International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 6 | ISSN : 2456-3307

An Enormous Inspection of MapReduce Technology

J. Rajesh Khanna
Assistant Professor, Department of CSE, BVRIT, Telangana, India

ABSTRACT

Since, the last three or four years, the field of "big data" has appeared as the new frontier in the wide spectrum of IT-enabled innovations and favorable time allowed by the information revolution. Today, there is a raise necessity to analyses very huge datasets, that have been coined big data, and in need of uniqueness storage and processing infrastructures. MapReduce is a programming model the goal of processing big data in a parallel and distributed manner. In MapReduce, the client describes a map function that processes a key/value pair to procreate a set of intermediate value pairs & key, and a reduce function that merges all intermediate values be associated with the same intermediate key. In this paper, we aimed to demonstrate a close-up view about MapReduce. The MapReduce is a famous framework for data-intensive distributed computing of batch jobs. This is over-simplify fault tolerance, many implementations of MapReduce materialize the overall output of every map and reduce task before it can be consumed. Finally, we also discuss the comparison between RDBMS and MapReduce, and famous scheduling algorithms in this field.

Keywords : Big Data, MapReduce, Scheduling, Processing Layer, Indexing, Data Layout.

I. INTRODUCTION

Today the volume of data being generated globally is increasing at a [1] dramatic rate. The enormous amount of data is rising everywhere due to advances in the [2] Internet and communication technologies and the interests of people using social media, Internet of Things, smart phones, sensor devices, online services and many more. The essential to manage efficiently the exponentially growing dataset is increasing each and every day. For examples, International Data Corporation (IDC) announces that 2.9 ZB (zettabytes) data of the universe were stored during the year of 2012 and this will increase up to 45 ZB by 2020 [3]. Correspondingly, Facebook processes around 550 TB (terabytes) data every day and Twitter generates 9 TB data day-to-day [4, 5]. The larger datasets don't only include a structured form of data but more than 75% of the dataset includes semi-structured, unstructured and raw form of data. The conventional data management tools such as the RDBMS, no longer prove to be adequate in handling this burst in data. This report gives an overview of the new ways to maintain such large datasets [6] by iterating over the MapReduce technique. In this survey paper, we review the background and state-of-the-art the MapReduce. The MapReduce [7] is a distributed computing model proposed by Google. The main purpose of MapReduce is to process volumetric data sets distributed and paralleled. It endows a programming model in which users can specify a map function that processes a key/value pair to originate a set of intermediate key/value pairs, and a detract function that merges all intermediate values related the same intermediate key [8]. The Map-Reduce has become a renowned model for developments in cloud computing. MapReduce is a programming model and an allied implementation for processing and generating spacious datasets that is responsible for a broad variety of real-world tasks[2, 6, 7].

The first section is the introduction of Big Data. The second section comparison between MapReduce and RDBMS. The third section discusses MapReduce. The fourth section about F footstep of MapReduce. In fifth section MapReduce execution. Again sixth section MapReduce framework and its components. The seventh section MapReduce user interfaces and eighth section MapReduce procedures . In the nine sections MapReduce performance and ten sections MapReduce scheduling algorithms. Finally, last section is conclusion.

II. COMPARISON BETWEEN THE RDBMS AND MAPREDUCE

The RDBMS is convenient for an application where data size is limited like it's in GBs, whereas MapReduce convenient for an application where data size is in Exabyte [2]. If the data access pattern is dominated by seeks, it will take longer to read or write huge portions of the datasets than streaming through it, which operates at the transfer rate. On the contrary, for updating a miniature portion of records in a database, a traditional B-Tree works well. If updating the majority of a database, a B-tree is less lower than MapReduce, [9] which uses merge as well as sort to rebuild the database. MapReduce acceptable in an application where the data is written once and read numerous times much the same in your Facebook profile you post your photo once and that picture of your seen by your friends numerous times, whereas RDBMS good for data sets that are incessant updated [10].

The structured data is data that is organized into entities that have a defined format, namely XML documents or database tables that correspond to a distinctive predefines schema. In semi-structured data, is looser, and though there may be a schema, it is frequently ignored, so it may be used as a guide to the structure of the data for instance, a spreadsheet, in which the structure is the grid of the cells, although the cells yourself may hold any form of data. However, unstructured data does not have any particular internal structure, for instance plain text or image data [2, 11]. MapReduce works well on any type of unstructured or semi-structured data, since it is designed to interpret the data at processing time. Eventually the RDBMS scaling is nonlinear, whereas MapReduce is linear.

Table 1. The Compared between RDBMS and MapReduce

	RDBMS	MapReduce
Data Size	Gigabytes	Petabytes/Exabytes
Access	Interactive and batch	Batch
Updates	Read and Write many times	Write once, Read many times
Structure	Static Schema	Dynamic Schema
Integrity	High	Low
Scaling	Nonlinear	Linear

III. THE MAPREDUCE

The MapReduce is a framework for supporting the parallel processing of enormous amounts of unstructured data. In MapReduce incorporate [6] both a model to structure the computation and a runtime to parallelize the computation on a cluster of workstations in a fault-tolerant manner. The MapReduce can be seen as a programming exemplar to write applications that can be decomposed in [12, 13]a phase Map and a phase Reduce. MapReduce is a programming model revolutionary by higher-order functions map and reduce commonly found in functional languages. In the circumstance functional programming, the map function enforces a given function to each element of a given list and return the new list. The reduce function provides all the elements of a list by applying a given function to an element and a partial outcome. In spite of, a lot more than this to make these phases happen easily. Every possible computing node has to access the data it process [13]. To remain on the platform, this may be done in multiple ways. The data may beforehand be directly accessible to the nodes or may need to be scattered or re-balanced. The data are then echeloned in key-value pairs. This part is normally not accounted when assess the performance of a MapReduce application.

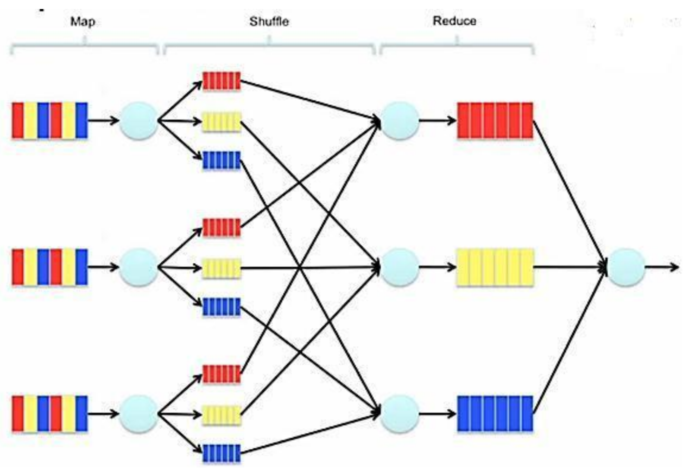

Figure 1. The Normal Workflow of a MapReduce

IV. THE FOOTSTEP OF MAPREDUCE

The input data is automatically isolated by the MapReduce library into pieces, called splits. Each Map task processes a single split, consequently M splits entails M Map tasks. Again Map phase consists in having Mappers read the corresponding splits and production intermediate key/value pairs [1]. Usually, the Mapper uses a subsidiary input reader to read the raw input data. The objective of the reader is to convert the input [13] to a key/value pairs amenable to be processed by the Map function. The MapReduce framework to endorsement a diversification of [6, 12] jobs that are not tied to any specific input format. After the Map phase, a segmentation function is applied over the intermediate key space to divide it into R segmentation, each one to be processed in a Reduce task after that both R and the segmentation function can be configured by the user.

- ***Task Assignment*** **-** The input data are echeloned into splits and, for each split, the master creates a Map task and allocate it to a worker.
- ***Input Reader*** **-** Every Map task executes a function to extract the key/value pairs from the raw data internally the splits.
- ***Map Function*** **-** Every key/value pair is fed into the user-defined Map function that can production zero, one or more intermediate key/value pairs. These pairs represent a temporary outcome, the intermediate data are kept in the local disks of the Mappers.
- ***Shuffle Phase*** **-** The median key/value pairs are assigned to Reducers by means of a segmentation function in a manner such that all median key/value pairs with the same median key will be processed by the same detract task and hence by the same Reducer. These median key/value pairs are spread at random across the cluster, the master passes to Reducers the information about the Mappers's location, because each Reducer may be able to remotely read its input.

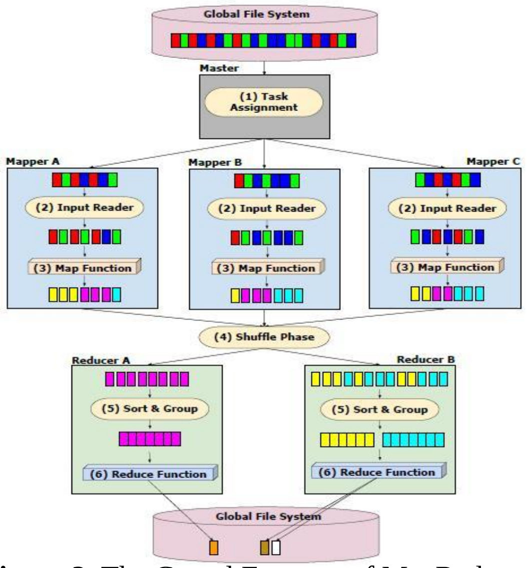

Figure 2. The Genral Footstep of MapReduce

- ***Combiner Phase (Optional)*** **-** If the user-defined detract function is exchangeable and associative, and there is a remarkable repetition of intermediate keys produced by each Mapper, an optional user-defined combiner function can be applied to the outputs of the Mapper [12]. The aim is to group, in pursuance of the intermediate key, all the intermediate values that were produced by the Map tasks of each Mapper. Normally, the same Reduce function code is used for the combiner function. If this combiner function is applied, each Mapper outputs only one median value per median key [13].
- ***Reduce Function*** **-** Every Reducer passes the assigned median keys, and the corresponding set of median values, to the user-defined decrease function. The output of the reducers is stored in the global file system for lastingness.
- ***Sort & Group*** **-** When the remote reads are ended, each Reducer sort the median data in order to group all its input pairs by their median keys.

V. A MAPREDUCE EXECUTION

The Apache Hadoop is the most famous open-source implementation of the MapReduce framework [2]. It is written in Java and has received contributions from big companies such as Facebook and others. The Google's MapReduce, Hadoop also employs two dissimilar layers. The Hadoop MapReduce Framework, a processing layer accountable for running MapReduce jobs and the HDFS, a distributed storage layer accountable for compatible storing the data among the cluster nodes.

A. Processing Layer

The entities associated with the processing layer are one master, named the JobTracker, and one or more workers, named the TaskTrackers. The main contribution of the JobTracker is to coordinate all the jobs running on the system and to allocate tasks to run on the TaskTrackers which periodically report to the JobTracker the headway of their running tasks. The Hadoop's scheduler makes each job [13] use the entire cluster and takes the jobs' priorities into account when scheduling them. This means that topmost priority jobs will run first. In spite of, Hadoop also supports other schedulers, including shared cluster schedulers that allow running jobs from several users at the same time.

In view of task scheduling, Hadoop does not build an a priori plan to set up which tasks are going to run on which nodes instead Hadoop take the plunge on which nodes to deploy each task in runtime. Example for employe nikhat a task, it is assigned the next one [14]. This means that should an employer nikhat all tasks related the data it stores locally, it may be fed tasks which entail acquire data from other workers. There are ten dissimilar places in the query-execution pipeline of Hadoop where User Defined Functions (UDFs) may be injected. A user of the framework can define how the input data is divided and how a divided is parsed into key/value pairs, for example. This side makes Hadoop is comfortably customizable MapReduce framework.

B. Storage Layer

The Hadoop DFS (HDFS) is a distributed file system designed to store unswerving less, i.e., once the data is written into the HDFS is not converted but can be read many times. The HFDS implementation uses three various entities first one NameNode, second one secondary NameNode and thired one or more DataNodes. A NameNode in charge of for storing the metadata of all lies in the distributed file system. In order to recover the metadata files in case of a NameNode lack of success, the SecondaryNameNode keeps a copy of the most recent checkpoint of the lesystem metadata.

Every file in HDFS is echeloned into various fixed-size blocks, similar that each block is stored on any of the DataNodes. Hadoop replicates each block places them strategically in order to make better availability two replicas on DataNodes on the same rack and the third one on a various rack, to inhibit from loss of data, should an entire rack be liquidated. It is worth noting the difference between input divided and HDFS blocks. As long as an HDFS block is an indivisible part of a file that is stored in each node, an input divided is logical data that is processed by a Map task. There is no necessity for divided to be tied to blocks, and not even less.

Think about the case where a file is divided by lines, such that each Map task processes one line. It may come about that a line over us from an HDFS block, i.e., the divided boundaries do not coincide with the HDFS block boundaries. In this situation, the Mapper processing that line must retrieve the next HDFS block to receive the final part of the line.

VI. THE MAPREDUCE FRAMEWORK AND IT'S COMPONENTS

The MapReduce is a programming model and an allied implementation for processing and generating volumetric data sets. Assume that programs written in this functional manner MapReduce are automatically parallelized and executed on a spacious cluster of commodity machines [15, 16]. In a primary MapReduce job, it consists of the following components.

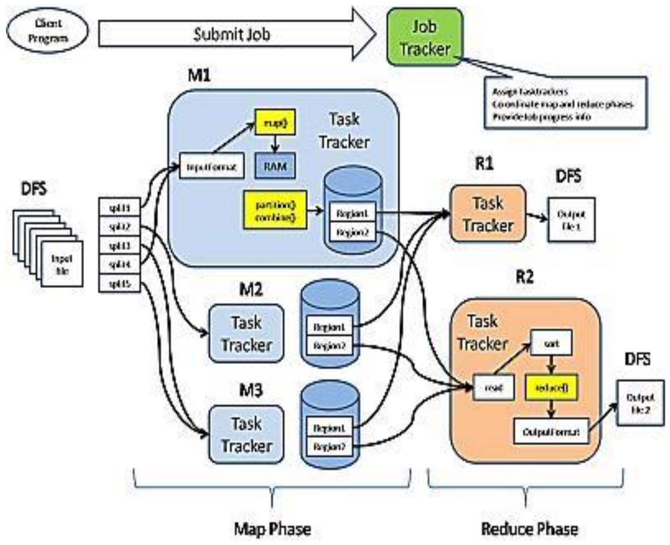

Figure 3. The Architecture & Components of MapReduce

Job Client – This is submits MapReduce jobs to job tracker.
Job Tracker – This is one part of a master node and it allocates job to task tracker.
Task Tracker – This is one part of slave node and it track all task data. As soon as finished the task informed to job tracker.
Pay Load – This is one type of applications, notably designed for MapReduce functions.
Mapper – This is the prime intention of the mapper is mapping the input data to intermediate key/value pairs.
Name Node – This is managing the HDFS Data.
Data Node – This is searched advance data are attendance in processing places.
Master Node – This is the prime intention of Master node is receiving job data from clients.
Slavenode – This is runs Map and Reduce jobs.

VII. THE MAPREDUCE USER INTERFACES

In this section provides some detail about user-facing aspects of the MapReduce framework. This helps you implement, configure, and tune your jobs in a fine-grained way.

Payload - In payload applications normally implement the Mapper and Reducer interfaces to provide the map and reduce methods. These form the core of the job.

Job Configuration - In JobConf describe a MapReduce job configuration. JobConf is the primary interface for a user to mention a MapReduce job to the Hadoop framework for execution. The framework tries to honestly execute the job as mentioned by JobConf.

Task Execution & Environment - The TaskTracker executes the Mapper/ Reducer task as a child process in a distinct jvm. After that child-task inherits the environment of the parent TaskTracker. The user can describe supplementary options to the child-jvm via the mapred.

Job Submission and Monitoring - The Job is the primary interface by which a user job communicates with the Resource Manager. Job provides condescension to submit jobs, track their progress, access component task reports and logs, and get MapReduce cluster status details.

Job Input - In the input format represents the input identification for a MapReduce job. The MapReduce framework relies on the input format of the job too. Again validate the input specification of the job. After that divide the input file into logical input, divide instances, each of which is then assigned to a personal Mapper. Again the RecordReader implementation used to gather input records from the logical input, divide for processing by the Mapper.

Job Output - In the output format represents the output identification for a MapReduce job. The MapReduce framework relies on the output format of the job too. Again validate the output specification of the job for example, inspection that the output directory does not previously alive.

VIII. THE MAPREDUCE PROCEDURES

In this paper, we are discuss about two different MapReduce procedures firstly the single-job procedure, in which subdivide of work to tasks is done without taking document sizes into account and secondly the chain-job procedure, in which a first job downloads the documents (take the plunge their sizes) and performs some preparatory entity-mining, while a second job (chained with the first) sustain the mining over size-aware division of the contents to produce the accomplished NEM analysis.

A. Single Job Procedure

In the single-job procedure includes an initial stage that queries a search engine to receive the hits to be processed and prefabricate the distribution of tasks, followed by a subsequent stage of the MapReduce job itself, both shown in figure 4. Here master node (where the JobTracker executes) carries out preliminary processing. First, it queries a Web Search Engine, which comeback a set of titles, URLs, and snippets [17] etc. Subsequently, the master attempt to determine the URL content length, in order to preferable balance the downloading and processing of the URL contents in the MapReduce job. Again instate this is to perform an HTTP HEAD request for each URL earlier to downloading it. These processes again and again pops the top of the stack and inserts it to the divide with the least total size, until the stack is empty. When the assignment of URLs is finished the produced divide are stored in HDFS. At the subsequent stage of the single-job procedure a number of mapper tasks are indite on a number of JVMs hosted by Cloud VMs.

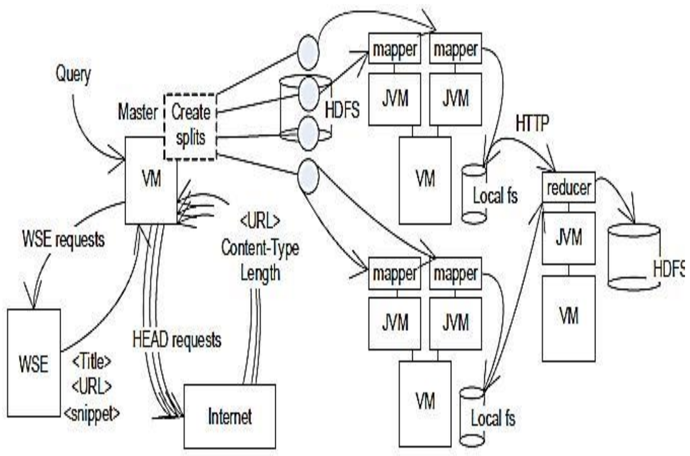

Figure 4. Single-job design

B. Chain Job Procedure

We have evolved a substitute MapReduce procedure that consists of two chained jobs (Jobs #1 and #2) as shown in figure 5. The appropriateness behind this design is the following Job #1 downloads the exhaustive document set and thus gains exact information [18] about content sizes. Here upon Job #2 is now capable of performing a size-aware assignment of the remaining documents to tasks. At the same moment, we have confidence that most users panegyrize a quick NEM preview on a sample of the hits before getting the full-scale analysis. In Job #1 is designed to carry out such a preview the master node queries the search engine getting the beginning set of titles, URLs, and snippets. Then, it creates the beginning divided of the URLs without using any information about their sizes. Eventual after Job#1 tasks in the first instance downloading documents while performing only limited-scale NEM analysis, there is no requirement to create much tasks than the number of JVM slots approachable.

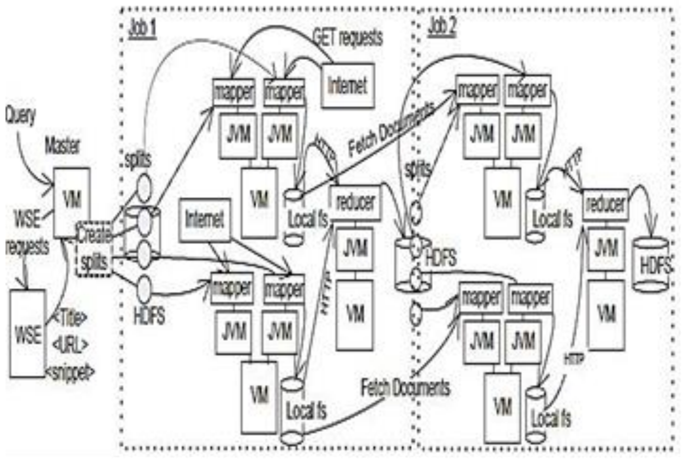

Figure 5. Chain-job design

IX. THE MAPREDUCE PERFORMANCE

The MapReduce may not endow the desired performance in inappreciable use cases, such as the selection scenario. In this section we describe the most pertinent of these techniques, like that indexing, data layouts, and co-location, which purpose at improving data accesses during the Map phase of selective query process.

A. Indexing

In a plain MapReduce framework, the execution of a job need that a full scan of the dataset is performed (example for reading all the records one by one) further if only a small subset of them is going to be selected. Additionally, all partitioned generate Map tasks even if some of them do not contain pertinent data and will not create any intermediate data. As well, in the Map phase, all blocks have to be perfectly

fetched from disk, which insinuate that all attributes of all records are fetched to main memory without taking into account which ones the job is fascinated in [19]. This issue can keep away from if the input data are indexed before the mapping step takes place. Momentarily, an index is built for one attribute and consists in a list of various values of that attribute contained in the indexed data, as well as possible the positions where these values can be found in the input file.

- ***Record Level Indexing***

Commonly, this access requires having the index inside each partition, which is then used at query time to select the applicable records from the partition, instead of reading the entire partition record by record [20].

- ***Split Level Indexing***

It has been attention that the execution time of a query depends on the number of waves of Map tasks that are executed, which in turn depends on the number of processed partition [21]. Additionally, the time to process a partition is dominated by the I/O time for reading its dataplus the overhead of commencement the Map task to process it.

- ***Block Level Indexing***

In this case where the partition is comprised of several blocks, one can create block-level indexes. All blocks contain several records, these indexes have the potential to bring meaningful time savings, in pursuance of to the monitoring [21] block-level indexes map attributes values to blocks that contain at least one record with that attributes value. By means of these indexes, when processing a partition, the applicable blocks are loaded and processed, whereas the blocks familiar not to match the selection specification are omitted.

B. Data Layout

The storage unit of HDFS consists in a fixed-size (generically 64MB to 96MB) block of data. Actually, the way the data are arranged within this block of data can much affect the repercussion times of the system. Now we call this inner organization of blocks the data layout. We now familiarize four various data layouts that may be applied on MapReduce.

Row Oriented

When utilization this layout all fields of one record are stored sequentially in the file, and several records are placed contiguously in the disk blocks. This layout endows rapidly data loading in many cases, the data to be stored is endow row-by-row, that means one entire record at a time. In Row-based systems are designed to adroitly process queries where several columns of a record required to be processed at the same time, the entire row can be brought back with a single access to the disk.

Column Oriented

When utilization this layout, datasets are vertically split by column, each column is stored independently and accessed only when the corresponding attribute is required. In this layout is well favorable for read operations that accessed a mini number of columns, since columns that are not applicable for the intended output are not read. As an outcome, record reconstruction may need network transfers to access the needed columns from various storage nodes.

Column Groups

In this layout consists in organizing all columns of a dataset in various groups of columns. The various column groups may have overlapping columns. The data within each column group is not connected to any particular data layout. It seems stored in a row or column-oriented way. A benefit of column groups is that it can keep away from the overhead of record reconstruction if a query desire an existing amalgamation of columns.

PAX

In this hybrid layout combines the [22] benefit of both row-oriented and column-oriented layouts. Every field of a record is on the same block as the row-oriented layout, make provision for low cost of record reconstruction. In spite of, within each disk page containing the block, PAX uses mini pages to group every values affiliation to each column.

X. MAPREDUCE SCHEDULING ALGORITHMS

In this section, we describe briefly scheduling algorithms is one of the most critical dimensions of MapReduce. There are many algorithms to address these matters with different techniques and outlook. Now a few of them get attention to improve data locality and few of them implements to provide synchronization processing.

A. Fair Scheduling Algorithm

The Fair scheduling is a method of assigning resources to jobs such that all jobs get on usual, an identical share of resources over time. Meanwhile a single job running, that job uses the entire cluster. When additional jobs are submitted tasks slots that free up are assigned to the recent jobs, because each job gets approximately the same amount of CPU time. Dissimilar the lapse Hadoop scheduler, which forms a queue of jobs, this lets, short jobs finish in pertinent time while not starving long jobs. It is also a simple way to [23] share a cluster between several of the others. The Fair Scheduler algorithm can range the number of concurrent running jobs [24] per user and per pool. This can be advantageous when a user must submit hundreds of jobs at one time, or to make sure that intermediate data does not fill up disk space on a cluster when too many simultaneous jobs are running.

B. Hadoop On Demand (HOD) Scheduling Algorithm

In this Hadoop On Demand (HOD) Scheduling Algorithm is a system for provisioning and managing unconstrained Hadoop MapReduce and Hadoop Distributed File System (HDFS) example on a shared cluster of nodes. These algorithms make it convenient for administrators and users to quickly setup and use Hadoop. The HOD is also a very advantageous tool for Hadoop developers and testers who requirement to share a physical cluster for testing their own Hadoop versions [23]. The HOD utilization the torque resource manager to do node allocation. In the allocated nodes, it can start Hadoop MapReduce and HDFS daemons. It automatically originates the convenient configuration files (Hadoop-site.xml) for the Hadoop daemons and client. This algorithm also has the ability to distribute Hadoop to the nodes in the virtual cluster that it allocates.

C. Capacity Scheduler Algorithm

In this algorithm contrive to run Hadoop Map-Reduce as a shared, multi-tenant cluster in an operational affable manner while maximizing the throughput and the usage of the cluster while running Map-Reduce applications. This algorithm to allow sharing a huge cluster while giving every company a minimum capacity promise. The central idea is that the available resources [25, 23] in the Hadoop Map-Reduce cluster segregates among multiple organizations who collectively fund the cluster based on computing necessity. There is an added gain that a company can access any excess competence now being used by others. This endow elasticity for the company in a cost-effective manner.

D. Center-of-Gravity Reduce Scheduler Algorithm

The Center-of-Gravity reduce scheduler algorithm to work designing a locality-aware, skew-aware alleviate task scheduler for reduction MapReduce network traffic [26]. The proposed scheduler algorithm attempts to schedule every alleviate the task at its center-of-gravity node laid down [27] by the network locations. Regarding scheduling reducers at their center-of-gravity nodes, they argue for reduce network traffic, which may possibly allow more MapReduce jobs to stick together on the same system [28].

XI. CONCLUSION

In the real world, data processing and storage approaches are facing many challenges in meeting the persistently increasing insistence of big data. Thus, it becomes greatly challenging to operate on the enormous data. In the Big data area, MapReduce is one of the key approaches used for nursing giant data sets. MapReduce was introduced to extricate volumetric data, computational problems, and in particular designed to run on commodity hardware and its based on divide and conquer principles. This is because of its appreciable flexibility, which allows automatic parallelization and execution on a huge-scale cluster with more than thousands of nodes. In this paper, presents MapReduce concepts, footstep of MapReduce, execution of MapReduce, MapReduce framework and its components, we also highlight the MapReduce user interfaces. Finally, we investigated the procedures, performance, scheduling algorithms in MapReduce. These surveys aim to provide an extensive overview and big-picture to readers of this exciting area.

XII. REFERENCES

[1]. Kim, G.-H., Trimi, S., & Chung, J.-H. (2014). Big-data applications in the government sector. Communicationsof the ACM, 57(3), pp 78–85.

[2]. Dr. Yusuf Perwej, "An Experiential Study of the Big Data," for published in the International Transaction of Electrical and Computer Engineers System (ITECES), USA, ISSN (Print): 2373-1273 ISSN (Online): 2373-1281, Vol. 4, No. 1, page 14-25, March 2017, DOI:10.12691/iteces-4-1-3.

[3]. R. Murugesh, I. Meenatchi, "A Study Using PI on: Sorting Structured Big Data In Distributed Environment Using Apache Hadoop MapReduce", International Journal of Computer Sciences and Engineering, Vol.2, Issue.8, pp.35-38, 2014.

[4]. "Apache Hadoop," Apache. Online]. Available: http://hadoop.apache.org/. Accessed: 18-Feb-2015].

[5]. M. Khan, P. M. Ashton, M. Li, G. A. Taylor, I. Pisica, and J. Liu, "Parallel Detrended Fluctuation Analysis for Fast Event Detection on Massive PMU Data," Smart Grid, IEEE Trans., vol. 6, no. 1, pp. 360–368, Jan. 2015.

[6]. K. Parimala1 G. Rajkumar, A. Ruba, S. Vijayalakshmi, "Challenges and Opportunities with Big Data", International Journal of Scientific Research in Computer Science and Engineering, Vol.5, Issue.5, pp.16-20, 2017.

[7]. Lee, D., Kim, J.-S., & Maeng, S. "Large-scale incremental processing with MapReduce", Future Generation Computer Systems, 36, pp 66–79, (2014), doi:10.1016/j.future.2013.09.010.

[8]. M. Khan, M. Li, P. Ashton, G. Taylor, and J. Liu, "Big data analytics on PMU measurements," in Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference on, 2014, pp. pp 715–719.

[9]. Qi, C., Cheng, L., & Zhen, X. (2014). Improving mapreduce performance using smart speculative execution strategy. IEEE Transactions on Computers, Vol. 63(4), pp 954–967. Doi:10.1109/TC.2013.15.

[10]. J. Kwon, K. Park, D. Lee, S. Lee, PSR: Pre-computing Solutions in RDBMS for Fast Web services Composition Search, in: Proceedings of the 2nd International Conference on Web Services, Salt Lake City, Utah, USA, ICWS 2007, pp. 808-815.