Mike Gualtieri, Forrester Analyst, proposes a definition that attempts to be pragmatic and actionable for IT professionals:

"Big Data is the frontier of a firm's ability to store, process, and access (SPA) all the data it needs to operate effectively, make decisions, reduce risks, and serve customers" (Gualtieri, December 2012).

A. Vs of Big Data

- **Volume of data:** Volume refers to amount of data. Volume of data stored in enterprise repositories have grown from megabytes and gigabytes to petabytes.
- **Variety of data:** Different types of data and sources of data. Data variety exploded from structured and legacy data stored in enterprise repositories to unstructured, semi structured, audio, video, XML etc.
- **Velocity of data:**Velocity refers to the speed of data processing. For time-sensitive processes such as catching fraud, big data must be used as it streams into your enterprise in order to maximize its value.

B. Comparison between Traditional DBMS and BigData

MapReduce is complementary to DBMS [26], not a competing technology.

i. Parallel DBMS are for efficient querying of large data sets.
ii. MR-style systems are for complex analytics and ETL tasks.
iii. Parallel DBMS require data to fit into the relational paradigm of rows and columns.
iv. In contrast, the MR model does not require that data files adhere to a schema defined using the relational data model. That is, the MR programmer is free to structure their data in any manner or even to have no structure at all.

C. BigData Pillars

1) Big Table – Relational, Tabular format – rows & columns
2) Big Text – All kinds of unstructured data, natural language, grammatical data, semantic data
3) Big Metadata – Data about data, taxonomies, glossaries, facets, concepts, entity
4) Big Graphs – object connections, semantic discovery, degree of separation, linguistic analytic, subject predicate object

D. Big Data: Infrastructure Requirements

i. Data Acquisition in Big Data

Even though the data will be in distributed environment, infrastructure must support to carry out very high transaction volumes and also support flexible data structures. To collect and store data, NoSQL are often used in Big data. NoSQL will not have any fixed schema since it supports high variety of data by capturing all types of data. Keys are used to identify the data point without designing schema with relationship between entities.

ii. Data Organization in Big Data

In the classical term of data warehousing, organizing data is called as data integration. Big data requires good infrastructure, so that processing and manipulating data in the original storage location can be done easily. It must also supports very high throughput to deal with processing steps of large data and handles large variety of data formats like structured format, unstructured format etc. Hadoop [10] [28] is a new technology that allows large data volumes to be organized and processed while keeping the data on the original data storage cluster. For example Hadoop Distributed File System (HDFS) [9], [10] is the long - term storage system for web logs. These web logs are turned into browsing behavior (sessions) by running MapReduce programs on the cluster and generating aggregated results on the same cluster [19]. These aggregated results are then loaded into a Relational DBMS system.

iii. Data analysis in Big Data

Since data is not always moved during the organization phase, the analysis may also be done in a distributed environment, where some data will stay where it was originally stored and be transparently accessed from a data warehouse. The infrastructure required for analyzing big data must be able to support deeper analytics such as statistical analysis and data mining, on a wider variety of data types stored in diverse systems; scale to extreme data volumes; deliver faster response times driven by changes in behavior; and automate decisions based on analytical models. Most importantly, the infrastructure must be able to integrate analysis on the combination of big data and traditional enterprise data. New insight comes not just from analyzing new data, but from analyzing it within the context of the old to provide new perspectives on old problems. For example, analyzing inventory data from a smart vending machine in combination with the events calendar for the venue in which the vending machine is located, will dictate the optimal product mix and replenishment schedule for the vending machine.

3. HADOOP

A. Introduction To Hadoop

Hadoop has been successfully used by many companies including AOL, Amazon, Facebook, Yahoo and New York Times for running their applications on clusters. For example, AOL used it for running an application that analyzes the behavioral pattern of their users so as to offer targeted services. Apache Hadoop [2] is an open source implementation of the Google's MapReduce parallel processing framework.

Hadoop hides the details of parallel processing, including data distribution to processing nodes, restarting failed subtasks, and consolidation of results after computation. This framework allows developers to write parallel processing programs that focus on their computation problem, rather than parallelization issues. Hadoop includes 1) Hadoop Distributed File System (HDFS) [9] [10] [25]: a distributed file system that store large amount of data with high throughput access to data on clusters and 2) Hadoop Map Reduce: a software framework for distributed processing of data on clusters.

A .1 HDFS- Distributed file system

Google File System (GFS) [9] [25] is a proprietary distributed file system developed by Google and specially designed to provide efficient, reliable access to data using large clusters of commodity servers. Files are divided into chunks of 64 MB, and are usually appended to or read and only extremely rarely overwritten or shrunk. Compared with traditional file systems, GFS is designed and optimized to run on data centers to provide extremely high data throughputs, low latency and survive individual server failures. Inspired by GFS, the open source Hadoop Distributed File System (HDFS) stores large files across multiple machines. It achieves reliability by replicating the data across multiple servers. Similarly to GFS, multiple replicas of data are stored on multiple compute nodes to provide reliable and rapid computations. Data is also provided over HTTP, allowing access to all content from a web browser or other types of clients. HDFS has master/slave architecture.

As shown in figure A.1, HDFS Architecture [9] consists of a single NameNode and multiple DataNodes in a cluster. NameNode is responsible for mapping of data blocks to DataNodes and for managing file system operations like opening, closing and renaming files and directories. Upon the instructions of NameNode, DataNodes perform block creation, deletion and replication of data blocks. The NameNode also maintains the file system namespace which records the creation, deletion and modification of files by the users. NameNode decides about replication of data blocks. In a typical HDFS, block size is 64MB and replication factor is 3 (second copy on the local rack and third on the remote rack).

A .2 Hadoop MapReduce

As shown in figure A.2, Hadoop MapReduce Architecture [2][9][10] is one of the parallel data processing paradigm designed for large scale data processing on cluster-based computing architectures. It was originally proposed by Google to handle large-scale web search applications. This approach has been proved to be an effective programming approach for developing machine learning, data mining, and search applications in data centers. Its advantage is that it allows programmers to abstract from the issues of scheduling [26], parallelization, partitioning, replicationand focus on developing their applications.

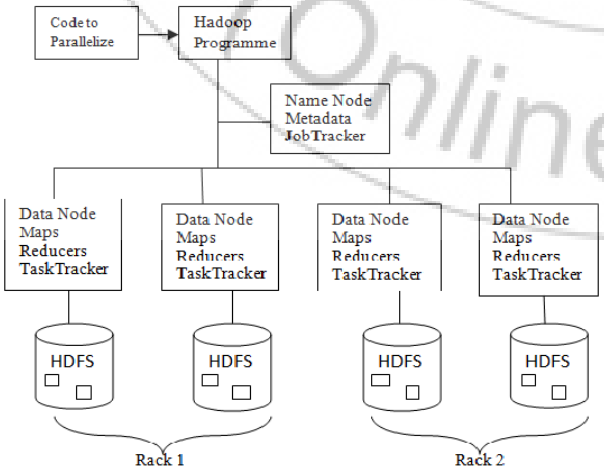

Figure A.1: Hadoop Distributed File System Architecture

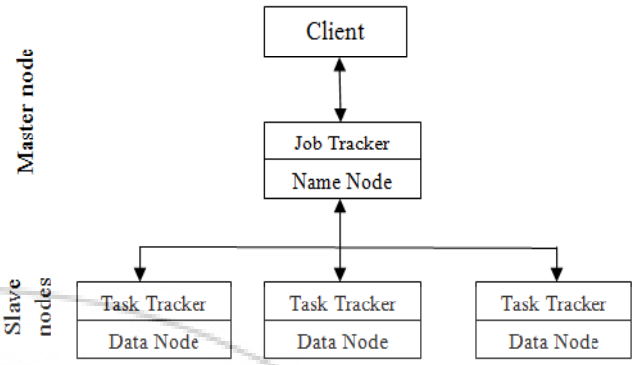

Figure A .2: Hadoop MapReduce Architecture

Hadoop MapReduce programming model consists of data processing functions: Map and Reduce [5][6][7]. Parallel Map tasks are run on input data which is partitioned into fixed sized blocks and produce intermediate output as a collection of <key, value> pairs. These pairs are shuffled across different reduce tasks based on <key, value>pairs. Each Reduce task accepts only one key at a time and process data for that key and outputs the results as <key, value> pairs. The Hadoop MapReduce architecture consists of one JobTracker (Master) and many TaskTrackers (Workers). The JobTracker receives job submitted from user, breaks it down into map and reduce tasks, assigns the tasks to Task Trackers, monitors the progress of the Task Trackers, and finally when all the tasks are complete, reports the user about the job completion. Each Task Tracker has a fixed number of map and reduce task slots that determine how many map and reduce tasks it can run at a time. HDFS supports reliability and fault tolerance of MapReduce computation by storing and replicating the inputs and outputs of a Hadoop job.

4. Scheduling in Hadoop

4.1 Scheduling

The default scheduling algorithm is based on FIFO where jobs were executed in the order of their submission. Later on the ability to set the priority of a Job was added. Facebook and Yahoo contributed significant work in developing schedulers i.e. Fair Scheduler and Capacity Scheduler respectively which subsequently released to Hadoop Community. Research work is being taking place in scheduling a job in Hadoop [29] [31] [26]. Some of the researchers have developed scheduling algorithms which are also discussed in this section.

i. Schedul Fifo er

This is a default scheduler which operates using a FIFO queue. A job is first partitioned into individual tasks, and then loaded into the queue and assigned to free slots on TaskTracker nodes. Each job would use the whole cluster, so jobs had to wait for their turn. Even though a shared cluster offers great potential for offering large resources to many users, the problem of sharing resources fairly between users requires a better scheduler. Production jobs need to complete in a timely manner, while allowing users who are making smaller ad hoc queries to get results back in a reasonable time.

ii. Fair Scheduler

The Fair Scheduler was developed at Facebook to manage access to their Hadoop cluster [3]. The Fair Scheduler [12][23] aims to give every usera fair share of the cluster capacity over time. Users may assign jobs to pools, with each pool allocated a guaranteed minimum number of Map and Reduce slots [7] [14]. Free slots in idle pools may be allocated to other pools, while excess capacity within a pool is shared among jobs. The Fair Scheduler supports preemption, so if a pool has not received its fair share for a certain period of time, then the scheduler will kill tasks in pools running over capacity in order to give the slots to the pool running under capacity. As jobs have their tasks allocated to Task Tracker slots for computation, the scheduler tracks the deficit between the amount of time actually used and the ideal fair allocation for that job. As slots become available for scheduling, the next task from the job with the highest time deficit is assigned to the next free slot. Over time, this has the effect of ensuring that jobs receive roughly equal amounts of resources. Shorter jobs are allocated sufficient resources to finish quickly. At the same time, longer jobs are guaranteed to not be starved of resources.

iii. Capacity Scheduler

Capacity Scheduler [11] originally developed at Yahoo addresses a usage scenario where the number of users is large, and there is a need to ensure a fair allocation of computation resources amongst users. The Capacity Scheduler allocates jobs based on the submitting user to queues with configurable numbers of Map and Reduce slots [6] [16]. Queues that contain jobs are given their configured capacity, while free capacity in a queue is shared among other queues. Within a queue, scheduling operates on a modified priority queue basis with specific user limits, with priorities adjusted based on the time a job was submitted, and the priority setting allocated to that user and class of job. When a Task Tracker slot becomes free, the queue with the lowest load is chosen, from which the oldest remaining job is chosen. A task is then scheduled from that job. Overall, this has the effect of enforcing cluster capacity sharing among users, rather than among jobs, as was the case in the Fair Scheduler.

iv. Longest Approximate Time to End (LATE) - Speculative Execution

It is not uncommon for a particular task to continue to progress slowly. This may be due to several reasons like– high CPU load on the node, slow background processes etc. All tasks should befinished for completion of the entire job. The scheduler tries to detect a slow running task to launch another equivalent task as a backup which is termed as speculative execution of tasks. If the backup copy completes faster, the overall job performance is improved. Speculative execution is an optimization but not a feature to ensure reliability of jobs. If bugs cause a task to hang or slow down then speculative execution is not a solution, since the same bugs are likely to affect the speculative task also. Bugs should be fixed so that the task doesn't hang or slow down. The default implementation of speculative execution relies implicitly on certain assumptions: a) Uniform Task progress on nodes b) Uniform computation at all nodes. That is, default implementation of speculative execution works well on homogeneous clusters. These assumptions break down very easily in the heterogeneous clusters that are found in real-world production scenarios. Matei Zaharia, proposed a modified version of speculative execution called Longest Approximate Time to End (LATE) algorithm that uses a different metric to schedule tasks for speculative execution. Instead of considering the progress made by a task so far, they compute the estimated time remaining, which gives a more clear assessment of a straggling tasks' impact on the overall job response time. They demonstrated significant improvements by Longest Approximate Time to End (LATE) algorithm over the default speculative execution.

v. Delay Scheduling

Matei Zaharia, Dhruba Borthakur, have discussed delay scheduler [21] [22]. Fair scheduler is developed to allocate fair share of capacity to all the users. Two locality problems identified when fair sharing is followed are – head-of-line scheduling and sticky slots. The first locality problem occurs in small jobs (jobs that have small input files and hence have a small number of data blocks to read). The problem is that whenever a job reaches the head of the sorted list for scheduling, one of its tasks is launched on the next slot that becomes free irrespective of which node this slot is on. If the head-of-line job is small, it is unlikely to have data locally on the node that is given to it. Head-of-line scheduling problem was observed at Facebook in a version of HFS without delay scheduling. The other locality problem, sticky slots, is that there is a tendency for a job to be assigned the same slot repeatedly. The problems aroused because following a strict queuing order forces a job with no local data to be scheduled.

To overcome the Head of line problem, scheduler launches a task from a job on a node without local data to maintain fairness, but violates the main objective of MapReduce that schedule tasks near their input data. Running on a node that contains the data (node locality) is most efficient, but when this is not possible, running on a node on the same rack (rack locality) is faster than running off-rack. Delay scheduling is a solution that temporarily relaxes fairness to improve locality by asking jobs to wait for a scheduling opportunity on a node with local data. When a node requests a task, if the head-of-line job cannot launch a local task, it is skipped and looked at subsequent jobs. However, if a job has been skipped long enough, non-local tasks are allowed to launch to avoid starvation. The key insight behind delay scheduling is that although the first slot we consider giving to a job is unlikely to have data for it, tasks finish so quickly that some slot with data for it will free up in the next few seconds.

vi. Dynamic Priority Scheduling

Thomas Sandholm*et* [27], proposed Dynamic Priority Scheduler that supports capacity distribution dynamically among concurrent users based on priorities of the users. Automated capacity allocation and redistribution is supported in a regulated task slot resource market. This approach allows users to get Map or Reduce slot on a proportional share basis per time unit. These time slots can be configured and called as allocation interval. It is typically set to somewhere between 25 seconds and 1 minute. For example a max capacity of 28 Map slots gets allocated

proportionally to three users. The central scheduler contains a Dynamic Priority Allocator and a Priority Enforcer component responsible for accounting and schedule enforcement respectively. This model appears to favor users with small jobs than users with bigger jobs. However Hadoop MapReduce supports scaling down of big jobs to small jobs to make sure that fewer concurrent tasks runs by consuming the same amount of resources [14][16].

To avoid starvation, queue blocking and to respond to user demand fluctuations more quickly preemption is also supported. In this mechanism task slots that were allocated may be preempted and allocated to other users if they were not used for long time. As a result of variable pricing mechanism users to get guaranteed slot during demand periods has to pay more. This scheme discourages the free-riding and gaming by users. However, the Hadoop MapReduce scheduling framework allows jobs to be split up in finer grained tasks that can run and possibly fail and recover independently. So the only thing the end users would need to worry about is to get a good enough average capacity over some time to meet their deadlines. This introduces the difficulty of making spending rate decisions to meet the SLA and deadline requirements. Possible starvation of low-priority (low-spending) tasks can be mitigated by using the standard approach in Hadoop of limiting the time each task is allowed to run on a node. Moreover, this new mechanism also allows administrators to set budgets for different users and let them individually decide whether the current price of preempting running tasks is within their budget or if they should wait until the current users run out of their budget. The fact that Hadoop uses task and slot level scheduling and allocation as opposed to job level scheduling also avoids many starvation scenarios. If there is no contention, i.e. there are enough slots available to run all tasks from all jobs submitted, the cost for excess resources essentially becomes free because of the work conserving principle of this scheduler. However, the guarantees of maintaining these excess resources are reduced. To see why, consider new users deciding whether to submit jobs or not. If they see that the price is high they may wait to preempt currently running jobs, but if the resources are essentially given out for free they are likely to lay claim on as many resources they can immediately. We note that the Dynamic Priority scheduler can easily be configured to mimic the behavior of the other schedulers. If no queues or users have any credits left the scheduler reduces to a FIFO scheduler. If all queues are configured with the same share (spending rate in our case) and the allocation interval is set to a very large value the scheduler reduces to the behavior of the static fair-share schedulers.

vii. Deadline Constraint Scheduler

Deadline Constraint Scheduler [17] addresses the issue of deadlines but focuses more on increasing system utilization. Dealing with deadline requirements in Hadoop-based data processing is done by (1) a job execution cost model that considers various parameters like map and reduce runtimes [7], input data sizes, data distribution, etc., (2) a Constraint-Based Hadoop Scheduler that takes user deadlines as part of its input. Estimation model determines the available slot based a set of assumptions:

i. All nodes are homogeneous nodes and unit cost of processing for each map or reduce node is equal
ii. Input data is distributed uniform manner such that each reduce node gets equal amount of reduce data to process
iii. Reduce tasks starts after all map tasks have completed;
iv. The input data is already available in HDFS.

Schedulability of a job is determined based on the proposed job execution cost model independent of the number of jobs running in the cluster. Jobs are only scheduled if specified deadlines can be met. After a job is submitted, schedulability test is performed to determine whether the job can be finished within the specified deadline or not. Free slots availability is computed at the given time or in the future irrespective of all the jobs running in the system. The job is enlisted for scheduling after it is determined that the job can be completed within the given deadline. A job is schedulable if the minimum number of tasks for both map and reduce [8] [14] is less than or equal to the available slots. This Scheduler shows that when a deadline for job is different, then the scheduler assigns different number of tasks to TaskTrackerand makes sure that the specified deadline is met.

viii. Resource Aware Scheduling

Resource Aware Scheduling [20] in Hadoop has become one of the Research Challenges in Cloud Computing [3]. Scheduling in Hadoop is centralized, and worker initiated. Scheduling decisions are taken by a master node, called the JobTracker, whereas the worker nodes, called TaskTrackers are responsible for task execution. The JobTracker maintains a queue of currently running jobs, states of TaskTrackers in a cluster, and list of tasks allocated to each TaskTracker. Each Task Tracker node is currently configured with a maximum number of available computation slots. Although this can be configured on a per-node basis to reflect the actual processing power and disk channel speed, etc available on cluster machines, there is noonline modification of this slot capacity available. That is, there is no way to reduce congestion on a machine by advertising a reduced capacity. In this mechanism, each Task Tracker node monitors resources such as CPU utilization, disk channel IO in bytes/s, and the number of page faults per unit time for the memory subsystem. Although we anticipate that other metrics will prove useful, we propose these as the basic three resources that must be tracked at all times to improve the load balancing on cluster machines. In particular, disk channel loading can significantly impact the data loading and writing portion of Map and Reduce tasks, more so than the amount of free space available. Likewise, the inherent opacity of a machine's virtual memory management state means that monitoring page faults and virtual memory-induced disk thrashing is a more useful indicator of machine load than simply tracking free memory.

5. Conclusion

Now days Big Data (Hadoop) is in huge demand in the market. There huge amount of data is lying in the industry. Hadoop can be implemented and used on large number of dataset. In Hadoop MapReduce is the most important component. In this paper we have studied many techniques for making the efficient scheduler for the map reduce so that

we can speed up our system or data retrieval technique like quincy, Asynchronous Processing, Speculative Execution, Job Awareness, Delay Scheduling, Copy Compute Splitting etc had made the scheduler effective for the faster processing. There are several research avenues in scheduling of Hadoop MapReduce for fast and efficient processing of the job. Future work includes developing Hadoop job schedulers in terms of meeting workflow deadlines, and scales up to tens of thousands of concurrent workflows.

References

[1] Andrew Pavlo, "A Comparison of Approaches to Large-Scale Data Analysis", SIGMOD, 2009.
[2] Apache Hadoop: http://Hadoop.apache.org
[3] B. Thirmala Rao, N. V. Sridevei, V. Krishna Reddy, LSS.Reddy, "Performance Issues of Heterogeneous Hadoop Clusters in Cloud Computing", Global Journal Computer Science & Technology Vol. 11, no. 8, May 2011,pp.81-87
[4] B. Thirumala Rao, Associate Professor Dept. of CSE Lakireddy Bali Reddy College of Engineering Dr. L. S. S. Reddy, Professor & Director Dept. of CSE Lakireddy Bali Reddy College of Engineering, "Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments".
[5] Chen He Ying Lu David Swanson, "Matchmaking: A New MapReduce Scheduling Technique", Department of Computer Science and Engineering, University of Nebraska-Lincoln Lincoln, U.S.
[6] Dean, J. and Ghemawat, S., "MapReduce: a flexible data processing tool", ACM 2010.
[7] DeWitt & Stonebraker, "MapReduce: A major step backwards", 2008.
[8] Dongjin Yoo, Kwang Mong Sim, "A Comparative review of job scheduling for MapReduce,", Multi-Agent and Cloud Computing Systems Laboratory, School of Information and Communication, Gwangju Institute of Science and Technology (GIST), Gwangju, Republic of Korea.
[9] Hadoop Distributed File System, http://hadoop.apache.org/hdfs
[10] Hadoop Tutorial: http://developer.yahoo.com/hadoop/tutorial/module1.html
[11] Hadoop's Capacity Scheduler: http://hadoop.apache.org/core/docs/current/capacity_scheduler
[12] Hadoop's Fair Scheduler http://hadoop.apache.org/common/docs/r0.20.2/fair_schedu ler.html
[13] J. Dean and S. Ghemawat, "Data Processing on Large Cluster", OSDI '04, pages 137–150, 2004
[14] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", p.10, (2004).
[15] Jean-Pierre Dijcks, "Oracle: Big Data for the Enterprise", 2013.
[16] Joel Wolf IBM T.J. Watson Research Hawthorne, NY 10532 jlwolf@us.ibm.com; Andrey Balmin, IBM Almaden Research; San Jose, CA 95120 abalmin@us.ibm.com; Deepak Rajan, Lawrence Livermore Labs Livermore, CA 94550, rdeepak@gmail.com; RaresVernica, Hewlett-Packard Laboratories Palo Alto, CA 94304 rares.vernica@hp.com; "CIRCUMFLEX: A Scheduling Optimizer for MapReduce Workloads With Shared Scans"
[17] K. Kc and K. Anyanwu, "Scheduling Hadoop Jobs to Meet Deadlines", in Proc. CloudCom, 2010, pp.388-392.
[18] M. Tim Jones, Micah Nelson, "Moving ahead with Hadoop YARN: An introduction to Yet Another Resource Negotiator", 2013.
[19] M. Zaharia, A. Konwinski, A. Joseph, Y. Zatz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments. In OSDI", 8th USENIX Symposium on Operating Systems Design and Implementation, October 2008
[20] Mark Yong, Nitin Garegrat, Shiwali Mohan: "Towards a Resource Aware Scheduler in Hadoop" in Proc. ICWS, 2009, pp:102-109
[21] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling in EuroSys 10", Proceedings of the 5th European conference on Computer systems, pages 265–278, New York, NY, USA, 2010. ACM.
[22] Matei Zaharia, Hruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, Ion Stoica, "Job Scheduling for Multi-User MapReduce Clusters", Electrical Engineering and Computer Sciences, University of California at Berkeley
[23] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar and Andrew Goldberg, "Quincy: Fair Scheduling for Distributed Computing Clusters", Microsoft Research, Silicon Valley — Mountain View, CA, USA
[24] Radheshyam Nanduri, Niteshaheshwari, Reddy Raja, Vasudeva Varma, "Job Aware Scheduling Algorithm for MapReduce Framework", 3rd IEEE International Conference on Cloud Computing Technology and Science Athens, Greece.
[25] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google file system", In 19th Symposium on Operating Systems Principles, pages 29–43, Lake George, New York, 2003.
[26] Stonebraker, M., "MapReduce and parallel DBMS: friends or foes?", ACM, 2010.
[27] Thomas Sandholm and Kevin Lai. "Dynamic proportional share scheduling in Hadoop in JSSPP", 15th Workshop on Job Scheduling Strategies for Parallel Processing, April, 2010
[28] Tom white, "Hadoop Definitive Guide", Third Edition, 2012
[29] V. Krishna Reddy, B. Thirumala Rao, LSS Reddy, "Research issues in Cloud Computing", Global Journal Computer Science & Technology Vol. 11, no. 11, June 2011,pp.70-76
[30] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, Wei Ding, "Data Mining with Big Data", 2013.
[31] Yang XIA†, Lei WANG1, Qiang ZHAO1, Gongxuan ZHANG2, "Research on Job Scheduling Algorithm in Hadoop".