International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), Volume 5, Special Issue 2, May 2015)
International Conference on Advances in Computer and Communication Engineering (ACCE-2015)

Skew Types Mitigating Techniques to Increase the Performance of MapReduce Applications

Vinutha J[1], Chandramma R[2]

[1]*Computer Science Department, Vivekananda Institute of Technology, Gudimavu, Kengeri Hobli, Bangalore-74*
[2]*Asst.Prof & HOD Computer Science Department, Vivekananda Institute of Technology, Gudimavu, Kengeri Hobli, Bangalore-74*

[1]vinuthaj.mtech@gmail.com
[2]rchandramma.vkit@gmail.com

***Abstract* — Now a day's large internet companies routinely generate more number of tera-bytes of logs and operation records. MapReduce is a programming model which has proven itself to be an effective model to process such large datasets. In MapReduce applications one significant issue is Skew. Skew occurs when a job's tasks have different processing requirements, due to processing uneven amounts of data or performing uneven amounts of work per record. Hadoop is an open source implementation of MapReduce. In this Paper a different types of skew and their mitigating techniques has been explained.**

***Keywords*— Hadoop, Issue, Job, MapReduce, Skew**

I. INTRODUCTION

MapReduce[1] is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world application tasks. Users specify the computation in terms of a map and a reduce function, and the runtime system automatically parallelizes the computation across large-scale clusters of commodity machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. MapReduce is a programming model initiated by Google's Team for processing huge datasets in distributed systems; it helps programmers to write programs that process big data and programmers are restricted to formulate the computation in two functions: Map and Reduce.

Applications in different domains such as bioinformatics, weather forecasting, environmental studies, and fraud detection deal with intensive data. It manages data partitioning, task scheduling, and nodes failure. The run-time system takes care of the details of partitioning the user input data, scheduling the program's execution across a set of commodity machines, handling machine failures, and managing the required inter-machine communication.

1.1 Programming Model

The computation takes a set of input (key,value) pairs, and produces a set of output (key,value) pairs. The user of the MapReduce library expresses the computation as two functions like Map and Reduce. Map, takes an input pair and produces a set of intermediate (key,value) pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function then the Reduce function, uses an intermediate key I and a set of values for that key. Finally it merges together these values to form a possibly smaller set of values. Each reduce invocation get zero or one output value. The intermediate values are supplied to the user's reduce function via an iterator.

Examples consider the problem of counting the number of occurrences of each word in a large collection of documents. The pseudo-code:

```
map(String key, String value):
// key: document name          // value: document contents
for each word w in value:
EmitIntermediate (w, "1");

reduce(String key, Iterator values):
// key: a word                 // values: a list of counts
Int result = 0;
for each v in values:
result += ParseInt(v);
Emit(AsString(result));
```

The map function emits each word plus an associated count of occurrences as just `1' in this example. Finally the reduce function sums together all counts emitted for a particular word.

The MapReduce 2 types are like

```
map (k1,v1)          -> list(k2,v2)
reduce (k2,list(v2)) -> list(v2)
```

1.2 *Examples* a few examples which can be expressed in MapReduce computations.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), Volume 5, Special Issue 2, May 2015)
International Conference on Advances in Computer and Communication Engineering (ACCE-2015)

Distributed Grep: The map function emits a line if it matches a given input pattern. The reduce function acts as an identity function that just copies the supplied intermediate data to the output.

Count of URL Access Frequency: The map function processes logs of web page requests and outputs <URL,1>. The reduce function adds together all values for the same URL and emits a <URL, total count> pair.

ReverseWeb-Link Graph: The map function outputs <target,source> pairs for each link to a target URL found in a page url named source. The reduce function adds the list of all source URLs associated with a given target URL and emits the pair: <target, list(source)>

Term-Vector per Host: It summarizes the most important words that occur in a document or a set of documents as a list of <word, frequency> pairs. The map function emits <hostname, term vector> pair for each input document where the hostname is extracted from the URL of the document. The reduce function is passed to each document term vectors for a given hostname. It adds these final term vectors , throwing away infrequent terms, and then emits a final <hostname, term vector> pair.

1.3 Execution Overview The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M map splits. The input splits can be processed in parallel by different variety machines. Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function (e.g hash(key) mod R). The number of partitions (R) and the partitioning function are specified by the user.

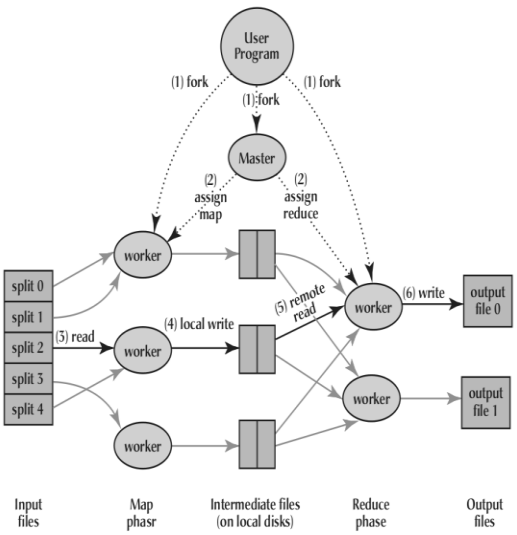

Fig. 1. MapReduce Execution Overview

Fig.1 shows the overall flow of a MapReduce operation in implementation stage. When the user program calls the MapReduce function, the following different sequence of actions occurs.

1. The MapReduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes MB per piece. Then it starts up many copies of the program on a cluster of machines.

2. One of the copies of the program is assign as the master. The rest are workers that are assigned work by the master. There are different M map tasks and R reduce tasks to assign. The master choose idle workers and assigns each one a map task or a reduce task.

3. A worker who is assigned a map task reads the contents of the corresponding associated input split. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate (key,value) pairs produced by the Map function are buffered in memory.

4. Periodically, the buffered pairs are written to ordinary local disk, which are partitioned into R regions by using different separating function. The locations of these buffered pairs on the local disk are again passed back to the master, then master is responsible for forwarding these locations to the reduce workers.

5. When a reduce worker is notified by the master about these buffered locations, it uses remote procedure calls to read the buffered data from the local disks of the individual map workers. When a reduce worker has read all intermediate data, it categorize it by the intermediate keys so that all occurrences of the same key are categorized together. If the number of intermediate data is too large to competent in memory, an external sort is used.

6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key determined, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is copied to a final output file for these reduce partition.

7. When all map tasks and reduce tasks have been completed, the master calls up the user written program. The MapReduce call in the user written program returns back to the user code. After completion, the output of the MapReduce execution is available in the R output files.

1.4 Mapreduce Background [2]

Dealing with Failure MapReduce is designed to deal with hundreds or thousands of commodity machines. Therefore, it must tolerate machine failure. The failure may be occur in master node or worker nodes.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), Volume 5, Special Issue 2, May 2015)
International Conference on Advances in Computer and Communication Engineering (ACCE-2015)

In case of master failure all MapReduce task will be aborted, and it have to be redone after assigning new master node. On the other hand, to track worker failure, the master monitors all workers by periodically checking worker status. If a worker doesn't respond to master ping in a certain amount of time, the master marks the worker as failed. The output of completed reduce tasks is stored in global file system, so completed reduce tasks do not need to be re-executed. In the other hand, the output of map tasks is stored in local disks, so completed map tasks must be re-executed in case of failure.

Advantages MapReduce supports data locality by collocating the data with the compute node; so it reduces network communication cost [3] and it support scalability; the runtime scheduling strategy enables MapReduce to offer elastic scalability which means the ability of dynamically adjusting resources during job execution. It has a fault tolerance strategy that transparently handles failure; it detects map and reduce tasks of failed nodes and reassigns it to other nodes in the cluster. It has the ability to handle data for heterogeneous system, since MapReduce is storage independent, and it can analyze data stored in different storage system.

1.5 Hadoop's Overview

Hadoop is an open source software framework for distributed storage and distributed processing of big data on clusters of commodity hardware. It is a java open source implementation of MapReduce. The two fundamental subprojects are the Hadoop MapReduce framework and the HDFS.

HDFS is a distributed file system that provides high throughput access to application data. It is inspired by the GFS. HDFS has master/slave architecture. The master server called by NameNode, splits files into blocks and distributes them across the cluster with replications for fault tolerance. It holds all metadata information about stored files. The master, called Job Tracker (JT), is responsible of : (a) Querying the NameNode for the block locations, (b) Considering the information retrieved by the NameNode, Job Tracker schedule the tasks on the slaves, called Task Trackers (TT), and (c) monitoring the success and failures of the tasks. The HDFS slaves, the actual store of the data blocks called DataNodes, serve read/write requests from clients and propagate replication tasks as directed by the NameNode.

In hadoop for job execution the MapReduce program is divided into two phases, map and reduce. For the each map side, it starts by reading the records in the Map process, and then the map function processes a data chunk into key/value pairs, on which the hash partitioning function is performed.

This intermediate result referred as record, is stored with its associate partition in the buffer memory which includes 100 MB for each map. If the buffered data reaches the buffer threshold 80% of the total size, the intermediate data will be sorted according to the partition number and then by key and spilled to the local disk as an index file and a data file. All files will be then combined as one final indexed file–Keys' Frequencies Variation.

The basic Apache Hadoop framework is composed of the following modules

- Hadoop Distributed File System (HDFS) - a distributed file system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.
- Hadoop YARN- an unique resource management platform responsible for managing computer resources in clusters and using them for scheduling of users applications.
- Hadoop MapReduce – a programming model for large scale data processing.

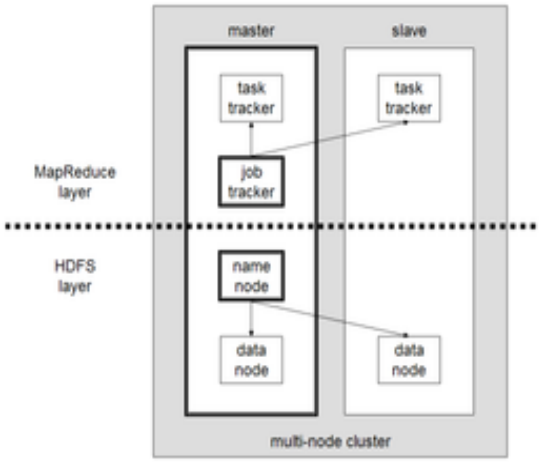

Fig. 2. Hadoop's Architecture

II. Types Of Skew And Their Mitigating Techniques In Mapreduce Applications

2.1 SkewTune: Mitigating Skew in MapReduce Applications [4]

SkewTune,a new technique for handling skew in parallel user-defined operations (UDOs). It is designed for MapReduce-type engines, characterized by disk-based processing and a record-oriented data model and it automatically mitigates skew in user-defined MapReduce programs.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), Volume 5, Special Issue 2, May 2015)
International Conference on Advances in Computer and Communication Engineering (ACCE-2015)

SkewTune relies on two properties of the MapReduce model: (1) MapReduce's ability to buffer the output of an operator before transmitting it to the next operator and (2) operator de-coupling, where each operator processes data as fast as possible without back-pressure from downstream operators.

SkewTune uses PageRank [5] as an example of a UDO. This UDO is expressed as a MapReduce job, then this job is run in two main phases: the map phase and the reduce phase. In each map and reduce phase, a subset of the input data is processed by distributed tasks in a cluster of computers. Each task corresponds to a partition of the UDO. When a map task completes, the reduce tasks are notified to pull newly obtained data. This transfer process is referred to as a shuffle**.** All map tasks must complete before the shuffle part of the reduce phase can complete, allowing the reduce phase to begin. Load imbalance can occur either during the map or reduce phases. We refer to such an imbalanced situation as map-skew and reduce-skew respectively.

2.2.2 Four common types of skew that SkewTune is designed to address.

*1. Map phase: Expensive Record***.** Map tasks process a collection of records in the form of key-value pairs, one-by-one. Ideally, the processing time does not vary significantly from one record to another record. However, based on the application, some records may require more CPU and memory to process than other records. These expensive records may simply be larger than other records comparatively, or the map algorithm's runtime may depend on the record value. PageRank is an application that can experience map phase: expensive record skew. PageRank is a new link analysis algorithm that assigns weights (ranks) to each vertex in a graph by iteratively aggregating the weights of its inbound neighbors. Vertexes with a large out degree take disproportionately longer to process because the map generates an output tuple per outgoing edge.

2. *Map phase: Heterogeneous Map*. MapReduce is a unary operator, but can be used to emulate an n-ary operation by logically concatenating multiple datasets as a individual input. Each dataset may require distinct processing, leading to a multi-modal distribution of task runtimes. CloudBurst is a MapReduce implementation of the RMAP algorithm for short-read gene alignment2, which aligns a set of genome sequence, reads against a reference sequence. CloudBurst scatter the appropriate alignment computation across reduces tasks by partitioning n-grams of both sequences and reads. As a skew-mitigation strategy, the sequences bearing frequent n-grams are replicated across reduce tasks, while other different sequences are hash-partitioned. These two algorithms exhibit different runtimes.

*3. Reduce phase: Partitioning skew***.** In MapReduce, the outputs of map tasks are distributed among reduce tasks via hash partitioning (by default) or some user-defined partitioning logic. The default hash partitioning is usually adequate to evenly distribute the data. However, hash partitioning does not guarantee an even distribution. For example, in the inverted index building application, if the hash function partitions the data based on the first letter of a word, reducers processing more popular letters are assigned a disproportional amount of data.

4. Reduce phase: Expensive Key Group. In MapReduce, reduce tasks process a sequence of (key, set of values) pairs, called key groups. As in the case of expensive records processed by map, expensive key groups can skew the runtime of reduce tasks.

SkewTune mitigates skew at runtime by repeating the following three steps.

1. Detect: The coordinator observes the execution of all tasks in a MapReduce phase and collects their time remaining estimates. When a slot becomes idle, the coordinator identifies the straggler, which is the task with the longest time remaining estimate.
2. Scan: The coordinator stops the straggler task. The unprocessed data is scanned either locally or in parallel to collect information for repartitioning. The information is sent to the coordinator.
3. Plan: The coordinator plans how to repartition the remaining data of the straggler task using the information from the scan and the time remaining estimates of all running tasks. Once the mitigators are scheduled, the coordinator goes back to the Detect phase and continues. SkewTune delivers up to a factor of 4X improvement on real datasets and real UDOs.

2.2 Skew-Resistant Parallel Processing of Feature-Extracting Scientific User-Defined Functions [6]

SkewReduce, a new shared-nothing parallel data processing system designed to support spatial feature extraction applications common in scientific data analysis and observe that these applications share a common structure that can be parallelized using the following strategy: (1) Partition the multidimensional space and assign each node a contiguous region,

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), Volume 5, Special Issue 2, May 2015)
International Conference on Advances in Computer and Communication Engineering (ACCE-2015)

(2) run a serial form of the analysis locally on each region, extracting locally found features and labeling the input data with these features if necessary (3) efficiently merge the local results by considering only those features that cross region boundaries, relabeling the input data as necessary.

SkewReduce system is an efficient optimizer, parameterized by user-defined different cost functions, that identifies how best to partition the input data to minimize computational skew, which results when some nodes take longer to process their input than other nodes. SkewReduce has two components. The first component is an API for expressing spatial feature-extraction algorithms. The functions in our API are translated into a dataflow that can run in a MapReduce-type platform .The second component of SkewReduce is a static optimizer that partitions the data to ensure skew-resistant processing if possible. The data partitioning is guided by a user-defined cost function that estimates processing times, collection of features and the input representation as output by the process and merge functions.

Applications: Astronomy Simulation, Cosmological simulations are used to study the structural evolution of the universe on distance scales ranging from a few million light-years to several billion light-years. In these simulations, the universe is modeled as a set of particles. These particles constitute gas, dark matter, and stars and interact with each other through gravity and dynamics. Every few simulation time steps, the simulator outputs a unique snapshot of the universe as a list of particles, each tagged with its unique identifier, location, velocity and other properties. Each of these scientific applications follow a similar pattern: data items (events, particles, pixels) are embedded in a metric space, and the task is to identify and extract emergent features from the low-level data such as populations, galaxies.

Skewresistant algorithm is the clustering algorithm which is typically executed on one snapshot at a time [7]. Given the size of individual snapshots, astronomers would like to run their clustering algorithms on a parallel data processing platform in a shared-nothing cluster.

2.3 LIBHPT: Lightweight Data Skew Mitigation in MapReduce [8]

The MapReduce is an effective tool for parallel data processing. The main significant issue in practical MapReduce application is the data skew. Data skew refers to the imbalance in the amount of the data assigned to each tasks or the imbalance in the amount of work required to process such data. Some of the MapReduce applications like Inverted Index, Grep and Join exhibit data skew Problems.

LIBHPT, a system does not require any pre-run sampling of the input data or prevent the overlap between the map and the reduce stages. It uses new sampling method which can achieve a highly accurate approximation to the distribution of the intermediate data by sampling only a small fraction of the intermediate data during the normal map processing. It automatically allows the reduce tasks to start copying as soon as the chosen sample map tasks complete. It supports the split of large keys when application semantics permit and maintains the total order of the output data. It inspect the heterogeneity of the computing resources when balancing the load among the reduce tasks appropriately. LIBHPT is relevant to a wide range of applications and is transparent to the users.

The Libhpt Algorithm

It includes the partitioning and sampling algorithm. Main goal is to balance the load across reduce stage tasks. The algorithm consists of three steps:

1. Sample partial map tasks
2. Estimate intermediate data distribution
3. Apply hybrid partition on the data

2.4 Handling Data Skew in Parallel Joins in Shared-Nothing Systems [9]

Parallel processing continues to be important in large complex data warehouses. The data warehouses processing requirements continue to expand in multiple direction dimensions. These include larger volumes, huge number of concurrent users, increasing number of difficult complex queries, and more applications which define complicated logical, semantic, and physical existing data models. Shared nothing parallel database management systems [10] can scale up “horizontally” by adding more nodes. Data skew occurs naturally in many applications. A skewed data in query processing not only slows down its system response time, but also generates hot nodes, which become a bottleneck throttling the overall system performance.

In a shared nothing architecture, virtual processors, responsible for performing the scans, joins, locking, transaction management, and other data management work, are called Parallel Units (PUs).Relations are usually horizontally partitioned across all parallel units which allows the system to exploit the I/O bandwidth of multiple disks by reading and writing them in parallel. Hash partitioning is commonly used to partition relations across all parallel Units. Tuples of a relation are assigned to a parallel units by applying a hash function to their Partitioning Column. This Partitioning Column is one or more attributes from the relation, specified by the user or automatically chosen by the system.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), Volume 5, Special Issue 2, May 2015)
International Conference on Advances in Computer and Communication Engineering (ACCE-2015)

A new join geography called PRPD which means Partial Redistribution & Partial Duplication used to improve the performance and scalability of parallel joins in the presence of data skew in a shared-nothing system, experimental results shows that PRPD significantly speeds up query elapsed time in the presence of data skew and mitigate Redistribution skew which cause as the result of a poorly designed hash function.

2.5. Practical Skew Handling in Parallel Joins [11]

An approach to dealing with skew in parallel joins in database systems and this approach is easily implementable within current parallel DBMS, and performs well on skewed data. A fundamental step underlying approach is an initial pass of sampling the relations to be joined. The resulting set of samples is used in two ways: (1) They are used to predict the level of practical skew occurs in the data, and hence to select the appropriate join algorithm and (2) They are used within the skew handling algorithms to determine the proper mapping of work to processors. Skew can occur whenever hashing is used to parallelize a task.

The skew handling algorithms are

1. *Simple range partitioning* - A basic approach to avoiding redistribution skew and to replace hash partitioning with range partitioning.
2. *Weighted range partitioning*- In this algorithm tuples are redistributed using weighted range partitioning.
3. *Virtual processor partitioning - round robin*- In this algorithm the number of partitions is a multiple of the number of processors. The exact correct number of partitions is a parameter of the algorithm. The partitions are allocated to processors using round robin allocation. It deals with the problem of join product skew.
4. *Virtual processor partitioning - processor scheduling.* This algorithm performs processor scheduling using LPT.

2.6 Efficient outer join data skew handling in parallel DBMS [12]

Large enterprises have been relying on parallel database management systems (PDBMS) to process their ever-increasing data volume and huge complex queries. The PDBMS scalability and performance of a comes from load balancing on all nodes in the system. Skewed part processing will significantly slow down query response time and degrade the overall system performance.

In this case a simple and efficient outer join algorithm called OJSO which means Outer Join Skew Optimization is implemented to improve the performance and scalability of parallel outer joins and it significantly speeds up query elapsed time even in the presence of data skew.

The OJSO algorithm applies to both right outer joins and full outer joins and is repeatedly applied to process more than two outer joins. In implementation the OJSO algorithm handles a single outer join at a time. After join order planning is done, the optimizer goes through all outer joins in a query, analyzes what join columns whose values could be set to NULLs and are used in later outer joins, and then repeatedly calls the OJSO algorithm to execute each outer join with the instruction of whether the results of an outer join should be split based on its previous analysis on join columns.

2.7 A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins [13]

A taxonomy of skew effects is developed, and a new performance model is used to compare the performance of two parallel join algorithms.

Types of Data Skew Effects

A. Intrinsic skew – which occurs when attribute values are not distributed uniformly. Hence, it can also be called attribute value skew (AVS). It is a property of the data. AVS may occur on both single and multiple node systems. A join of two relations with AVS may have greater join selectivity and therefore a larger join product compared to a join of two uniformly distributed relations with the same cardinality.

B. Partition skew – which occurs on parallel implementations when the workload is not balanced between nodes. It can occur only on multiple node systems, and its effects vary between implementations. In particular, some types of partition skew can occur even when the input data are uniformly distributed.

Parallel join algorithms may be divided into four stages: 1.Tuples are retrieved from disk. 2. Selection and projection predicates are applied. 3. Tuples are partitioned and redistributed. 4. Partitions are joined on each node. Partition skew may occur at each stage. Thus, four types of skew may be observed:

1. *Tuple Placement Skew (TPS)* occurs when the initial distribution of tuples varies between partitions. For example, tuples may be partitioned by using clustering attribute which are in user specified ranges.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), Volume 5, Special Issue 2, May 2015)
International Conference on Advances in Computer and Communication Engineering (ACCE-2015)

2. *Selectivity Skew (SS)* occurs when the selectivity of selection predicates varies between nodes. example, A selection predicate that includes a range selection on the partitioning attribute.
3. *Redistribution Skew (RS)* occurs when there is a mismatch between the distribution of join key values in a relation and the distribution expected by the redistribution mechanism.
4. *Join Product Skew (JPS)* occurs when the join selectivity at each node differs.

III. Conclusion

In this paper we explained types of skew and their mitigating techniques in MapReduce applications. MapReduce is a programming model and an associated implementation for processing and generating large datasets which is amenable to a broad variety of real-world application tasks. Mapreduce allows users to specify the computation in terms of a map and a reduce function. In MapReduce Skew happens when there is one node has assigned data to be processed more than others either in Map or Reduce Stage.

REFERENCES

[1] MapReduce: Simplified Data Processing on Large Clusters Jeffre Dean et al

[2] MapReduce: State-of-the-Art and Research Directions Abdelrahman Elsayed et al

[3] T. White, *Hadoop: The Definitive guide*, 1st ed.: O'Reilly Media, 2010.

[4] SkewTune : Mitigating Skew in MapReduce Applications YongChul Kwon1 et al

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In Proc. of the 7th WWW Conf., pages 107,117, 1998.

[6] Skew-Resistant Parallel Processing of Feature-Extracting Scientific User-Defined Functions YongChul Kwon et al

[7] Kwon et. al. Scalable clustering algorithm for N-body simulations in a shared-nothing cluster. Technical Report UW-CSE-09-06-01, Dept. of Comp. Sci., Univ. of Washington,

[8] LIBRA: Lightweight Data Skew Mitigation in MapReduce Qi Chen, Jinyu Yao, and Zhen Xiao, *Senior Member, IEEE*

[9] Handling Data Skew in Parallel Joins in Shared-Nothing Systems Yu Xu Teradata San Diego et al

[10] M. Stonebraker. The case for shared nothing. *IEEE* Database Eng. Bull., 9(1):4–9, 1986.

[11] Practical Skew Handling in Parallel Joins David J. DeWitt,Jeffrey F.et al

[12] Efficient outer join data skew handling in parallel DBMS Yu Xu Teradata,Pekka Kostamaa

[13] A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins Christopher B. Walton, Alfred G. Dale Roy M. Jeneve