International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 4; Issue: 3; March -2015

Mitigating Data Skew Using Map Reduce Application

Ms. Archana P.M
4th sem, M.Tech (C.S.E)
M.S.E.C, V.T.U Bangalore, India
archanaanil062@gmail.com

Mr. Malathesh S.H
Associate Professor C.S.E Dept.
M.S.E.C, V.T.U Bangalore, India
mhavanur@gmail.com

***Abstract*-There is a growing need for ad-hoc analysis of extremely large data sets, especially at internet companies where innovation critically depends on being able to analyze terabytes of data collected every day. Parallel database products, over a solution, but are usually prohibitively expensive at this scale. Besides, many of the people who analyze this data are entrenched procedural programmers. The success of the more procedural map-reduce programming model, and its associated scalable implementations on commodity hardware, is evidence of the above. However, the map-reduce paradigm is too low-level and rigid, and leads to a great deal of custom user code that is hard to maintain, and reuse. The map reduce is an effective tool for parallel data processing. One significant issue in practical map reduce application is the data skew. The imbalance of the amount of the data assigned to each tasks to take much longer to finish than the others.**

Keywords: **Map Reduce, Data Skew, Sampling, Partitioning.**

1 INTRODUCTION

Hadoop is an open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage. Map reduce is an important programming model for large-scale data-parallel applications such as web indexing, data mining, and scientific simulation. Hadoop is an open source implementation of Map Reduce enjoying wide adoption and is often used for short jobs. The completion of the job in the Hadoop depends on the slowest running task in the job. If one task is significantly running task in the job. If one task is significantly longer to finish than others. It can delay the progress of entire jobs. The straggler can be happen from various region like among which data skew is important one. The data skew refers to the imbalance the amount of work required to process such data. Hadoop's performance is closely tied to its task scheduler, which implicitly assumes that cluster nodes are

homogeneous and tasks make progress linearly, and uses these assumptions to decide when to speculatively re-execute tasks that appear to be stragglers[2].

1.1The main objective of this paper summarized as follows.

- Implement the method for general user defined Map reduce programs. The method has better approximation to the distribution of the intermediate data.
- The LIBRA can adjust the work load allocation and deliver improved performance even in the absence of data skew when the performance underlying computing platform.
- Implement the innovative approach to balance the load among the reduce tasks which supports the split of large key when application semantics permit.
- Implement the LIBRA in the Hadoop and evaluate the performance for the some popular applications.
- The LIBRA performance is higher than the other. The result will show that LIBRA can improve the job execution time to a factor 4.

2 LITERATURE SURVEY

Our implementation of Map Reduce runs on a large cluster of commodity machines and is highly scalable: a typical Map Reduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of Map Reduce programs have been implemented and upwards of one thousand Map Reduce jobs are executed on Google's clusters every day[1]. The standard approach to handling skew in parallel systems is to assign an equal number of data values to each partition via hash partitioning or clever range partitioning. These strategies effectively handle data skew, which occurs when some nodes are assigned more data than others. Computation skew, more generally, results when some nodes take longer to process their input than other nodes and can occur even in the absence of data skew | the runtime of many scientific tasks depends on the data values themselves rather than simply the data size. These histograms were also shown earlier to be optimal for join selectivity estimation, thus establishing their universality. In this paper, presents a new strategy called LIBRA (Lightweight Implementation of Balanced Range Assignment) to solve the data skew problem for reduce-side applications in Map Reduce. Compared to the previous work, our contributions include the following: We propose a new sampling method for general user defined Map Reduce programs. The

method has a high degree of parallelism and very little overhead, which can achieve a much better approximation to the distribution of the intermediate data. We use an innovative approach to balance the load among the reduce tasks which supports the split of large keys when application semantics permit. Figure 1 shows that with our LIBRA method, each reducer processes roughly the same amount of data.

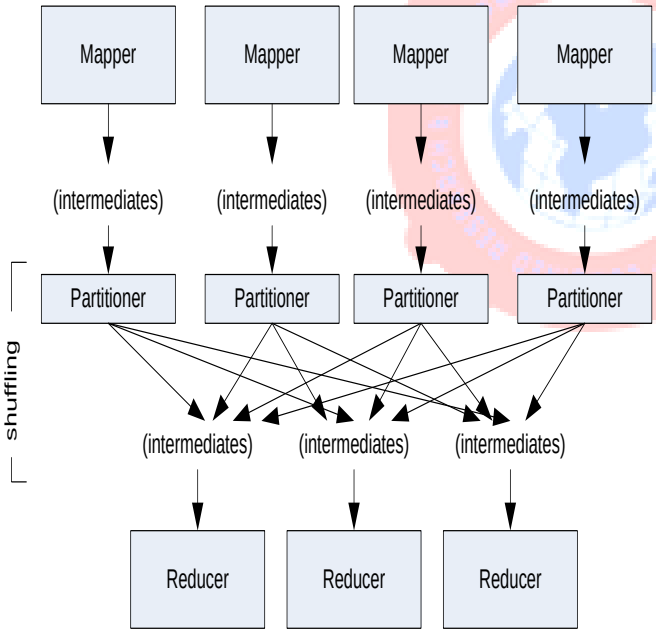


Figure 1: scenario of proposed system

Data skew mitigation in LIBRA consists of the following steps:

- Small percentage of the original map tasks are selected as the sample tasks. They are issued first whenever the system has free slots. Other ordinary map tasks are issued only when there is no pending sample task to issue.
- Sample tasks collect statistics on the intermediate data during normal map processing and transmit a digest of that information to the master after they complete.
- The master collects all the sample information to derive an estimate of the data distribution, makes the partition decision and notifies the worker nodes.
- Upon receipt of the partition decision, the worker nodes need to partition the intermediate data generated by the sample tasks and already issued ordinary map tasks accordingly. Subsequently issued map tasks can partition the intermediate data directly without any extra overhead.

  Reduce tasks can be issued as soon as the partition decision is ready. They do not need to wait for all map tasks to finish.

In a Map Reduce system, a typical job execution consists of the following steps: 1) After the job is submitted to the Map Reduce system, the input files are divided into multiple parts and assigned to a group of map tasks for parallel processing. 2) Each map task transforms its input (K1, V1) tuples into intermediate (K2, V2) tuples according to some

user defined map and combine functions, and outputs them to the local disk. 3) Each reduce task copies its input pieces from all map tasks, sorts them into a single stream by a multi-way merge, and generates the final (K3, V3) results according to some user defined reduce function.

2.1 Algorithm:

In this section the sampling and partitioning algorithm in LIBRA as shown below. Our goal is to balance the load across reduce tasks.

The algorithm consists of three steps:

1) Sample partial map tasks

2) Estimate intermediate data distribution

3) Apply range partition

In the following, we will describe the details of these steps.

Problem Statement

We first give a formulation of the intermediate data between the map and the reduce phases can be represented as a set of tuples: (K_1,C_1), $(K_2,C_2),\ldots,(K_n,C_n)$, where K_i represents a distinct key in the map output, and C_i represents the number of tuples in the cluster of K_i. Without loss of generality, we assume that $K_i < K_{i+1}$ in the above list. Then our goal is to come up with a range partition on keys which minimizes the load of the largest reduce task. Let r be the number of reduce tasks. The range partition can be expressed as: $0 = pt_0 < pt_1 < \ldots < pt_r = n$ with reduce task i taking responsibility of keys in the range of $[K_{pt_{i-1}+1}, K_{pt_i}]$. Following the cost model proposed by previous work [3], [4],[5] we define the function $Cost(C_i)$ as the computational complexity of processing the cluster K_i in reduce tasks which must be specified by the users. For example, the cost function of the sort application can be estimated as $Cost(C_i) = C_i$ (for each cluster K_i, reducers only need to output C_i tuples directly). For reduce side self-join application, the cost function should be c_i^2 since reducers need to output C_i tuples for each tuple in cluster K_i. By specifying the exact cost function, we can balance the execution time of each reducer one step further. Then the objective function can be expressed as follows:

$$\text{Minimize} \quad \max_{i=1,2,\ldots,r} \{ \sum_{j=pt_{i-1}+1}^{pt_i} Cost(C_j) \}$$

Since the number of unique keys can be large, calculating the optimal solution to the above problem is unrealistic. Therefore, we present a distributed approximation algorithm by sampling and estimation.

Sampling Strategy

After a specific map task j is chosen for sampling, its normal execution will be plugged in with a lightweight sampling procedure. Along with the map execution, this procedure collects a statistic of (K^j_i, C^j_i) for each key K^j_i in the output of this task, where C^j_i is the frequency (i.e., the number of records) of key K^j_i. Since the number of such $(K^j_i; C^j_i)$ tuples can be on the same order of magnitude

as the input data size, we keep only a sample set S_{sample} containing the following two parts:

_ $S_{largest}$: p tuples with the largest C^j_i

_ S_{normal}: q tuples randomly selected from the rest according to uniform distribution (excluding tuples in $S_{largest}$)

This sampling task then transmits the following statistics to the master: the sample set $S_{sample} = S_{largest} \cup S_{normal}$, the total number of records (TR_j) and the total number of distinct clusters (TC_j) generated by this task. The size of the sample set $p + q$ is constrained by the amount of memory and the network bandwidth at the master. The larger $p + q$ is, the more accurate approximation to the real data distribution we will achieve. In practice, we find that a small $p+q$ value has already reached a good approximation and brings negligible overhead.

Estimate Intermediate Data Distribution

After the completion of all sample map tasks, the master aggregates the sampling information in the above step to estimate the distribution of the data. It first combines all the sample tuples with the same key into one tuple (K_i,C_i) by adding up their frequency It then sorts these combined tuples to generate an aggregated list L. Suppose there are m maps for sampling and S^j_{sample} is the sample set of map j. Then the aggregated list L is:

$$L = \{(K_i, C_i = \sum_{j=1}^{m} \{C_l^j | K_l^j = K_i\})\}, K_i < K_{i+1}$$

To calculate the total number of records TR, we simply sum up the record counts in all sample map tasks. However, calculating the total number of distinct clusters TC is hard because clusters processed by different map tasks may share the same key and hence should not be counted twice. For example, assume that there are two sample map tasks and their sample sets are: *{(A;* 10), (*B;* 5), (*C;* 3), (*D;* 2), (*E;* 2)*}*, *{(A;* 20), (*B;* 3), (*D;* 1), (*F;* 1), (*H;* 1)*}*, in which p = 2 and q = 3. By summing up the frequencies of the same key, the merged sample set S_{sample} is *{(A;* 30), (*B;* 8), (*C;* 3), (*D;* 3), (*E;* 2), (*F;* 1), (*H;* 1)*}*.

Range Partition

We adopt the above approximation to the data distribution to get an approximate solution to the range partition. We need to generate a list of partition points in the aggregated list L where $0 = pt_0 < pt_1 < \ldots < pt_r = |L|$ and minimize:

$$\max_{i=1\ldots r} \{ \sum_{j=pt_{i-1}+1}^{pt_i} \{P_j \times Cost(Q_j)\}\}$$

We use dynamic programming to solve this optimization problem: let $F(i; j)$ represent the minimum value of the largest partition sum of cutting the first i items into j partitions,

and $W(a,b) = \sum_{l=a}^{b} \{P_l \times Cost(Q_l)\}$.

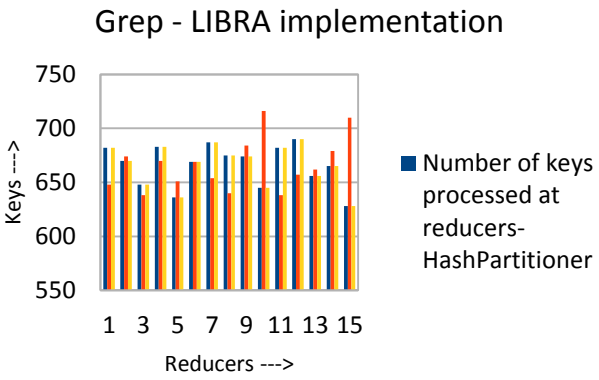
Then the recursive formulation of $F(i; j)$ is:

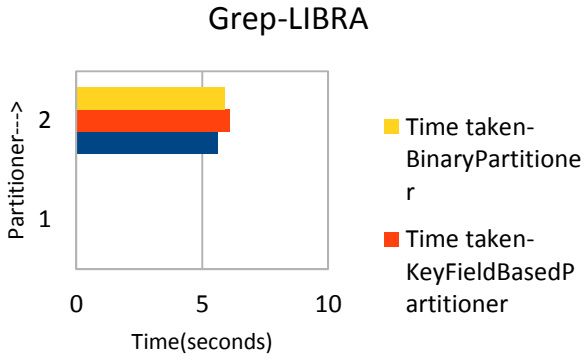
$$F(i,j) = \min_{k=j-1\ldots i-1} \{\max\{F(k,j-1), W(k+1,i)\}\}$$

The partition decision can be derived from optimized decision of $F(i; j)$.

3. PERFORMANCE EVALUATION

A LIBRA program can be run on Grep application and analyzed the split of data for various partitions, also checked the job execution time. The histogram and the time for all the partitions are shown below:





4. RESULTS

Our research work on this paper signify the program for the different partitioner, and found that the new sampling strategy improves the performance significantly, where the data gets evenly distributed across all the reducers, thereby reducing the data skew. Also, the job execution times gets improved significantly.

CONCLUSION

The Map Reduce programming model has been successfully used at Google for many different purposes. We attribute this success to several reasons. First, the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as Map Reduce computations. Data skew mitigation is important in improving Map Reduce performance. This paper has presented LIBRA, a system that implements a set of innovative skew mitigation

strategies in an existing Map Reduce system. One unique feature of LIBRA is its support of large cluster split and its adjustment for heterogeneous environments. In some sense, we can handle not only the data skew, but also the reducer skew.

REFERENCES

[1] J. Dean and S. Ghemawat,"Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, January 2008.

[2] Y. Kwon, M. Balazinska, and B. Howe, "A study of skew in mapreduce applications," in *Proc. of the Open Cirrus Summit*, 2011.

[3] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skew-resistant parallel processing of feature-extracting scientific user-defined functions," in *Proc. of the ACM symposium on Cloud computing (SoCC)*, 2010.

[4] S. Ibrahim, J. Hai, L. Lu, W. Song, H. Bingsheng, and Q. Li, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in *Proc. of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010.

[5] G. Benjamin, A. Nikolaus, R. Angelika, and K. Alfons, "Handling data skew in mapreduce," in *Proc. of the International Conference on Cloud Computing and Sesrvices Science (CLOSER)*, 2011.