IJCSE International Journal of Computer Sciences and Engineering Open Access

Research Paper Vol.-6, Issue-7, July 2018 E-ISSN: 2347-2693

An Improved Shuffling Approach Towards Skew Mitigation in Mapreduce

N. K. Seera[1*], **S. Taruna**[2]

[1]Research Scholar, Banasthali Vidyapeeth, Jaipur, India
[2] Institute of Engineering and Technology (IET*)*, J.K. Lakshmipat University, Jaipur, India

[*]*Corresponding Author: narinder.k2010@gmail.com*

Available online at: www.ijcseonline.org

Accepted: 14/Jul/2018, Published: 31/Jul/2018

Abstract**—** In MapReduce applications, map tasks are generally launched in parallel and are assigned equal sized input splits to work on. Thus map side skews are rare to occur. In contrast, reduce side skews are much more challenging because the shuffling of the intermediate data, partition sizes and partition assignment to worker nodes cannot be determined at early stages. Therefore it is one of the critical problems in MapReduce model which should be thoroughly studied and possible solutions need to framed. This paper studies various causes of skew and common approaches used for skew mitigation in real world applications. Paper presents a novel approach to address reduce side skew where the large volume of intermediate data is preprocessed by intermediate nodes to make the size of intermediate keys smaller. The partial results from intermediate nodes are collected, aggregated and sent to final worker nodes to generate final output. The proposed model is applicable to applications where there is no interdependency between values of similar keys. The approach used by proposed model is contrary to the approach where the data of skewed nodes is repartitioned dynamically into small fragments and assigned to idle nodes in the cluster.

Keywords**—** MapReduce, Skew Mitigation, Shuffling, Partitioning

I. Introduction

Since few years, the rapid growth of data-intensive and business-intelligence applications accelerated the use of distributed data processing tools such as MapReduce [1]. MapReduce is a widely used programming model meant for distributed computing of large scale data on a cluster of commodity hardware. It has been in great interest for processing big data applications due to its well-known powerful features such as scalability, elasticity, flexible programming model etc [49]. But processing voluminous data in a distributed fashion demands fair task distribution among different computational nodes of the cluster. Unfair task distribution in distributed environment causes unnecessary delays in task competition resulting in overall performance deterioration. In MapReduce model, skew occurs when data is assigned unevenly to processing nodes.

Researchers in this field have done pioneering work on scheduling MapReduce tasks uniformly [38]-[40]. Few studies assumed that initially the input to MapReduce tasks is uniformly distributed [9] and the intermediate results are also evenly distributed at later stages (reduce side) as they are hash partitioned. Unfortunately, all real world datasets are not always uniform in nature and the applications processing such datasets may experience skew at map side or reduce side. Examples of such applications are PageRank [2], CloudBurst [22][26], Inverted Index [5], Friends-of-Friends, Top K% [30] etc.

There are two main factors which cause skew in big data applications – internal and external. Internal factor is the result of poor application logic or unevenness in data itself. External factor refers to the heterogeneity of machines in the cluster. Some common solutions to overcome the problems caused by these factors include rescheduling, repartitioning, speculative execution [1][7][14] etc. A large number of studies identified the problem of skewed jobs in MapReduce which have been reported in Section 5.

Considering unevenness in the data, it is observed that skew arises in reduce phase when few keys are associated with large cluster of values, while others with a very small cluster. When dealing with large volumes of data, the huge amount of intermediate data generated as map outputs are typically moved among nodes of the cluster which requires writing intermediate data locally to disk, reading it later for shuffling and finally distributing it to reduce nodes. This causes significant overheads in network transfers which usually effects overall job execution. Moreover, reducers processing keys with large cluster of values may take significantly longer to finish as compared to other reducers. There are few Hadoop-based systems which follow repartitioning approach, where the long running task is sub divided into smaller partitions and those partitions are assigned to idle nodes in the cluster or to the nodes which finish early [4][44]. But this repartitioning approach entails extra overheads in partitioning tasks and re-distributing them. The application master has to keep track on the statistics of nodes where sub-partitions are assigned.

In this paper, we proposed an approach to address record size skew by working on intermediate data. The main objective is to provide a shuffle service for shuffling and preprocessing intermediate data to reduce the size of *<key, list of values>* before it reaches final reduce node. Intermediate keys are processed by multiple nodes so that when it reaches final reduce node, reduce side skew is unlikely to occur. This results in reducing large amount of data to be shuffled and hence also reducing shuffle delays. Our approach is contrary to re-partitioning, where the keys with larger cluster of values are repartitioned into smaller clusters. The idea is similar to FP-Hadoop [12][30] which introduced a new phase *Intermediate-Reduce (IR)* in MapReduce model. It creates IR fragments and IR splits, where set of intermediate values are processed by intermediate reducers and then by final reducers (*in the Final Reduce phase)*. In FP-Hadoop, the master keeps the IRF (Intermediate Reduce Fragments) metadata and the Reduce Scheduler schedules the tasks using one of the algorithm – Greedy, Locality-aware or IR Size. The proposed system, introduces a new component *Shuffle Tracker* to predict the size of keys, shuffling data and scheduling it on a node.

The main contributions of this paper are:

- Studying different approaches of skew mitigation
- Studying the importance of handling intermediate data to be shuffled
- Discussing the design details of proposed model where intermediate data is first processed by intermediate nodes to reduce the size of large keys and then by final reducers.
- Discussing the design and working of Application Master for executing a MapReduce job.

Rest of the paper is organized as – Section 2 introduces background details of MapReduce and causes of skew. Section 3 discusses motivation behind this study. Section 4 briefs the current approaches used for skew mitigation in MapReduce applications. Section 5 explains the importance of handling intermediate data. Section 6 discusses the proposed model with implementation details. Section 7 talks about related work in this direction. Finally, Section 8 concludes the paper.

II. Background

This section gives an overview of job execution in MapReduce framework and the main causes of skew in MapReduce applications. This will help in understanding the main difference in map side and reduce side skew.

A. MapReduce Pipeline

In Hadoop [34], job execution is carried out by two types of nodes- job tracker and task tracker. Job tracker is responsible for assigning tasks to task trackers and monitoring overall job execution whereas task trackers are the nodes where actually computations are performed. Each task tracker executes an instance of a map() function [51]. A task tracker can also be configured to execute multiple child JVMs (Java Virtual Machine) in multiple slots which depends on number of cores in the processor. When a client submits a MapReduce job, the job tracker manages it and creates number of map tasks per input split. An input split is a logical division of data which corresponds to a block on HDFS [29]. The job tracker keeps the complete information regarding these input splits, their location on HDFS and their size. Map tasks process these input splits in the order of their sizes so that the largest one gets processed first to reduce overall runtime.

When a task is assigned to a task tracker, it reads its input split and starts processing its records one by one. These records are read in form of <key, value> pairs. The map() function produces intermediate results as <key, value> pairs which are stored in a buffer area (configurable parameter), specially allocated for it. Reduce phase starts when all map tasks finish. All reducers are assigned with a different partition to work on values belonging to similar keys. This is called shuffling. The reduce tasks first apply merge sort algorithm on all the input it receives and then processes it further to generate final output which is written on HDFS. Once all reduce tasks are over, the client who submitted the job is informed by the job tracker (master).

There are situations when few reducers are overburden as they are assigned with more values as some reducers are assigned less than average number of values. In this case, data skew is likely to occur. Next section briefly explains various types of skew with their causes.

```
Input – record in form of (k,v)
Output – List of intermediate (k1,v1) pairs
    1.  Map (key, value)
    2.      For all key є set
    3.        do
    4.        Emit (k1,v1)
    5.      End for
Input – Intermediate (k1,list (v1)) with same keys
Output – List of final (k2,v2) pairs
    1.  Reduce (k1, list (v1))
    2.      For all v1 in list
    3.        do
    4.        Process v1
    5.        Emit (k2,v2)
    6.    End for
```

Fig 1: MapReduce Algorithm

B. *Causes of Skew*

Table 1: Causes of skew and their major factors

Stragglers	When some machines run tasks slowly relative to other machines due to hardware malfunctioning. Such problems are resolved by rescheduling the slow tasks on some other idle nodes of the cluster.
Map Side Skew	
Expensive Records	***Major factor*** – Varying record size. Some MR applications are CPU bound and the records may take disproportionate time to process. Map nodes processing such expensive records may suffer from skew, consuming significantly more time than the average run time of remaining map nodes.
Heterogeneous Maps	***Major factor*** - Different computing powers of heterogeneous machines in the cluster. Some map tasks finish too early and some tasks take too long to complete.
Non homomorphic Maps	***Major factor*** – Different runtime of map tasks (in case, where runtime of CPU intensive algorithms depend directly on input data).
Reduce Side Skew	
Partitioning Skew	***Major factor*** - choice of poor partition function to partition the intermediate data. Most of the Hadoop jobs use hash partitioner to evenly distribute the data among reduce nodes. Still skew may occur if few keys contain large amount of values. Few studies show the use of custom defined functions, range partitioner or radix partitioner as the good choice to distribute data.
Record Size Skew	***Major factor*** - Applications generating few keys with a large cluster of values and few keys with small cluster. Solution - Split the larger cluster into smaller ones and redistributes data among nodes. But it imposes extra overhead on task scheduler for task migration.
Computational Skew	***Major factor*** - Applications where the runtime depends on the properties of data values instead of data volume. In such cases, some tasks take longer to process the data assigned to them.

III. Motivation

A. *Effects of Skew – A motivational Example*

WordCount is a very popular and common application in Hadoop which reads an input file and returns the frequency of each word in that file. The map phase reads all words and generates intermediate <key,value> pairs which are stored on local disks of mappers. In some cases, for job optimization combiners are used for local aggregations where each mapper produces local aggregated list of words with their frequencies. These intermediate results are then partitioned on the basis of similar key groups and are passed to reducers for final results. Now in some cases skew may occur at reducer side where few reducers get more data to process as few key groups are assigned more values. To balance the load among reduce nodes, most of the existing algorithms wait until all the map tasks finish before launching reduce tasks. Fig 2 describes the runtime of map and shuffle phase of WordCount application in [10]. It is clear from the figure that the shuffle phase consumes much longer to finish as compared to small map tasks and reduce phase.

A large number of studies in this direction reveal that partition skew has gained much attention and several solutions have been proposed for the same. Only few studies addressed the problem of long delays in shuffle phase by running some online algorithms [10][16][27]. Therefore we put our efforts to propose a solution where shuffle phase starts in parallel to map-reduce phases and does not wait till all the map tasks finish.

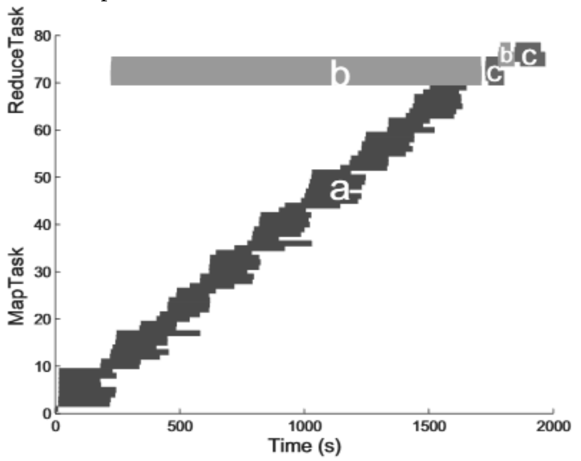

Fig 2: Execution of WordCount in [10]. Region a, b and c represents - actual map, shuffle and reduce execution times respectively. Area stuck between b and c indicates - sorting time

B. *Research Gap*

When dealing with large volumes of data, the huge amount of intermediate data generated as map outputs are typically moved among nodes of the cluster which requires writing intermediate data locally to disk, reading it later for shuffling and finally distributing it to reduce nodes. This causes significant overheads in network transfers which usually effects overall job execution.

Moreover, we know that Hadoop parallelizes job execution by launching multiple map and reduce tasks concurrently. But this parallelism is not fully exploited due to the tight coupling between map, shuffle and reduce tasks due to which MapReduce jobs suffer significant delays in execution affecting overall job performance. Also during partitioning, there are cases where few keys are assigned large number of values and few keys with lesser values. This uneven partition sizes cause skew in reduce phase. Surprisingly, the poor impact of shuffling and partitioning have not gained attention in the literature and motivated us to propose better solution for this problem.

IV. CURRENT APPROACHES FOR SKEW MITIGATION

A. Preprocessing and Sampling

In this approach, the sampling algorithm accurately approximates the distribution of intermediate keys to reducers. The algorithm samples a small fraction of the map output and partitions the data accordingly. It prioritizes the execution of sampled data over normal reduce tasks. Whenever it finds any idle node, it assigns the sampling data to it. Because it launches a small task, it finishes quite early in the system without delaying remaining reduce tasks. It takes into consideration the heterogeneity of the computing resources while balancing the load among the reduce tasks. Sampling approach is also used in [10][13][14].

B. Speculative Execution

When some machines run tasks slowly relative to other machines due to hardware malfunctioning, speculative execution is performed where these victim tasks are rescheduled on some fast machines [1][7][14]. Such slow tasks are often called stragglers and a number of studies proposed different solutions to overcome the delay caused by stragglers. Some studies used task scheduler to collect the information regarding the jobs which are taking longer to complete so that these tasks can be dynamically assigned to some other machines.

C. Custom Partitioning and Re-partitioning

LEEN [9] addresses partitioning skew by partitioning all intermediate keys with respect to their frequencies and fair distribution of reducers' input. It uses a heuristic technique to identify the best node suitable for partitioning any specific key. It guarantees fairness in data distribution under large key frequencies variations along with high performance. Few studies [4][44] believe in addressing skew by continuous monitoring the runtime statistics of an application and partitioning data on-the-fly. By means of computing local statistics during map phase and aggregating them to produce global statistics, the global data distribution is approximated. Based on this distribution, the map output is directed to reducers in a way that achieves improved load balancing. Partitioning functions used in these studies include hash partitioning; range partitioning, radix and round-robin partitioning.

D. Batching at Reducer Side

Some studies believe in improving the performance of MapReduce by reducing the number of disk accesses using batching at reduce side. In this approach, instead of writing map output to local files, data is shuffled directly and then written to a single file at reduce side, resulting in one file per reduce task. This method significantly reduces disk seeks at the reducer side. [15] and [41] used batching and sampling methods to address data skew.

V. SIGNIFICANCE OF HANDLING INTERMEDIATE DATA

The intermediate data generated by map tasks are stored locally in a buffer area. Size of this buffer is a configurable parameter which can be set via *mapreduce.task.io.sort.mb* property. When the size of this buffer reaches almost 80% of its available size, the contents are transferred to a file (spill) on the disk. This process is called spilling. Spilling happens at least once, when the mapper finished, because the output of the mapper should be sorted and saved to the disk for reducer processes to read it. The spill file contains partitions where similar keys are stored together. A combiner can be used to perform local aggregations on map side before spilling is done. The map task is said to be complete when all spill files are merged into a single output file and the (application) master is informed. Now, the reducer fetches its partition from each of the mappers and merges the partitions, called shuffling. After a reducer has obtained its complete set of inputs, a user defined function is applied on it to generate final output to be written on HDFS. This complete process is shown in figure 3.

It has been observed [41] that the cost of handling intermediate data grows when number of keys in intermediate <k,v> data set grows with variable size values. The presence of skew in map outputs affects number of reduce waves and their completion time. With large number of reduce waves, number of retrievals from mappers local disk also increases, which in turn increases number of disk seeks. So an increase in the variable size of intermediate data, degrades Hadoop performance non-linearly.

This paper presents a model where regardless of skew in intermediate data, the keys are processed by multiple reducers in parallel. The key objective is to utilize the computing power of all reduce nodes equally and making reduce nodes work on an average of intermediate data in parallel. This model would also work in the worst case, where almost 90% of values belong to the similar key. The design details of the model are given in the next section.

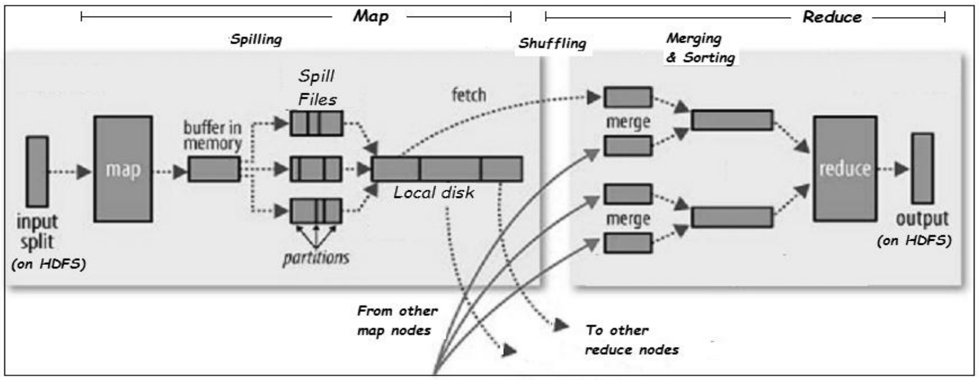

Fig 3: MapReduce Pipeline

VI. DESIGN DETAILS

The proposed model provides a shuffle service for shuffling and preprocessing intermediate data to reduce the size of *<key, list of values>* before it reaches final reduce node. Intermediate keys are processed by multiple nodes so that when it reaches final reduce node, reduce side skew is unlikely to occur. It starts scheduling reduce tasks as soon as map tasks begin producing their output. The main difference between the current approach used by Hadoop for scheduling map-reduce tasks and proposed model is that in the proposed model the keys with large cluster of values are not processed by a single reduce node making it overloaded. Instead, a combiner based approach is followed up where similar keys are processed by multiple reduce nodes and the partial results are forwarded to the final reducer for aggregation.

A. Design overview

The basic flow of <key,value> pairs in proposed model is shown in Figure 4. The figure shows the required components of the proposed model. It includes mapper nodes, an application master and reducer nodes. The mapper nodes process map tasks on their input split. The map outputs are stored in the buffer, which is spilled into different partitions in the spill files available on local disks. The ApplicationMaster is a *framework-specific library* which executes a single application. It is responsible for negotiating resource containers from the ResourceManager, tracking and monitoring the status of the running application. With the help of Application master, YARN shares the metadata of all running applications with the cluster. When map tasks produce intermediate data, it is processed and aggregated by reducer nodes.

In the proposed model, the application master is configured with two new components *Shuffle Tracker* and *Merger*. *Shuffle Tracker* first collects map outputs and predicts the size of each key it receives. If the size of the key is larger than the size of partition that can be handled by reduce nodes, it partitions it into smaller fragments and schedules the fragments on multiple nodes. Then, *Merger* collects the partial results from these nodes. It is responsible for merging partial results belonging to same key into single partition. After merging, *Shuflle Tracker* is notified. Merger continuously keep reporting the partial results it receives from the nodes. When Shuffle Tracker receives multiple partitions for same keys, it again re-computes the size of the key. The cycles continue to iterate till the size of the key is reduced to the required partition size so that it can be efficiently processed by final reduce nodes. The final reducers process these partial (aggregated) results and produces a single output file which is written on HDFS.

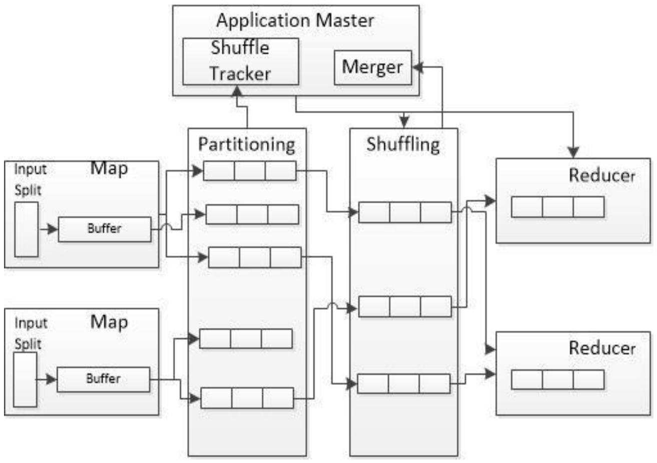

Fig 4: Architecture of proposed MapReduce Execution Strategy

B. Partition Placement

The proposed model works on the objective to balance the distribution of intermediate data among reduce nodes. Unlike other distributed systems, where an optimal partition placement plan is developed, this model does not need to plan partition placement strategy. Instead, the partition is scheduled on a free node as soon as mappers start writing their outputs on their local disks. All nodes get equal workload because partitions belonging to the same key are not scheduled on the same node. Other hadoop based systems such as iShuffle [11] follow heuristic approach – *largest partition* and *least workload* for partition placement because best placement is a NP-hard problem.

C. Job Execution Model

In MapReduce programming model, when map tasks are assigned to mapper nodes, the execution of task begins by reading the input split defined by RecordReader. The map function is applied to the input split and intermediate data is generated. The set of intermediate <key,value> pairs are written to buffer. Before spilling the contents of the buffer the intermediate keys are split into partitions and the values belonging to same key are placed in sorted fashion in the same partition. The combiner aggregates the values within the partition to reduce its size. Before writing the intermediate output file to local disk, all partitions (or spills) are merged. In the proposed model, these partitions are not merged together. Instead they are read by *Shuffle Tracker* which is responsible for predicting the size of each key and scheduling it on a reduce node. The sizes of keys are determined by merging the partitions of same keys arriving from different mapper nodes.

The intermediate <key,value> pairs are dynamically grouped together into intermediate splits, as

(k,v) → list (k1,v1)   // processed by each mapper
(k1, list (v1)) → (k2, list (v2))   // by intermediate redcuers
(k2, list (v2)) → list (k3,v3)   // final output by reducers

In the second stage, where intermediate keys are processed, multiple intermediate reducers may work on same keys for faster computations and to avoid reduce side skew.

D. Shuffle and Reduce Task Scheduling

The model disconnects the shuffling and reduce tasks and these tasks run on different nodes in the cluster. Shuffling is done by *Shuffle Tracker*, component running on Application Master whereas reduce tasks run on reduce nodes. The sequence of flow is shown in the figure. It consists of three main steps. In the first step, the S*huffle Tracker* collects intermediate results produced by mappers from their local files stored on disks. During this step the size of spill files and size of each partition within a spill is determined and the statistics are stored on Application Master. In the second step, *Shuffle Tracker* creates splits from different partitions of belonging to same key and launches them on intermediate reducer nodes. For this purpose multiple nodes can be used to work on same keys in parallel. The size of the newly created partitions should be either 64MB or 128MB (depending upon the configuration of the block size). This is why every reduce node gets roughly equal amount of data to process. Finally in the third step, the partial results from all the intermediate nodes are collected and sorted by key. A *Merger* receives the partial results which are merged together (belonging to the same key). Then it is forwarded to final reduce nodes for final output and being written to HDFS.

VII. RELATED WORK

As discussed in section 4, different studies put their efforts in different directions to address skew mitigation. [1][20] used speculative execution where the tasks on the slow machine are launched at fast machines. Dryad [44] uses speculative execution to run fully distributed and scalable applications on a cluster of machines. [19] uses LATE scheduler to estimate remaining time of jobs on slow machines so that they can be rescheduled on some other machines. Q. Chen et al [25] proposed a scalable model for data skew. This model uses process bandwidth and progress rate in a current phase to decide slow tasks and it calculates tasks remaining time and makes predictions about process speed using EWMA (Exponentially Weighted Moving Average).

[10] and [14] uses a sampling approach which accurately approximates the distribution of intermediate keys to reducers. It prioritizes the execution of sampled data over normal reduce tasks. It takes into consideration the heterogeneity of the computing resources while balancing the load among the reduce tasks appropriately. MTCRS [13]**,** a Minimum Transmission Cost Reduce Task Scheduler, uses a mathematical model based on ARS sampling method to handle the problem of data locality and partitioning skew. It accepts the waiting time and transmission cost of each reduce task to predict where to launch any specific reduce task for a given partition.

[23] [24] proposed a dynamic MapReduce system with situation aware mappers which constantly examines the execution of all map tasks and dynamically splits the map input data. [4] used two load balancing algorithms- fine partitioning and dynamic fragmentation, to deal with complex reduce jobs and skewed data. These algorithms are based on cost model which evaluates cost of reduce tasks in distributed environment and helps in uniform load balancing of highly skewed tasks.

SkewTune [5] system mitigates skew in both map side and reduce side by repartitioning the long jobs and allocating to idle nodes freed up by shorter jobs. It runs three algorithms – *detect* to identify the longest time by estimating the remaining time of jobs, *scan* to collect the repartitioning information, and *plan* to repartition the job and assign it to available nodes. EDSHA [36] is a skew handling approach which works as an intermediate task between map and reduce tasks. It identifies the high performance node in the cluster by examining the efficiency and execution times of nodes and assigns huge data to it dynamically. ImKP [50] based MapReduce framework also deals with reduce Side skew by using group based ranking technique to aggregate intermediate data.

CHISEL [37], uses two different approaches to deal with skewed data. It uses skew detection and mitigation approach for computational skew that occurs at map side, and skew avoidance approach for reduce side skew. SkewReduce [3] system employs a static optimizer which requires user to provide cost functions to partition the input data to avoid computational skew. Partitioning among reducers depends on this cost function to optimize data distribution. iShuffle [11] proposed a novel shuffle-on-write operation which decouples shuffle phase from reduce tasks because the coupling of these two phases delays job completion and desecrates the parallelism. [9][45] address partitioning skew, where all intermediate keys are dynamically partitioned with respect to their frequencies. It is done by continuous monitoring the runtime statistics of a running application. Partitioning skew is also addressed in [8][15][27][32] and [33].

OPTIMA [28], mitigates partition skew by predicting the distribution of workload of all reduce tasks. It exploits a technique called *deviation detection* to examine the overloaded tasks and thus reduces their execution time by making suitable resource allocation for these overloaded tasks. DREAMS [31] also uses dynamic resource allocation to handle skew at reduce side. In case of partition skew, rather than repartitioning the data, it allocates few more resources to the victim node to let it finish faster. [6] addressed the problem of record size skew and discussed how such skews can be handled in an application specific manner. [16] [17] [18] describe techniques to handle specific types of record size skew. [7] uses greedy bin-packing heuristics to deal with partition skew.

VIII. CONCLUSION & FUTURE WORK

The proposed model works for applications where there is no interdependency between values of same keys. For example, the results of aggregate functions like sum, count etc can be obtained correctly by partitioning the values into smaller partitions and merging the final results later on. But this is not the case with average function. In such a case, it is required that all the values associated with the same keys should be processed at same node. As a future work, we plan to extend this model where a variety of applications experiencing shuffle skew and reduce side skew can be handled efficiently and can be processed in a cost optimized way. To make this model working, few parameters need to be included in configuration files supported by Hadoop. The system must provide transparent services to user where user can tune parameters according to their cluster deployment and type of their application.

References

[1]. J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, OSDI 2004.

[2]. Y. Kwon et al, “A study of skew in mapreduce applications”, 5th Open Cirrus Summit, 2011.

[3]. Y.Kwon et al, “Skew-resistant parallel processing of feature-extracting scientific user-defined functions”, in Proceedings of ACM Symposium on Cloud Computing, 2010, pp. 75- 86.

[4]. B. Gufler et al, “Handing Data Skew in MapReduce, in Proceedings of 1st International Conference on Cloud Computing and Services Science”, 2011, pp. 574- 583.

[5]. Y. Kwon et al, “SkewTune: Mitigating skew in MapReduce applications”, ACM 2012, SIGMOD 2012 USA.

[6]. J. Lin, “The Curse of Zipf and Limits to Parallelization: A Look at the stragglers problem in Map Reduce”, July 2009, USA.

[7]. J. Rosen and B. Zhao, “Fine Grained Micro Tasks for MapReduce Skew Handling”, 2012.

[8]. M. Hanif and C. Lee, “An efficient key partitioning scheme for heterogeneous MapReduce clusters”, ICACT 2016, ISBN 978-89-968650-7-0.

[9]. S. Ibrahim et al, “Handling partitioning skew in MapReduce using LEEN”, Springer 2013.

[10]. Y. Le et al, “Online Load Balancing for MapReduce with skewed Data Input”, IEEE Transactions, 2014.

[11]. Y. Guo et al, “iShuffle: Improving Hadoop Performance with shuffle-on-write”, 10th International Conference on Autonomic Computing 2013.

[12]. R. Akbarinia et al, “An efficient solution for processing skewed MapReduce Jobs”, Globe'2015: 8th International Conference on Data Management in Cloud, Grid and P2P Systems, Sep 2015, Spain.

[13]. X. Tang et al, “A Reduce Task Scheduler for MapReduce with minimum transmission cost based on sampling evaluation”, IJDTA, Vol 8, No 1 (2015), pp 1-10.

[14]. Qi Chen et al, “LIBRA: Light Weight data skew mitigation in Map Reduce”, IEEE Transactions on Parallel & Distributed Systems, 2015, Vol 26, Issue 9.

[15]. A. Rasmussen et al. “Themis: an i/o-efficient MapReduce”. In Proceedings of the Third ACM Symposium on Cloud Computing, page 13. ACM, 2012.

[16]. S. Ibrahim et al, “Leen: Locality/fairness-aware key partitioning for MapReduce in the cloud”. In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, pages 17–24.IEEE, 2010.

[17]. M. Hammoud et al, “Center-of-gravity reduce task scheduling to lower mapreduce network traffic” in 2012 IEEE 5th International Conference on, pages 49–58. IEEE, 2012.

[18]. M. Hammoud and M.F. Sakr, “Locality-aware reduce task scheduling for mapreduce”. in 2011 IEEE Third International Conference on, pages 570–576. IEEE, 2011.

[19]. M. Zaharia et al, “Improving mapreduce performance in heterogeneous environments”. In Proceedings of the 8th USENIX conference on Operating systems design and implementation, pages 29–42, 2008.

[20]. G. Ananthanarayanan et al, “Reining in the outliers in map-reduce clusters using Mantri”. In Proceedings of the 9th USENIX conference, OSDI’10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[21]. Zaharia et al, “Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling”. In Proc. of the ACM European Conference on Computer Systems (EuroSys) 2010).

[22]. M.C. Schatz, “Cloudburst: highly sensitive read mapping with MapReduce”. Bioinformatics, 25(11):1363–1369, 2009.
[23]. R. Vernica et al, “Adpative Map Reduce using Situation aware Mappers”, ACM 978-1-4503-0790-1, EDBT March 26-30, 2012.
[24]. R. Vernica et al, “Efficient parallel set-similarity joins using map reduce”, in Proceedings of SIGMOD Conf, pages 495-506, 2010.
[25]. Q. Chen, C. Liu and Z. Xiao, “Improving MapReduce Performance Using Smart Speculative Execution Strategy”, IEEETransactions on Computers (TC)63(4), 2014.
[26]. Y. Kwon et al, “Managing Skew in Hadoop”, IEEE Computer Society Technical Committee on Data Engineering 2013.
[27]. Y. Gao et al, “Handling data skew in MapReduce cluster by using partitioning tuning”, Journal of Health engineering, Volume (2017), 2017.
[28]. Liu et al, “OPTIMA: on-line partitioning skew mitigation for MapReduce with resource adjustment”, Journal of Network and Systems Management, vol. 24, no. 4, pp. 859–883, 2016.
[29]. Apache Software Foundation, “Hadoop Distributed File System: Architecture and Design”, 2007.
[30]. R. Akbarinia et al, “FP-Hadoop: Efficient Processing of Skewed MapReduce jobs”, Information Systems, Elsevier, 2016, 60, pp-69-84.
[31]. Z.Liu et al, “Dynamic Resource Allocation for MapReduce with Partitioning Skew”, IEEE Transactions on Computers, Issue No. 11 - Nov. (2016 vol. 65)ISSN: 0018-9340, pp: 3304-3317.
[32]. S. R. Ramakrishnan et al, “Balancing reducer skew in MapReduce workloads using progressive sampling”, in Proceedings of the Third ACM Symposium on Cloud Computing, pp. 16–28, ACM 2012.
[33]. J. Berlinska and M. drozdowski, “Mitigating Partitioning Skew in MapReduce Computations”, MISTA 2013.
[34]. J. Dittrich and J.-A. Quian_e-Ruiz. “Efficient Big Data processing in Hadoop MapReduce”. Proceedings of the VLDB Endowment (PVLDB), 5(12):2014{2015, 2012C. Doulkeridis, K. Norvaget, A Survey of Large Analytical Query Processing in Map-Reduce, the VLDB Journal, 2013.
[35]. B. Arputhamary et al, “EDSHA: An Efficient Data Skew Handling Approach for MapReduce Model using Time Series Data”, IJCTA 9(27) 2016, pp 423-430.
[36]. P. Dhawalia et al, “Chisel++: Handling partitioning skew in MapReduce framework using efficient range partitioning technique”, DIDIC 2014, pp 21-28.
[37]. H. Chang et al, “Scheduling in MapReduce-like systems for fast competition time”, In proceedings of IEEE INFOCOM, China, 2011.
[38]. J.Tan et al, “Coupling task progress for MapReduce resource aware scheduling”, in Proceedings of IEEE INFOCOM, 2013.
[39]. F. Chen et al, “Joint scheduling of processing and shuffle phases in MapReduce systems”, in Proceedings of IEEE INFOCOM, 2012.
[40]. Y. Yuan et al, “On interference-aware provisioning for cloud based big data processing”, in Proceedings of ACM/IEEE IWQoS, June 2013.
[41]. S. Rao et al, “Sailfish- A framework for large scale data processing”, ACM Symposium on Cloud computing, SOCC 2012, UA, 2012.
[42]. Y. Liang et al, “Variable sized map and locality aware reduce on public-resource grids”, Future Gen Computing Systems, 27(6) : 843-849, June 2011.
[43]. M. Isard et al, “Dryad: Distributed Data-parallel programs for sequential building blocks”, In proceedings of EuroSys Conf, 2007.
[44]. K. Devine et al, “Partitioning and Load balancing for emerging parallel applications and architectures”, Chapter 6, Parallel Processing for Scientific Computing, 2006.
[45]. T. Y. Chen et al, “LaSA: A locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment”, Collaboration Technologies and Systems (CTS), 2013, pp. 342-346.
[46]. S. Seo et al, “HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment”, IEEE International Conference, New Orleans, LA, pp. 1-8.
[47]. N.K. Seera and S. Taruna, “Analyzing cost parameters affecting Map Reduce application performance”, I. J. Computer Science and Information Technology, 2016, 8, 50-58.
[48]. N. Kaur and S. Taruna, “Efficient data layouts for cost optimized map reduce operations”, IEEEXplore, 2015, 600-604.
[49]. M. Kaur and G. Dhaliwal, “Performance comparison of MapReduce and Apache Spark on Hadoop for Big Data Analysis”, International Journal of Computer Sciences and Engeering, Vol 3 (11), PP- 66 – 69, Nov 2015.
[50]. M. Dhivya et al, “Hadoop MapReduce Online in Big Figure Analytics”, International Journal of Computer Sciences and Engeering, Vol 2 (9), PP- 100 – 104, Sep 2014.
[51]. J. Rajesh Khanna, “An Enormous Inspection of MapReduce”, International Journal of Scientific Research in Computer Science, Engineering and Information technology, Vol 2, Issue 6, IISN 2
[52]. Ouyang, X, Zhou, H, Clement, et al.,”Mitigate Data Skew Caused Stragglers through ImKP Partition in MapReduce”, Proceedings. 36th IEEE International Performance Computing and Communications Conference (IPCCC), 10-12 Dec 2017, San Diego, California, USA. IEEE.

Authors Profile

N.K. Seera, is a research scholar in Banasthali vidyapeeth. She is carrying out her research work in Data Processing using MapReduce frmaework. Her areas of intererst are Databases, Distributed computing, Big Data etc. She has published various research papers in National and International Journals. She is also a memebr of CSI and other autonomous governing bodies.

Dr S.Taruna, is working as an Associate Professor in Institute of Engineering and Technology (IET) at JK Lakshmipat University. She has 20 years of teaching, research, and administrative experience. She has also worked with Banasthali University for 12 years and CIStems Software Ltd for 8 years. She did her PhD from Banasthali University and currently supervising PhD candidates working in the domain of Communication Network, Data Mining and Cloud Network. She has several publications to her credit and has presented research papers at National and International conferences and journals . She is Reviewer and Committee Member of various International Journals and Conferences. Her area of interest for Research, Training and Consultancy includes Adhoc & Sensor Network, Data Mining , Information Retrieval and Cloud Network.