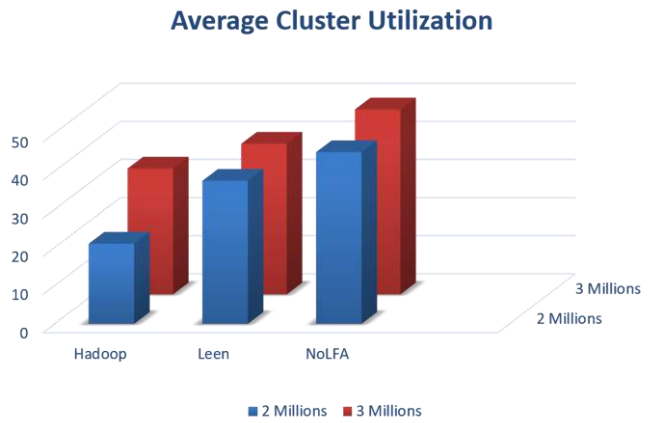

Fig. 8. Average performance gain of Hadoop, Leen, and NoLFA. Results are normalized according the number of key-value pairs processed by each scheme accordingly.

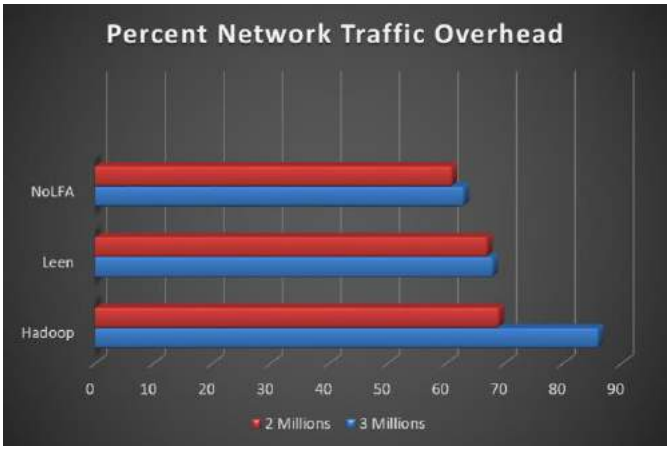

Fig. 9. Percent Network Traffic Overhead vs Numbers of Key-value pairs. The upper red bars shows the value for 2 million key-value pairs while the lower blue bar show it for 3 million key-value pairs to process.

for Leen with the number of key-value pairs set to 2 million and 3 million; and for NoLFA with the key-value pairs set to 2 million and 3 million, accordingly. X-axis shows the average percent network traffic overhead caused in each instance of experiment. Y-axis shows different partitioning techniques used in the model experiments. The top bottom blue bar shows the case when the number of key-values pairs to be processed is 3 million, whereas the top red bar represent the case of 2 million key-value pairs to be processed, accordingly. The results shows that the network overhead increases as the amount of data need to be processed increases in each instance of experiment.

With the above reasoning, we claim that it is clear that capacity awareness play an important role in selection of the partitioning different keys to different nodes in the cluster, and has a positive influence on the overall performance and near optimal utilization of the cluster.

VI. Related Work

Previous work aiming to improve the performance of MapReduce system achieved the desired goal through various approaches including reduction of network cramming by inserting partial data awareness into the shuffle phase, skew mitigation, replica awareness, and network awareness.

Authors in [18] proposed two schemes of pre-fetching and pre-shuffling for communal MapReduce environments. Pre-fetching use data locality and assign tasks to nearest node to the data block, whereas pre-shuffling reduce network overhead of slouching the key-value pairs. Our scheme NoLFA decouple the mapper and reducer tasks and scan over the keys frequency table generated upon execution of map phase and cross reference it with the capacity table created after executing the sample jobs on the nodes in the cluster to achieve the goal of partial balanced reduce tasks throughout the cluster. ShuffleWatcher [19] proposed a multi-tenant Hadoop scheduler that tries to curtail the network traffic in shuffle phase while maintaining the particular fairness constraints of the system. The working principle of the ShuffleWatcher is on the basis of the following three steps. First, it limit the intra-job map shuffle according to the network traffic load. Second, it auspiciously apportion the map tasks to localize the intermediate data. Finally, it exploit the confined intermediate data and delayed shuffle to reduce the network traffic in shuffle phase by favorably scheduling reduce tasks in nodes crofting the intermediate data. Unlike ShuffleWatcher, NoLFA take the capacity information of each node in the cluster whereas distributing the tasks which is very helpful in case of the heterogeneity in the cluster. EC-Cache [20] introduced a load-balanced, low latency cluster cache via erasure coding to overawed the inadequacy of selective replication. It employs erasure coding through two principles. First, by splitting and erasure coding individual objects during writes. Second, late binding. These led to improving load-balancing in the system. Tang et al. proposed a sampling evaluation to solve the problems of partitioning skew and intermediate data locality for the reduce tasks called Minimum Transmission Cost Reduce Task Scheduler [21] (MTCRS). They used communication cost and waiting time of each reduce task as heuristic whereas deciding which task to assign to which node in the cluster. Their scheduling algorithm used Average Reservoir Sampling for the spawning of parameter sizes, and location of intermediate data partitions for their rummage-sale mathematical estimation model. On the other hand, NoLFA used Random Sampling.

Transferring data over the network is costly and causes performance degradation more severely in federated clusters. Kondikoppa et al. [22] introduced a network-aware scheduling algorithm for Hadoop system which work in federated clusters, improving the map tasks scheduling and

consequently tries to abate the network traffic overhead leading to improved performance gain. NoLFA has different approach of decoupling the map and reduce tasks and routinize the keys-capacity frequency table to achieve the specified goal. Locality Aware Reduce Scheduling (LARS) [23] abate data transfer in their proposed grid-enabled MapReduce framework. Due to heterogeneity awareness of nodes in the grid, the map data size varies leading to assigning map tasks associated with different data size to different worker nodes according to their computation power. The LARS algorithm will select the nodes with largest region size of the intermediate data to be the destination for the reduce tasks. NoLFA achieve the desired goal with the frequency-capacity table.

Another concern is the partitioning skew that ascends due to an unstable distribution of map output across nodes, causing a massive size of data input for some reduce tasks while lesser for others. Centre-of-Gravity (CoG) [24] reduce scheduling add locality and skew cognizance to the scheduler. They allocates the reduce tasks to nodes nearer to nodes creating the intermediate data for that listed reduce tasks. SkewReduce [25] was proposed with the intention to dazed the computation skew in MapReduce systems where the partition run time depends on the data values as well as input size. It uses a user defined cost function based optimizer to regulate the partitioning parameterization of input data to curtail the computational skew. NoLFA only consider the case where the computational time of an input partition depends upon the input data size rather than both. LEEN [13] attenuates the partitioning skew and minimalize the transfer of data using network through load balancing of the data distribution among the nodes in the cluster. It also improve the data locality of MapReduce tasks in the process. Unlike LEEN, NoLFA work in heterogeneous environment as well through our capacity awareness algorithm. Chen el al. [26] proposed Dynamic Smart Speculative technique to alleviate the problems with default speculation implementation like skew, indecorous phase percentage configuration and asynchronous twitch of certain tasks with the cost of degradation of performance for batch jobs. Whereas FP-Hadoop [27] introduces a new phase called intermediate reduce (IR) to parallelize the reduce task to efficiently tackle the reduce data skew problem. IR process the blocks of intermediate data in parallel. NoLFA has a different approach of decoupling the mappers and reducers tasks as introduced in our previous work [28].

VII. Conclusion

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 partitioning algorithm, on the average our approach achieves better performance gain and outperform both of the previously mentioned partitioning schemes.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2017R1A2B4010395).

References

[1] M. James, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," *McKinsey Glob. Inst.*, no. June, p. 156, 2011.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST Spec. Publ.*, vol. 145, p. 7, 2011.

[3] J. Dean and S. Ghemawat, "MapReduce," *Commun. ACM*, vol. 51, no. 1, p. 107, 2008.

[4] A. Matsunaga, M. Tsugawa, and J. Fortes, "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications," *2008 IEEE Fourth Int. Conf. eScience*, pp. 222–229, 2008.

[5] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu, and G. Fox, "Twister : A Runtime for Iterative MapReduce," *HPDC '10 Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput.*, pp. 810–818, 2010.

[6] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang, "Introducing map-reduce to high end computing," *2008 3rd Petascale Data Storage Work.*, pp. 1–6, 2008.

[7] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-Reduce for Machine Learning on Multicore," *Adv. Neural Inf. Process. Syst. 19*, pp. 281–288, 2007.

[8] Apache!, "Apache™ Hadoop!" [Online]. Available: http://hadoop.apache.org/. [Accessed: 19-Dec-2016].

[9] Amazon!, "Amazon Elastic Compute Cloud (EC2)." [Online]. Available: http://aws.amazon.com/ec2/. [Accessed: 19-Dec-2016].

[10] Yahoo!, "Yahoo Developer Network." [Online]. Available: https://developer.yahoo.com/blogs/hadoop/yahoo-launches-world-largest-hadoop-production-application-398.html. [Accessed: 1-Jan-2017].

[11] 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.

[12] 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.

[13] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "LEEN: Locality/fairness-aware key partitioning for MapReduce in the cloud," *Proc. - 2nd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2010*, no. 2, pp. 17–24, 2010.

[14] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," *Nsdi'10*, pp. 21–21, 2010.

[15] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments." *Osdi*, pp. 29–42, 2008.

[16] "Amazon EC2 Instance Types." [Online]. Available: https://aws.amazon.com/ec2/instance-types/. [Accessed: 3-Jan-2017].

[17] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds," *Proc. 16th ACM Conf. Comput. Commun. Secur*, pp. 199–212, 2009.

[18] S. Seo, I. Jang, K. Woo, I. Kim, J.-S. Kim, and S. Maeng, "HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment," *2009 IEEE Int. Conf. Clust. Comput. Work*, pp. 1–8, 2009.

[19] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "ShuffleWatcher: Shuffle-aware Scheduling in Multi-tenant MapReduce Clusters," *2014 USENIX Annu. Tech. Conf. (USENIX ATC 14)*, pp. 1–13, 2014.

[20] K. V Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, K. Ramchandran, and I. Osdi, "EC-Cache : Load-Balanced , Low-Latency Cluster Caching with Online Erasure Coding This paper is included in the Proceedings of the R / W," *Osdi*, 2016.

[21] X. Tang, L. Wang, and Z. Geng, "A Reduce Task Scheduler for MapReduce with Minimum Transmission Cost Based on Sampling Evaluation," vol. 8, no. 1, pp. 1–10, 2015.
[22] P. Kondikoppa, C.-H. Chiu, C. Cui, L. Xue, and S.-J. Park, "Network-aware Scheduling of Mapreduce Framework on Distributed Clusters over High Speed Networks," pp. 39–44, 2012.
[23] Y. L. Su, P. C. Chen, J. B. Chang, and C. K. Shieh, "Variable-sized map and locality-aware reduce on public-resource grids," *Futur. Gener. Comput. Syst.*, vol. 27, no. 6, pp. 843–849, 2011.
[24] M. Hammoud, M. S. Rehman, and M. F. Sakr, "Center-of-gravity reduce task scheduling to lower MapReduce network traffic," *Proc. - 2012 IEEE 5th Int. Conf. Cloud Comput. CLOUD 2012*, pp. 49–58, 2012.
[25] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skew-Resistant Parallel Processing of Feature-Extracting Scientific User-Defined Functions," 2010.
[26] Q. Chen, C. Liu, and Z. Xiao, "Improving MapReduce performance using smart speculative execution strategy," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 954–967, 2014.
[27] M. Liroz-Gistau, R. Akbarinia, D. Agrawal, and P. Valduriez, "FP-Hadoop: Efficient processing of skewed MapReduce jobs," *Inf. Syst.*, vol. 60, pp. 69–84, 2016.
[28] M. Hanif and C. Lee, "An Efficient Key Partitioning Scheme for Heterogeneous MapReduce Clusters," *Proc. 18th IEEE Int. Conf. Adv. Commun. Technol.*, 2016.

Muhammad Hanif was born in Pakistan. He received his B.S. degrees in computer and software engineering from University of Engineering and Technology (UET), Peshawar, Pakistan in 2012. He is currently pursuing his MS leading to PhD degree in Computer Software Engineering at Hanyang University, Seoul, South Korea. His current research interest includes Cloud & Distributed Computing, Big Data Analytic Engines, Stream Processing Frameworks, and Distributed Scheduling.

Choonhwa Lee was born in South Korea. He has been with the Division of Computer Science and Engineering at Hanyang University, Seoul, South Korea since 2004, and currently as Professor. He received his B.S. and M.S. degrees in computer engineering from Seoul National University (SNU), South Korea, in 1990 and 1992, respectively, and his Ph.D. degree in computer engineering from the University of Florida, Gainesville, in 2003. He worked as senior research engineer at LGIC Ltd from 1992 to 1998. He is a member of IEEE since 2004. His research interests include cloud computing, peer-to-peer and mobile networking and computing, and services computing technology.