SURVEY ON LOAD BALANCING AND DATA SKEW MITIGATION IN MAPREDUCE APPLICATIONS

Vishal A. Nawale\textsuperscript{1}, Prof. Priya Deshpande\textsuperscript{2}

\textsuperscript{1}Information Technology, MAEER’s MITCOE, Kothrud, Pune, India
\textsuperscript{2}Information Technology, MAEER’s MITCOE, Kothrud, Pune, India

ABSTRACT

Since few years Map Reduce programming model have shown great success in processing huge amount of data. Map Reduce is a framework for data-intensive distributed computing of batch jobs. This data-intensive processing creates skew in Map Reduce framework and degrades performance by great value. This leads to greatly varying execution time for the Map Reduce jobs. Due to this varying execution times results into low resource utilization and high overall execution time since new Map Reduce cycle can start only after all reducers are completed. During this paper, we are going to study various methodologies and techniques used to mitigate data skew and partition skew, and describe various advantages and limitations of these approaches.

Keywords: Data Skew, Hadoop, Map Reduce, Partition Skew.

I. INTRODUCTION

Big Data, now a days this term becomes common in Information Technology industry. As today’s situation there is lot of data available in IT industry but there is nothing useful before big data comes into picture. As we know there is huge amount of data surrounds us but we can’t make it useful for us. Reason for this is that we don’t have any traditional system such a powerful that can make out analysis from this huge amount of data. When we usually talk about big data the first name comes in mind is Hadoop, Doug Cutting helped to create Apache Hadoop as requirement for data exploded from the web and grew far beyond the handling of conventional systems. With Hadoop there is no data that’s too big. In today’s world as we know it where daily tremendous amount of data is generated, with the use of Hadoop various industries can now find value in data that was in recent times considered useless.
Hadoop uses Hadoop Distributed File System (HDFS) as the primary distributed storage. HDFS is well known for distributed, scalable and portable file-system using commodity hardware [8].

Name Node/Master Node is having metadata information about whole system such as data stored on data nodes, free space, active nodes, passive nodes, job tracker, task tracker and many other configuration files such as replication of data [8].

Data Node is a type of slave node which is used to save data and task tracker in data node which is used to track the ongoing jobs which are coming from name node [8].

A small-size Hadoop cluster includes one master node and multiple slave nodes. A slave node acts as a Data Node in HDFS architecture and acts as Task Tracker in Mapreduce framework. These types of clusters are used in only non-standard applications. Again Large-size cluster includes a dedicated Name Node which manages all file system index or metadata, a secondary Name Node which periodically generates snapshots of Name Node to prevent or recover from Name Node failure and reducing loss of data. Similarly in MapReduce framework a standalone Job Tracker server which manages job scheduling and several task trackers are running on Data Nodes or slaves [11],[8].

Since from last few years, MapReduce framework is becoming a paradigm for the bulk operations or workloads and a programming model for parallel data processing. MapReduce programs written in many languages like Java, Ruby, Python and C++. Hadoop and MapReduce framework is used by big players of market like Facebook, Google, Yahoo, Amazon, etc. MapReduce works by dividing processing into two phases namely map phase and reduce phase. Both phase has (key, value) pair as input and output [8].

Figure 1: Hadoop Architecture [14]
To maximize the MapReduce performance, preferably we want all tasks to finish around the same time. The job completion time in MapReduce depends on the slowest running task in the job. While MapReduce is a popular data processing tool, it still has several important limitations. In particular, skew is a significant challenge in many applications executed on this platform. When skew arises, some tasks in a job take significantly longer time to process their input data than remaining tasks this causes to slow down the entire computation.

1. Causes of Skew

Data skew can occur in both the map phase and reduce phase. Map skew occurs when some input data are more difficult to process than others, but it is rare and can be easily addressed by simply splitting map tasks. Lin [12] has provided an application-specific solution that split large, expensive records into some smaller ones. Again data skew present in the reduce phase (also called reduce skew or partitioning skew) is much more challenging.

It's well known that skew has many causes. The original MapReduce paper considered the problem of skew caused by resource contention and hardware malfunction. To deal with this type of skew, MapReduce and Hadoop include a mechanism in which the last few tasks of a job are speculatively replicated on a different machine and the job completes when the fastest replicas of these final tasks complete. Mainly there are two causes of Skew: First cause of skew is uneven distribution of data in which data as an input provided to map tasks is unevenly distributed may be in the size of data and/or complexity of the data.
Second cause of skew related to straggler nodes it’s a node which takes more time to process data than other nodes [13].

Figure 3: Uneven distribution of data [13]

Figure 4: Skew caused by straggler node. [13]

II. LITERATURE SURVEY

Qi Chen et al. [1] presents LIBRA a sampling and partitioning algorithm which balances the load on reduce tasks. The architecture of LIBRA system is shown in figure 5. Data skew mitigation in LIBRA consists of the following steps:

From map tasks a small percentage is selected as a sample tasks, these sample tasks are issued first whenever there are free slots are available in the system. Only when all sample tasks are completed then only ordinary map tasks are issued. These Sample tasks collect statistics during normal map tasks processing on the intermediate data and send this collected distribution about data to the master after they complete. The master node collects all the sample information and derives an estimate of the data distribution to take the decision of partition then notifies about this to the slave nodes. When slave nodes receive partition decision then they need to partition the data according to decision and already issued normal map tasks also partition the intermediate data. There is no need for reduce tasks to wait for completion of all issued map tasks; they can be issued as soon as decisions about partitioning are ready.
Sampling method of LIBRA achieves a good estimate to the distribution of the whole original data set. LIBRA reduces the inconsistency or unpredictability of job execution time significantly by partitioning the intermediate data more evenly across reduce tasks. Use of LIBRA can be made in various applications and can raise up to a factor of 4X performance improvement than earlier. LIBRA can be used in both homogeneous and heterogeneous environments. The extra operating cost of LIBRA is minimal and negligible even in the absence of the skew.

Q. Chen et. al.[2] proposes a new speculative execution strategy named Maximum Cost Performance (MCP). When machine unusually takes long time to complete a task this machine is called as straggler machine, this results into increase in the job execution time and decreases the cluster throughput drastically. One common approach is present called Speculative Execution used to deal with the straggler nodes problem, it just backup the straggler task on alternative node.

Various speculative execution strategies are improving the performance in Heterogeneous and homogeneous environment. But there are some drawbacks that degrade the performance. When the existing strategies cannot work well, the new strategy is introduced called MCP, which extends the usefulness of speculative execution strategy. In this strategy cost refers to the computing resources occupied by tasks, the performance refers to the reduction in the job execution time and the boost in the cluster throughput. MCP aims at selecting straggler nodes perfectly and rapidly and backing up on proper worker nodes. It is scalable and gives good performance in both small clusters and large clusters.

To perfectly and promptly identify straggler nodes, MCP provide three methods as follows:
1. Make use of the progress rate and the process bandwidth within a phase to decide on slow tasks.
2. Use EWMA (Exponentially Weighted Moving Average) to make prediction about process speed and to calculate a task’s remaining time.
3. With the consideration of the load on the cluster it uses cost-benefit model to determine which task to backup on another node.

To pick appropriate worker node for backup tasks, it takes both data skew and data locality into consideration.
Y. Kwon et al. [3] designed SkewTune, its API-compatible with Hadoop providing the same capability of parallel job execution environment while it’s added capabilities to detect and mitigate skew. SkewTune is designed to work for MapReduce type systems where each operator reads data from disk and writes data to disk and is thus decoupled from other operators in the data flow. SkewTune continuously monitors the execution of a UDO (User Defined Operations) and detects the current bottleneck task that dominates and delays the job completion. When such a task is identified, the processing of that task is stopped and its remaining unprocessed input data is repartitioned among idle nodes. The repartition occurs only when there is an idle node present or when new nodes are dynamically added to accelerate the job. Thus if there is not present any task experiencing considerable data skew, SkewTune imposes no overhead. If a node is idle and at least one task is expected to take a long time to complete, SkewTune stops the task with the longest expected time remaining. It repartitions data on other idle nodes and then parallelizes the remaining input data for that task taking into consideration to expected future availability of all nodes in the cluster. SkewTune continuously detects and removes bottleneck tasks until the job completes.

When repartitioning a task, SkewTune takes into account the predicted availability of all nodes in the cluster to minimize the job completion time. The availability is estimated from the progress of running tasks. An unprocessed remaining input data of a task is repartitioned on the available nodes and utilize them fully or nodes that are expected to become available after some time. Again SkewTune concatenate all the repartitioned data to construct the final output exactly same as if there wasn’t any partition happened.

Y. Kwon et al. [4] developed a static optimizer that takes a User-defined Operation (UDO) as input and outputs a data partition plan that balance load among all processing nodes known as SkewReduce. SkewReduce does not only balance the amount of data assigned to each operator partition, but rather the expected processing time for the assigned data. While a static optimizer could be applied to different types of UDOs, SkewReduce was actually designed for feature extracting applications. In these applications, emergent features are extracted from the data items that are embedded in a metric space. These applications are captured by two UDOs: one that extracts features from sub-spaces and another that merges features that cross partition boundaries.

The key idea behind SkewReduce is to ask the user for extra information about their UDOs (the feature extraction and merge functions). The user creates additional cost functions that characterize the UDOs runtimes. Firstly the cost function takes as input a sample of the input data, the sampling rate that this sample corresponds to and secondly the bounding hypercube where the sample was actually taken from. In response the function returns a real number, which should represent an estimate of the processing time for the data. The sampling rate $\alpha$ allows the cost function to properly scale up the estimate to the overall dataset. The best cost function returns the values proportional to the actual cost of processing the data partition bounded by the hypercube. The SkewReduce plans quality depends on the quality of the cost functions. However, author found that even inaccurate cost functions could help reduce runtimes compared to no optimization at all. For the domain experts such as statisticians and scientists, author assume that writing a cost model is easier than debugging and optimizing distributed programs. The cost models are specific to the implementation and can be reused across many data sets for the same UDOs.

Given the cost models, a sample of the input data, and the cluster configuration (e.g., the number of nodes and the scheduling algorithm), SkewReduce searches a good partition plan for the input data by (a) when there is considerable data skew is expected for some part of data then applying fine grained data partitioning, (b) when data skew is not expected applying coarse grained data partitioning, and (c) balancing the partition granularity in a way that minimizes expected job completion time under the given cluster configuration. More specifically, the algorithm proceeds as follows. Starting from a single partition that corresponds to the entire hypercube bounding the input data, and thus also the data sample, the optimizer greedily splits the most expensive leaf partition in
the current partition plan. The optimizer stops splitting partitions when two conditions are met: (a) All partitions can be processed and merged without running out of memory; (b) No further leaf-node split improves the runtime: i.e. further splitting a node increases the expected runtime compared to the current plan. To check the second condition, the SkewReduce optimizer first checks that the original execution time of the partition is greater than the expected execution time of the two sub-partitions running in parallel, followed by a newly added merge function to reconcile features found in individual sub partitions. If the condition is satisfied, SkewReduce further verifies that the resulting tasks can also be scheduled in the cluster in a manner that reduces the runtime. It proceeds with the split only when both conditions are met.

Wei Yan et al.[5] presented a scalable solution to achieve load balanced record linkage over the mapreduce framework. The solution in this paper contains two low memory load balancing algorithms that work with a sketch based approximate data profiles. The solution provided in this paper divided in two parts as follows. First, sketch based profiling method 1) scalable with the size of the number of blocks and input data and 2) efficient for constructing sketch profile, such that constant time taken by each update. Second proposed Cell Block division and Cell Range division algorithms can efficiently break up expensive blocks without losing any record pairs that need to be compared. A sketch is a two-dimensional array of cells, each indexed by a set of pair wise independent hash function.

![Figure 6: The workflow of MapReduce-based record linkage facilitated by sketch [5]](image)

Figure 6 shows the overall design of system with a representation of linking two data sets R and S. This design is based on two rounds of MapReduce jobs. The profiling job analyzes two input data sets and provides the load estimation in terms of sketch. This generated load estimation information is passed by map tasks into the second MapReduce job (comparison), where load balancing strategies are applied to perform the actual record linkage task.

In Sketch based data profiling, FastAGMS sketch is used because it provides the most accurate estimation for the size of join operation, regardless of data skew. Two FastAGMS sketches are maintained to estimate workload in the form of record-pair comparisons within each block for data sets R and S. The map tasks of the profiling job builds local sketches based on the input data from two data sets. After completing map tasks local sketches are sent to one reducer, where they are combined to build the final sketch by entry-wise adding all local sketches.
In Cell Block Division Algorithm, final sketch from profiling job provides an estimation of the record comparison workload, where each cell carries the estimated workload for multiple blocks. A simple idea that comes into mind is to pack these cells into \( n \) partitions and assign each reducer its own partition. It may happen some cells have workload larger than the average, so a division procedure is required for large cells to divide it into subcells otherwise keep the cell as a single subcell.

In Cell Range Division Algorithm, as the Cell Block division algorithm divides larger cells, this may still lead to imbalanced reducer’s workload due to variation in the size of the subcells. To overcome this new more sophisticated pair-based load balancing strategy that strives to generate a uniform number of pairs for all reduce tasks. Each map task processes sketch and can therefore enumerates the workload per cell. Label each record pair in sketch with a global index and divide all record pairs into \( n \)-equal length ranges.

Benjamin Gufler et al.\[6\] present TopCluster, a sophisticated distributed monitoring approach for load balancing in MapReduce systems. TopCluster requires cluster threshold which controls the size of the local statistics that are sent from each mapper to the controller. The result is a global histogram of (key, cardinality) pairs, which approximates the cardinalities of the clusters with the most frequent keys. This global histogram is used to estimate the partition cost. As the global histogram contains the largest clusters, the data skew is considered in the cost estimation.

The TopCluster algorithm computes an approximation of the global histogram for each partition in three steps: 1) Local Histogram (mapper): For each partition a local histogram is maintained on mapper. 2) Communication: After mapper completes it sends following information to the controller a) the presence indicator for all local clusters and b) the histogram for the largest local clusters. 3) Global Histogram (controller): The control approximates the global histogram by a) sum aggregating heads of the local histograms from all mappers and b) By computing a cardinality estimation for all clusters that are not in the local histograms.

1. **Challenging Issues**

In MapReduce system to improve performance of applications skew is the big issue, again this skew is present in both map phase and reduce phase also. But reduce skew or partition skew is much more challenging as it requires appropriate distribution of clusters to reduce tasks. Efficient and scalable cluster partitioning scheme is required to overcome reduce skew and to improve throughput of MapReduce applications.

2. **Summary**

MapReduce applications causes degrade in their performance due to presence of skew in input data. To tackle this problem various authors have proposed their solutions, we will make analysis of them as follows:

LIBRA \[1\] supports large cluster split. Evaluation of performance in both synthetic and real workloads demonstrates that the resulting performance improvement is significant. LIBRA has adjustment for heterogeneous environments. Overhead is minimal and negligible in the absence of skew.

MCP\[2\] shows good results, can run jobs up to 39\% faster and improves the cluster throughput by up to 44\% compared to Hadoop release 0.21. It got success in minimizing the skew by detecting straggler nodes and introduces increase in cluster throughput also. But MCP only considers speculative execution at the end of a stage; this can be future scope for this paper.

SkewTune\[3\] shows improvement by a factor of 4 over Hadoop, it is broadly applicable as it makes no assumptions about the cause of skew. SkewTune preserves the order and partitioning properties of the output. Limitations of SkewTune as follows:
1. It occupies more task slots than regular to run the skew mitigation task.
2. It cannot split exact large keys because it does not sample any key frequency information. Therefore, as long as large keys are being gathered and processed, the system cannot rearrange them.
3. It mitigate skew in reduce phase but cannot improve copy and sort phases, which may be the performance bottleneck for some applications.

SkewReduce[4] improves the execution speed by a factor of up to 8 over default Hadoop and runtime improves even if the cost functions provided by user were not perfectly accurate but that are good enough to identify expensive results in the data. Limitation of SkewReduce is that it puts an extra burden on user who must provide cost functions.

Wei Yan et al.[5] shows advantage as load balancing algorithms can decrease the overall job completion time by 71.56% and 70.73% of the default settings in Hadoop and both algorithms proposed have same load balancing performance while requiring much less memory.

TopCluster[6] shows advantage as it’s tailored to suit the characteristics of MapReduce frameworks, especially in its communication behavior and also it provides a good basis for partition cost estimation, which in turn required for effective load balancing.

III. CONCLUSION AND FUTURE WORK

In this paper, we have surveyed common causes of skew in MapReduce applications and various techniques to tackle or minimize skew for the MapReduce so that faster processing of the jobs will be done in the system. We have summarized current gaps in the research. In future work, we will focus on solving skewed data distributions on the mappers and consider analyzing more sophisticated statistics of data distribution in order to estimate the workload per partition more accurately.

REFERENCES