© 2016 IJSRSET | Volume 2 | Issue 3 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

A Survey on Hadoop Storage Issues

Reetesh Rai, Shravan Kumar

LNCT, Jabalpur, Madhya Pradesh, India

ABSTRACT

Hadoop is an open source implementation of Google's MapReduce framework. MapReduce is the heart of the apache's Hadoop. The file system which is used by the Hadoop for storing the files is known as Hadoop distributed file system (HDFS) which is an open source implementation of the google file system (GFS). Hadoop allows the parallel processing of the large data sets by splitting the larger data set into smaller partitions and each partition is fed to the separate task in the data node by the job tracker. The data node is the node where the data actually resides. The task tracker resides on the data node and it runs the tasks and also reports the status of the tasks to the job tracker. In a MapReduce, the slowest running task decides the job completion time. If the task is slower, it delays the progress of the entire job. This slowest running task is known as the straggler. There can be many reasons for the straggler to occur. One of the reasons is the data skew. This paper reviews the different types of the data skew, where in MapReduce data skew can occur and what is the measure taken to overcome these problems.

Keywords: Mapreduce, HDFS, Straggler, Data Skew

I. INTRODUCTION

Apache's Hadoop is an open source implementation of the Google's Map/reduce. World Wide Web has proved to be the efficient platform for developing applications which are data intensive in nature. As huge volumes of the data are generated day by day, more number of popular applications becomes data-intensive in nature. This makes the data mining and web indexing applications to access the largely expanding data sets ranging from gigabytes to several tear or peat bytes. By the help of the MapReduce model, the google processes the 20 peat bytes of the data per day in a parallel Fashion. The performance and the scalability of the MapReduce are increased because in a MapReduce model the large data set is split into many smaller partitions and the job is then partitioned into numerous smaller tasks. All these smaller tasks run on the multiple different nodes in a cluster. For example, YAHOO makes use of a cluster consisting of the 10,000 nodes to process.

Hundreds of tear bytes of data generated. Facebook generates about 15 TB of data per day and it is also processed by the Hadoop. Websites like, the amazon also makes use of the Hadoop to process huge volumes of the data on daily basis. Scientific applications like the seismic simulation and NLP also makes use of the Hadoop to the fullest. In a MapReduce, data locality determines the MapReduce performance. To prove to be the load balancer, the Hadoop distributes the data across the nodes in a cluster based on available disk spaces. In case of homogenous environment, the computing and disk capacity are identical in all the nodes. However, in case of heterogeneous environment, the nodes are not identical. Nodes differ in the computing capacity and disk capacity. So in this case some of the tasks may take unusually more time to complete than the tasks on the high performing nodes. These high performance nodes can complete processing data faster than low performing nodes. This causes the imbalance in the processing of data and thus the entire job is delayed by the slowest running task. This slowest running task which delays the execution of the entire job is known as the straggler [12]. There can be various reasons for a straggler to occur. Straggler can occur because of many external and internal factors. In case of heterogeneous environment, where the straggler occurs just because of the difference in the disk and the computing capacity, techniques like speculative execution can be used to overcome the issues. However, in case of the homogenous

environments the issue of the straggler cannot be resolved by means of the techniques. One of the reasons for a straggler in the homogenous environment can be a data skew which cannot be resolved by simply transferring the task to the other machine. There is an increased demand for the user-defined operations (UDO) arising from the complex and advanced analytics of the data.

II. METHODS AND MATERIAL

1. Hadoop Distributed File System

HDFS is an open source implementation of the google file system (GFS). HDFS is designed to store large files and all these files are stored on the clusters of commodity hardware [9]. The files that are meant to be Stored in the HDFS are of hundreds of megabytes, gigabytes or terabytes in size. It can be a petabyte of a data as well. HDFS does not require any expensive or highly reliable hardware, but it just requires the commodity hardware which is the hardware that is commonly available.

1.1 HDFS Architecture

The architecture of the HDFS consists of the specified nodes which are [6], [7], and [9] Name Node and Data Node and various daemon processes like Job tracker and Task tracker. The Name Node is a node which does not store any actual data but it stores the Meta data information about the data like number of the blocks, on which rack and in which Data Node the particular data block is stored in. It also stores the information about the file system directory tree. The Data Node is the node which stores the actual data. The Name Node is the single point of failure and it is also considered as the center piece of the HDFS. The Name Node is the master node. Whenever any client application wants to add/delete/move/copy any data from a file in HDFS, it has to contact the Name Node directly. Name Node has a disadvantage that it is a single point of failure and if it fails it is as if whole system fails. However, the newer versions of Hadoop have a secondary Name Node on the separate machine. This keeps track of images of the primary Name Node and helps in case of a failure. The data node communicates with the Name Node by sending the heart beat messages after every 3 seconds. If the Data Node does not communicate with the Name Node for a specified amount of time, it is considered to be dead and the replication of the data blocks on that data node is performed on the other working Data Node. Job tracker is a daemon process whose task is to submit and track the MapReduce jobs in Hadoop. It submits the job to the job tracker. Task tracker is a daemon process that runs the tasks and also reports the status of the task to the job tracker. Thus the task tracker performs the tasks of the map and reduces. Task tracker runs on the Data Node. The concept of blocks is also in the HDFS. The size of the blocks in the HDFS is a larger unit of 64MB by default. The larger block size in the HDFS is used to minimize the costs of seeks. Thus it takes more time to transfer the data from the disk than the time to seek to the start of the block. Consider the transfer rate of the 100MB/s and a seek time of 10ms. To make the seek time 1% of the transfer time, the block size should be around 100MB. Thus the default size of the block is 64 MB [3] but in some cases the block size is kept as 128.

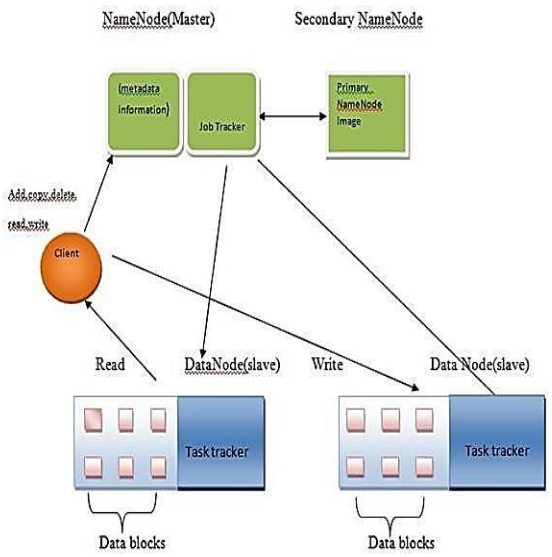

Figure 1. Block diagram of HDFS architecture

1.2 Main features of HDFS

- Quick recovery: The architecture of the HDFS is built in such a way that there is a quick and automatic recovery when a fault is detected. HDFS consists of the large number of nodes and each node may store only a part of the file. Some nodes are non-functional and carry out the tasks of fault detection and reporting.
- Access to data sets: In HDFS the main focus is on the high throughput of the data access. Applications that are developed to run on HDFS are not general purpose applications. They do not run on general purpose file system. Thus they

need streaming type of access to the data sets stored in HDFS. HDFS provides a streaming type of data access to the applications designed to run on it.

- Support for large files: The size of the file in HDFS ranges from the gigabytes to terabytes. HDFS provides a full support to such large file. It also provides high bandwidth to access such large files.
- Portability: HDFS has been designed in such a way that it can be used in any platform. Thus we can make use of platform of our choice.
- Moving just the computation than moving the data: When the size of the file is huge, it is efficient to move the computation near the data area other than moving data to the computation area. This saves a lot of bandwidth and also minimizes the network congestion. HDFS provides a full support to the applications to move the computation closer the area where data is located. This simplifies the operations in HDFS.

1.3 Benefits of block level abstraction in HDFS

- We can have a larger file in the network than any single disk and the blocks of the file can be stored in any disks in the cluster.
- Replication is done on these blocks thus the issues of fault tolerance and availability are easily met. Here each block is replicated at least 3 times on the separate physical machine thus providing protection against machine failure and data corruption.
- The storage management is simplified by using the block level abstraction. By this way, it becomes very easy to calculate how much number of blocks can be stored on a particular disk and which block is on which disk.

2. Mapreduce Programming Model

It is considered as the heart of the Hadoop. MapReduce is an inherently parallel data processing programming model. There are various frameworks like the google MapReduce, Microsoft Dryad and Apache's Hadoop which support MapReduce programming model [12]. Among all these, the Apache's Hadoop is an open source. The MapReduce programs run by the Hadoop can be written in various languages like java, python, ruby and c++. All these MapReduce programs are data-intensive in nature and process very large data sets in a parallel fashion.

2.1 Data processing in MapReduce:

The working of MapReduce is based on these two phases. Map phase consists of the Map function and Reduce phase consists of the Reduce function [9], [10]. The first phase is the map phase which takes the raw data such as the text file as the input. The input is divided into the several parts known as splits and each split is fed to the separate map task. The size of the split is same as the size of the data block on the HDFS. Map task transforms the input data into the (key, value) pairs which is also known as the intermediate data [1], [2]. The map output is then fed to a combiner functions which is a user defined function. The combiner function output serves as the input to the reduce tasks. The combiner functions help to dispatch the (key, value) pairs that share the same key to the same partition. The partitions are decided by a default petitioner such as hash partitioned or some user defined petitioners. The locations about these partitions are sent to the Name Node. The Name Node then assigns a reduce tasks to the nodes and also passes the information about these partitions to the hose nodes. Thus the reduce nodes communicates with those partitions and are fed by (key, value) pairs present in those partitions. Thus the reduce task is performed which processes and simplifies the intermediate data. The reducer output is stored Directly on the HDFS while the mapper output is stored on the local file system.

Hadoop provides the data locality optimization which helps to run a map task on the node where the input data resides on the HDFS. The MapReduce always splits the input data for parallel process. Each split is smaller as compared to whole file and thus each split takes less time to get processed. So when we process the large number of splits in parallel, the processing is better load- balanced. Hence, a faster machine will be able to process more splits over the course of the job than a slower machine. The splits must not be too small otherwise we may face an overhead of managing the splits. In most cases, the size of splits is same as size of HDFS block (64MB) [8], [11]. If there is a single reduce task, the output from all the map tasks is fed to that single reduce task. Thus there is no data locality optimization in reduce tasks. Here the map outputs have to be transferred across the network to the node where the reduce task is carried out. To guarantee the reliability, the reduce outputs are stored in HDFS and

are replicated across the nodes with one replica on the local node. BUT when there are multiple reducers, a partitioning function is needed to partition the map outputs into different partitions. Each partition gets the (key, value) pairs with same keys. Thus the number of partitions is equal to the number of different keys.

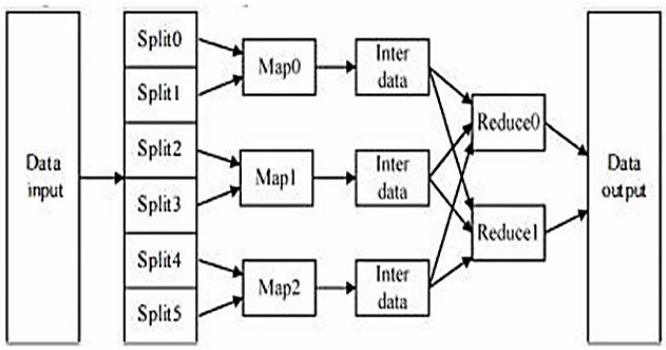

Figure 2. Working process of MapReduce

2.2 Stragglers in MapReduce

As discussed earlier, the straggler is the slowest running task which delays the execution of entire job [12]. Stragglers are caused by various factors.

It is very easy to overcome the straggler caused by the external factors. The commonly used method is called as the speculative execution. If the machine is performing slowly, or if a machine fails or if there is any faulty hardware in the machine, we can overcome it by simply Shifting the workload to some other machine which is performing well.

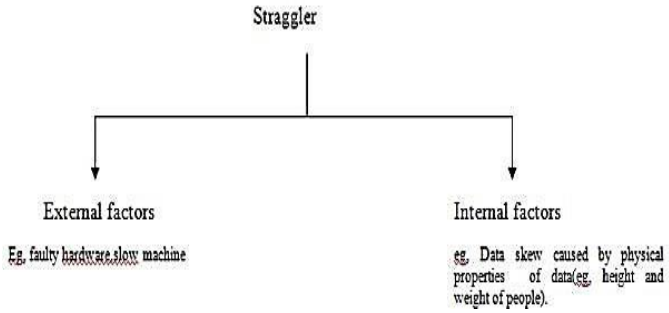

Figure 3. Causes of Straggler

In this way we can easily overcome the straggler issue. However, the speculative execution cannot be used when the straggler is caused by the internal factors of the data like the physical properties of data (height of persons, weight of persons), the speculative execution cannot be used because shifting the work load to other machine does not change the properties of the data and thus cannot overcome the straggler issue. This straggler caused by this issue is known as the data skew. Data skew can occur in both the phases of the MapReduce. When the data skew occurs in the Map phase, it can be easily mitigated by splitting the map tasks. The more complex data, that takes time and is difficult to process, is responsible for the data skew on the Map phase. The data skew on the map phase is rarely observed. The data skew on the reduce side is very difficult to overcome and is a challenging problem. A number of data-intensive applications like the data mining and web indexing applications as well as the scientific data-intensive applications have witnessed the same data skew problems.

2.3 Types of data skew in MapReduce applications

2.3.1 Map side data skew

There are three causes of data skew on the map side [4], [5]:

- Slow performing cup: The map tasks take the data and transform it into the (key, value) pairs. Each map task is given an equal amount of data so the focus here is on the amount of data to be processed and not on the time to process the data. Some of the machines which do not perform well take significantly large amount of time to process data than the one which perform well, thus causing the task to lag behind.
- Complex map tasks: Each map task is assigned a data set of same size. However, some of the map tasks are so complex that they require different Processing and more time than other map tasks. This leads to a data skew on the map side.
- Varying data distribution: The distribution of the input data to the map task may vary significantly. In some cases, the map task depends on the CPU intensive algorithms. Thus the runtime of such algorithms depend directly on the distribution of the input data. This leads to data- skew on the map side.

2.3.2 Reduce Side Data Skew

The various causes of the data skew on the reduce side are [5]:

- Skew caused by partitioning: partitioning is the division of the intermediate data in such a way that the (key, value) pairs with the same key are placed in one partition which is fed into the same reducer. The default partitioned used is the hash- partitioned

or some other user defined partitioned can also be used. However, even after evenly distribution of the data to the reducers by means of the partitioning functions, a reduce-side skew can still occur. Consider a scenario, when the partitioning function distributes the (key, value) pairs perfectly across the reducers, some reducers may still get more data because the (key, value) pairs that are assigned to it contain more values than others. The partitioning logic must not rely on the values computed during the map task otherwise it causes a skew on the reduce side.

- Larger clusters: This type of reduce side skew caused by the complex map task in the map side. The reduce tasks process the data in the form of (key, value) pairs. Some of the reducers may get a single larger cluster and other reducers may get smaller clusters, giving rise to a data skew on the reduce side. In order to overcome this type of data skew, it should be able to split the larger cluster into smaller clusters and then distribute these data clusters evenly across the reducers to avoid the data skew.

2.4 Existing solutions to DataSkew

There is an increased demand for user defined operations (UDO) for advanced analytics of large data sets. MapReduce provides enough supports writing UDO's and using them for massive processing of large data sets. The user just has to write the Map and Reduce functions, API's for writing UDO's are provided by MapReduce. Skew is a known problem that occurs either in the Map phase or in the Reduce phase and it is related to parallel database management systems and adaptive or stream processing systems. One solution to mitigate skew is the implementation of skew-resistant operators. There is a Disadvantage in this approach that it imposes an extra burden on the operator writer. It only applies to operations that satisfy some specific properties. Another technique involves dividing the work into smaller partitions and transferring these partitions to different machines when needed. Such a strategy imposes significant overhead due to either task migration or extra task scheduling.

SkewTune is another technique for handling skew in parallel data processing in MapReduce [13]. SkewTune is designed for MapReduce type programming model. Two properties of the MapReduce model on which the skew Tune relies are:

- MapReduce buffers the output of one operation before it is passed for the next operation.
- MapReduce has an operator decoupling, where each operator processes data independently.

SkewTune helps in mitigating skew and does not impact the fault-tolerance and scalability of MapReduce. Two very common types of skew are mitigated by SkewTune. Skew which is caused by varying or uneven distribution of data to partitions and skew which is caused by some larger data sets that take longer time to process than others.

2.4.1 Some key features of SkewTune are:

- SkewTune is compatible with MapReduce programmers. There is no need to change even a single line of the code.
- SkewTune guarantees that the output of any operation consists of the same number of partitions and also preserves the total ordering of the data in those partitions.
- When a skew arises the SkewTune reduces the processing times by factor of 4 and also adds a very low overhead when there is no skew

2.4.2 Working of SkewTune:

The working principle of the SkewTune [13] depends on re-allocation of the parts of straggler to the slots which are fast working or which have already completed its task. Consider the following figure.

Here, the job completion time of the entire job is decided by the slowest running task (straggler) which is task T2 in this case [13]. First the SkewTune detects the straggler and tries to mitigate it. It re-partitions the task T2 in such a way that it allocates the partitions to every slot available. Once the slot becomes available, it starts running part of the T2 task (straggler) which was allocated to it as shown in the figure B.

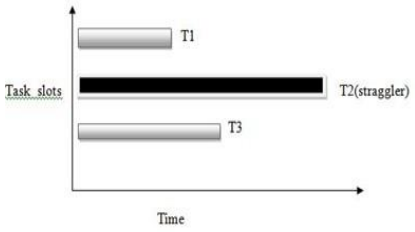

Figure 4. Data Skew caused due to straggler (without skew tune)

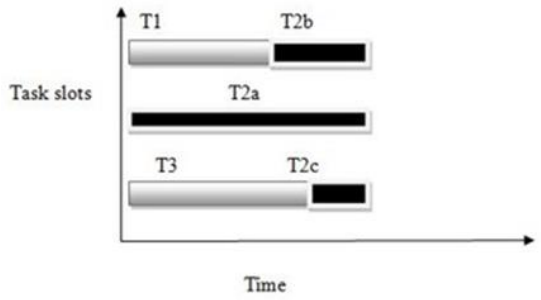

Figure 5. Mitigation of the straggler using SkewTune

Here the task T2 is re-partitioned in to three tasks (T2a, T2b, and T2c). The task T1 completes before every task, so it gets the T2b part of the T2 and Task T3 completes after Task T1 finishes so it gets the T2c part of the task T2. In this way every task finishes at around the same time.

Another already existing technique for mitigating the reduce side skew is known as LIBRA (light weight implementation of the balanced range assignment) [12]. It uses an efficient technique to balance the work load among reduce tasks by splitting the larger data sets. LIBRA can also balance the work load in case of heterogeneous environment with no data skew. In order to solve the data skew problems, LIBRA makes use of the new sampling method which integrates a small percentage of the sample tasks into the normal map tasks. These sampling tasks are given a preference over the normal map tasks and they collect the statistics about the distribution of the input data. Sampling tasks transmit information about the input distribution to the master node. The master then upon receiving sample information from different sample tasks derives an estimate about the data distribution. Based on this sample information, the master makes decision about partitioning and sends this partitioning information to worker nodes. Thus it becomes easy for the worker nodes to partition the intermediate data without any extra overhead. There are various sampling methods like Random sampler, interval sampler, and split sampler. But none of these could be used because we cannot achieve good approximation about the distribution of input data. LIBRA uses its own sampling method.

2.5 Large cluster splitting in LIBRA to mitigate reduce side-skew

In a MapReduce framework, each cluster is processed by a different reducer. The number of keys is equal to the number of clusters as each cluster has different key. So if the cluster is larger than other clusters, the reducer it is allocated to takes longer time than other reducers [12] [3]. This leads to data skew on the reduce side. Consider key1, key2, key3 are three keys associated with the intermediate data with the values as 100, 10, 10 respectively. When we partition them into two reducers, one of the reducer gets key with value 100 and other reducer gets two keys each having value 10. Thus the reducer 1 gets more data to be processed than the reducer 2, leading to data skew. To mitigate this LIBRA provides the larger cluster splitting technique. Using this technique, we can split the larger cluster in such a way that 60% of the cluster with key1 is allocated to reducer 1 and the rest 40% is allocated to reducer 2. Thus it balances the load among the reducers and overcomes the data skew issue as shown in figure below.

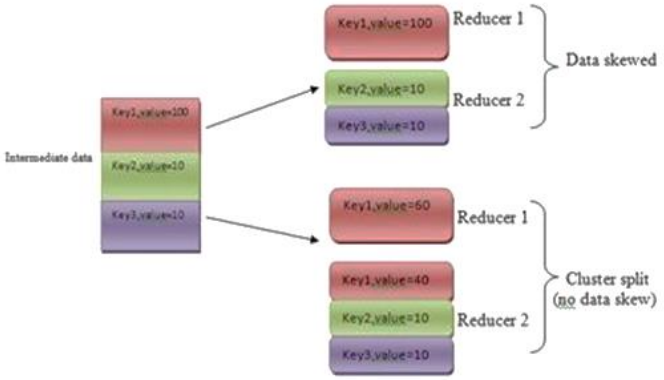

Figure 6. Large cluster split to mitigate reduce-side skew

III. CONCLUSION

To improve the data processing performance in MapReduce, it is important to mitigate the data skew caused in any phase of MapReduce (Map phase or Reduce phase). This paper is a survey of the already existing technique for mitigating the data skew in MapReduce applications. Skew Tune is one of the techniques which does not require any special input from the user instead it observes the complete execution of the job and automatically re-partitions the UN processed data among many tasks as they become available. It maintains the total ordering and partitioning

decisions on the input data. The performance is increased 4 times on a normal MapReduce jobs.

The other technique used is known as LIBRA. LIBRA is mostly used for mitigating the reduce-side skew. It supports the splitting of the large data cluster so that there is no imbalance in the data allocation to reducers. There is a minimal and negligible overhead caused by the sampling method used in LIBRA. Here sample and map operations are combined and these operations take the same time as the normal map operations take. LIBRA increases the reduce side job execution time by partitioning the intermediate data more evenly.

IV. REFERENCES

[1] Joe B. Buck, Noah Watkins, Jeff Lefebvre, Leoni Ioannides, Carlos Matzah, Neola Polyposis, Scott Brandt, "SciHadoop: Array-based Query Processing in Hadoop", UC Santa Cruz, Dept. of Computer Science.

[2] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian,James Majors, Adam Manzanares, and Xiao Qin, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters", Department of Computer Science and Software Engineering Auburn University, Auburn, AL 36849-5347.

[3] Benjamin Gufler, Nikolaus Augsten2, Angelika Reiser2 and Alfons Kempe "Handling data skew in MapReduce", Technische Universit at Munchen, Munchen, Germany 2Free University of Bozen-Bolzano, Bolzano, Italy

[4] YongChul Kwon1, Kai Ren2, Magdalena Balazinska1, and Bill Howe1, "Managing Skew in Hadoop", 1University of Washington, 2Carnegie Mellon University.

[5] YongChul Kwon, Magdalena Balazinska, Bill Howe, "A Study of Skew in MapReduce Applications", University of Washington, USA

[6] JinWoo Lee, SyKyoung Kim, "Study for Performance Improvement of Parallel Process According to Analysis of Hadoop", Computer Engineering Hanbat National University Daejeon, Korea

[7] Weijia Xu, Wei Luo, Nicholas Woodward, "Analysis and Optimization of Data Import with Hadoop", 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum.

[8] Da-Wei, Zhang, Fu-Quan, Sun,Xu Cheng and Chao Liu, "Research on Hadoop-based Enterprise File Cloud Storage System" Information Technology and Business Management Department Dalian Neusoft Institute of Information Dalian, China

[9] AiLing Duan, "Research and Application of Distributed Parallel Search Hadoop Algorithm", 2012 International Conference on Systems and Informatics (ICSAI 2012).

[10] AiLing Duan , HaiFang Si , 1.School of Information Science and Engineering, Henan University of Technology, Zhengzhou,450001, China, "Research and Practice of Distributed Parallel Search Algorithm on Hadoop_MapReduce" , 2012 International Conference on Control Engineering and Communication Technology

[11] Tom white, "Hadoop: the definitive guide.

[12] Qi Chen, Jinyu Yao, and Zhen Xiao, Senior Member, IEEE, "LIBRA: Lightweight DataSkew Mitigation in MapReduce" , IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS.

[13] YongChul Kwon, Magdalena Balazinska, Bill Howe, Jerome Rolia University of Washington, HP Labs, "SkewTune in Action: Mitigating Skew in MapReduce Applications".