Scheduling and Energy Efficiency Improvement Techniques for Hadoop Map-reduce: State of Art and Directions for Future Research

Ph. D Seminar Report

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

by

Nidhi Tiwari
Roll No: 114054003

under the guidance of

Dr. Umesh Bellur

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai
## Contents

1 Introduction ................................................................. 1

2 Map-Reduce Programming Model ........................................ 3
   2.1 Hadoop Map-Reduce Implementation .................................. 5
   2.2 Hadoop MR Scheduling Algorithms ...................................... 6

3 Scheduling Algorithms for Hadoop Map-Reduce ......................... 8
   3.1 Overview of Hadoop Map-Reduce Scheduling Problem ................. 8
   3.2 Taxonomy for Hadoop Map-Reduce Scheduling Algorithms .......... 8
      3.2.1 Adaptive vs Non-adaptive ........................................ 9
   3.3 Hierarchical Scheduling Algorithms .................................. 9
      3.3.1 User Level .......................................................... 10
         3.3.1.1 Fair Scheduling Algorithm .................................. 10
         3.3.1.2 Capacity Scheduling Algorithm ................................ 10
      3.3.2 Job Level ........................................................... 10
         3.3.2.1 First-Come First-Serve Scheduling Algorithm ................. 10
         3.3.2.2 Fairness Scheduling Algorithm ................................ 11
         3.3.2.3 SLA-based Scheduling Algorithms ............................... 12
      3.3.3 Task Level .......................................................... 16
         3.3.3.1 Map-Task Level ............................................... 16
            3.3.3.1.1 Replica-aware scheduling algorithms ....................... 16
         3.3.3.2 Reduce-Task Level .............................................. 19
            3.3.3.2.1 Locality-aware scheduling algorithms ....................... 19
         3.3.3.3 Speculative-Task Level ....................................... 21
            3.3.3.3.1 Latency-aware scheduling algorithms ....................... 22
   3.4 Evaluations done for different Scheduling Algorithms .......... 24
   3.5 Comparison of Scheduling Algorithms ................................ 25
   3.6 Conclusions and Future Directions for Research ................. 25
      3.6.1 Conclusions for Scheduling Algorithms .......................... 25
      3.6.2 Directions for Future Work for Scheduling Algorithms .......... 25

4 Energy Efficiency in Hadoop Map-Reduce Systems .................... 27
   4.1 Overview of Hadoop Map-Reduce Energy Efficiency Problems ........ 27
   4.2 Classification of Energy Efficiency Techniques proposed for Map-Reduce .. 28
   4.3 Techniques for improving Energy Efficiency .......................... 29
      4.3.1 Map-reduce programming model modification techniques ......... 29
         4.3.1.1 Workload energy-aware scheduling techniques ................. 29
         4.3.1.2 Hardware energy-aware scheduling techniques ................. 32
      4.3.2 HDFS cluster modification techniques .............................. 33
         4.3.2.1 Replica binning-based data placement and zoning techniques .. 33
         4.3.2.2 Temporal binning-based data placement and zoning techniques . 35
   4.4 Evaluations done for different Energy Efficiency techniques ........ 37
4.5 Comparison of MR Energy Efficiency Improvement Techniques .......... 38
4.6 Conclusions and Future Directions for Research ......................... 38
  4.6.1 Conclusions for Energy Efficiency Techniques ..................... 38
  4.6.2 Directions for future research in Energy Efficiency improvement techniques 39

5 References ... 42
List of Algorithms

1. Delay Algorithm ......................................................... 12
2. Schedulability Test Algorithm ........................................ 14
3. Constraint Scheduling Algorithm ...................................... 14
4. Maestro First wave Scheduling Algorithm ......................... 18
5. Maestro Runtime Scheduling Algorithm ............................ 19
6. Center of Gravity Reduce Scheduling Algorithm .................. 20
7. Longest Approximate Time to End (LATE) Scheduling Algorithm 22

List of Figures

1. Map-Reduce program workflow. Source - [DEA08] ................. 3
3. Taxonomy for Map-Reduce Scheduling Algorithms .................. 9
4. Hierarchical Scheduling policies proposed for Map-Reduce Clusters .... 9
5. Maestro Algorithm Results ............................................ 19
6. LATE algorithm results ............................................... 23
7. Classification Map-reduce Energy Efficiency Improvement Techniques .... 28
8. MIA workload analysis ............................................... 30
9. Energy Savings with BEEMR ......................................... 32
10. Cumulative Frequency Distribution of Lifespan metrics .............. 35

List of Tables

1. Notations used in deadline constraint scheduling algorithm in [KAM10] ... 13
2. Notations used in deadline scheduling algorithm in [JOR09] ............... 15
3. Summary of experiments done for evaluating scheduling algorithms in various papers .................................................. 24
4. Comparison of Scheduling Algorithms based on the criteria used by them ... 25
5. Notations used in Energy Framework of AIS techniques proposed in [WIL10] .. 30
6. Summary of experiments done for evaluating proposed energy efficiency techniques in various papers ........................................ 38
7. Comparison of Energy efficiency techniques for Map-Reduce for the desired properties ............................................. 39
Abstract

MapReduce has become ubiquitous for processing large data volume jobs. As the number and variety of jobs to be executed across heterogeneous clusters are increasing, so is the complexity of scheduling them efficiently to meet required objectives of performance. This report presents a survey of some of the MapReduce scheduling algorithms proposed for such complex scenarios. A taxonomy is provided for Map-reduce algorithms based on their runtime nature. The algorithms proposed for each hierarchical level of MapReduce scheduling are described in detail. Some pointers for future research to further improve the scheduling techniques are provided. Another aspect of MapReduce is that the size of their clusters is usually in hundreds and thousands, while it is used for processing infrequent batch and interactive jobs in parallel across these machines. Thus there is a need to look at energy efficiency of MapReduce clusters. A survey of some of the techniques proposed to improve MapReduce energy efficiency is done. The studied techniques have been classified based upon the MapReduce component they work-on. Details of techniques in each category are provided. Few suggestions for future research are given based on the gaps observed in these works.
1 Introduction

With the current trend in increased use of internet in everything, lot of data is generated and is analysed. Web search engines and social networking sites capture and analyze every user action on their sites to improve site design, detect spam and fraud, and find advertising opportunities. Facebook collects 15 TeraBytes of data each day into its PetaByte-scale data warehouse [THU09]. Powerful telescopes in astronomy, genome sequencers in biology, and particle accelerators in physics are churning out massive amounts of data for scientists. Key scientific breakthroughs are expected to come from computational analysis of such data. Many basic and applied science disciplines now have computational subareas, e.g., computational biology, computational economics, and computational journalism. Given these many use cases, there is a need to keep improving the BIG Data management techniques. The processing of this can be best done using Distributed computing and parallel processing mechanisms. Map-reduce is one of the most popularly used such technique. MapReduce breaks a computation into small tasks that run in parallel on multiple machines, and scales easily to very large clusters of inexpensive commodity computers. MapReduce is becoming a ubiquitous programming model. The leading example is Google, which uses its MapReduce framework to process 20 petabytes of data per day [DEA08]. Other instance are of Mars [BHE08] that harnessed graphics processors power for MapReduce. Hadoop, an open-source MapReduce implementation, has widely been adopted by industries such as Facebook, and academia. Hadoop [APHAD] is being deployed in many cloud platforms also. For example, Amazon has equipped their software stack with Hadoop to facilitate running large-scale data applications on Amazon EC2 [AMEC2]. The New York Times rented 100 virtual machines for a day to convert 11 million scanned articles to PDFs [GOT07]. In addition, researchers at Cornell, Carnegie Mellon, University of Maryland and PARC are starting to use Hadoop for seismic simulation, natural language processing, and mining web data [HADAP, SCH08]. Due to its wide adoption, the performance of Hadoop in particular (and MapReduce in general) has become an important research topic.

Each MapReduce application is run as a job that is submitted to the MapReduce runtime. Each job is split into a large number of Map and Reduce tasks before being started. The runtime is in charge of running tasks for every job until they are completed. The tasks are actually executed in any of the slave nodes that the MapReduce cluster comprises of. In particular, the task scheduler is responsible for deciding what tasks are run at each moment in time, as well as what slave node will host the task execution. One basic principle used is: moving computation towards data is cheaper than moving data towards computation. So, Hadoop attempts to schedule map tasks in the vicinity of input chunks seeking reduced network traffic in an environment characterized by scarcity in network bandwidth. In a multi-job environment, the task scheduler has the responsibility to ensure that performance is achieved for all jobs. Initially MapReduce was used for batch data processing, but it is now being used in shared, multi-user environments where different type of jobs with different priorities need to be executed: from small, almost interactive executions, to very long programs that can take hours to complete. In these new and changing scenarios, task scheduling is becoming even more relevant as it is responsible for deciding what tasks are run, and thus the performance delivered by each application to each user.

The US EPA report [EPA07] has indicated that the power consumption of datacenters is growing at a fast rate. It states that the energy usage at datacenters doubled between 2000 and
2006 which is equivalent to the electricity consumed by 5.8 million average U.S household. The report predicted that their power consumption will double again in 2011 to go up to 100 billion kilowatt hours/year which would be worth of $7.4 billion/year. This indicates the increasing operational costs for the enterprise data centers. The map-reduce clusters form a big part of data centers contributing to these huge energy expenses. The sheer scale of the Hadoop MR clusters make it critical to improve their operating efficiency, including energy. Yahoo’s data centers process 170 petabytes of data on cluster of 38000 servers [BAL10]. Such large clusters are created to support peak workloads. It was observed that Facebook workload had high peak-to-average ratios [YAN12] then the large clusters remain under utilized consuming peak power most of the time leading to energy inefficiency. Even with continuous stream of workload high number of long idle periods have been observed like a node was idle for 40 seconds or longer for 38% of the time [JAC10]. This also causes energy wastage. Over the lifetime of IT equipment, the operating energy cost is comparable to the initial equipment acquisition cost [CHE10] and constitutes a significant part of the TCO of a datacenter [BEL10]. The 3 years TCO for hybrid cluster of 2600 nodes itself has been shown to be around $15.1 million analytically [RIN10]. Hence, energy conservation of the extremely large-scale, commodity server farms has become a priority. There is a concerted effort to improve energy efficiency for Internet data centers, encompassing government reports [EPA07], standardization efforts [GREGR], and research projects in both industry and academia [MAT08, JAC10, KAM10, ZHE12, YAN12]. Considering the growing importance of Hadoop Map-reduce, complexity of the scheduling jobs/tasks and need for improving energy savings, this report surveys some of the research done on Hadoop Map Reduce scheduling algorithms and energy efficiency techniques. It also provides some research directions in these areas.

The next sections of report are structured as follows. Section 2 provides a detailed overview of MapReduce and Hadoop implementation. Section 3 highlights challenges in MapReduce scheduling, presents a taxonomy for MapReduce scheduling algorithms studied in this report, describes the scheduling algorithms proposed for each hierarchical level used in Hadoop Scheduling and suggests some points for future work. Section 5 underlines energy efficiency issues in MapReduce, provides a classification of energy efficiency improvements techniques, describes the proposed techniques in each category and provides pointers for future direction.
2 Map-Reduce Programming Model

Most common huge volume data processing programs do counting, sorting, merging etc. Such programs require to perform first a computation on each record i.e. requires to map an operation to each record. Then combine the output of these operations in appropriate way to get the answer, i.e. apply a reduce operation to groups of records. Map-reduce programming model [DEA08] provides simple map and reduce interfaces for users to define these operations. The programmer implements the map() function, that will execute on each input key/value pair to produce intermediate key/value pair output, and the reduce() function, which takes these grouped intermediate key/value pairs to generate the final result of the application. MapReduce runtime environment takes care of parallelizing their execution and co-ordinating their inputs/outputs as shown in Figure 1.

![Figure 1: Map-Reduce program workflow. Source - [DEA08]](image)

The MR runtime executes a MR program on large input data in distributed cluster is as follows:

1. The input file is split into M pieces of same size (16 to 64 MB data/piece depending on user’s configuration) and distributed on the cluster. The instances of user program are distributed to the machines in the cluster; one of them is assigned the role of master while others execute as workers.

2. Master instance’s job is to assign M map tasks (equal to M data pieces) and R reduce tasks to workers. It assigns one task at a time as workers become idle.

3. A worker executing a map task, parses the corresponding (local) data split and feeds each key/value pair to the user-defined Map function. The intermediate key/value pairs produced by the function as stored in memory.
4. The intermediate key/values in memory are partitioned into R sets by (map) worker using a (configurable) partition function (default is hash(intermediate key) mod R), so that same intermediate key/value pairs go to one partition. These pairs are flushed to local disk periodically into R files (1 for each reduce task). When the map task is completed, worker sends the locations (file names) of partitions to the master.

5. The master informs the idle/running reducer workers about the partitions locations (as when it receives updates from completed map tasks). The reducer workers read their corresponding partition files using remote procedure calls. After reading all the intermediate data, reducer worker sorts and groups them by intermediate keys. This is also called as shuffling task (reading and sorting by reducer).

6. The reducer worker then parses data for each intermediate key and passes all corresponding values to an instance of user-defined reduce function. The output of reduce is appended to a output file for this reduce partition on global file system.

7. After completion of all map and reduce tasks the control is passed to back to the calling program.

So the final output is the R files with consolidated data at intermediate key level in each file. These are mostly used as input for next MR or another distributed application. This execution flow shows how MR distributes data and load across nodes such that multiple nodes in the cluster execute the same (map or reduce) function but on a different chunk of input data.

Failure Handling is a key feature of MR programming mode. It works as follows:

1. For failure detection/management, master periodically pings all the worker machines, if a worker does not respond in stipulated time, it is marked as failed and no more tasks are scheduled on it. Master failure management can be implemented by having its backup.

2. In case master finds that a worker machine has failed, all the map tasks executed in past on it are reset to their initial state and rescheduled; as the local data on that worker won’t be accessible. The new partitions’ location are then sent to master, which sends these updated partitions location from then on to the new/already running reducer workers. As the output is on a global file system, reduce tasks executed in past on a recently failed worker are not marked for reschedule by the master.

3. In case a machine has been wrongly marked as failed and its map tasks have been rescheduled, multiple completion messages for the same maps will come to master. If master receives a completion message (with file names) for already completed map tasks, it ignores, else it stores those names and marks that map task as completed.

4. A reduce worker writes to a temporary file and renames its temporary file to final output name only on completion. The global file system handles the rename operation by multiple processes.

Some of the tricks used by MR programming model to provide improved performance are:

- Locality - MR programming model uses google file system as the underlying file system. This file system divides each file into 64 MB chunks and stores several copies of each chunk on different machines. The MR master takes the input file location into consideration and
schedules map tasks accordingly to have lesser network bandwidth utilization. First it
tries to schedule a map task on machine containing a replica of the corresponding data.
If the machine containing replica is not idle, then it looks for another idle machine in the
rack, then in same network switch area and so on for scheduling that map task.

- Backup Tasks - Some tasks may get hanged/delayed due to resource competition on the
corresponding machines. These are called straggler tasks. To handle these, when a MR
operation is nearing completion, master schedules a copy of these on idle workers and then
as soon as any one of the copy finished task is marked as completed. This is done to finish
stragglers computation faster and thus reduce a jobs response time.

2.1 Hadoop Map-Reduce Implementation

Hadoop is an open source MapReduce library implementation provided by the Apache Software
Foundation. It is being used widely for running huge data processing applications on large
clusters of nodes. It uses Hadoop File System (HDFS) as the underlying file system. The
combined architecture of Hadoop Map-Reduce is shown in Figure 2.

![Figure 2: Hadoop Map-Reduce Implementation Architecture. Source: [TOM09]](image)

The Hadoop File System (HDFS) has a master/slave architecture [JJF11]. The master
process (called as NameNode) manages the global name space and controls the operations on
files. Each slave process (called as DataNode) stores the files in form of data blocks and performs
operations as instructed by the NameNode. The NameNode manages the data replication and
placement for fault-tolerance, performance and reliability. The NameNode splits and stores files
in 64MB data blocks across the DataNodes. Usually 3 replicas of each data block are stored
in the HDFS. Failure detection mechanism is implemented in form of regular heartbeats from
DataNodes to NameNode. If there is no heartbeat from a DataNode for long time it is marked
as failed not used for further operations and if required additional duplicates of its data are
created [JJF11].

Hadoop MR also has a master/slave architecture. The master process is called JobTracker,
it distributes tasks and co-ordinates their input/output among the HDFS DataNodes. The slave
process called TaskTracker runs on each node in cluster. Each slave is assigned a fixed number
of map slots and reduce slots based on configuration and/or the number of cores in the node.
The TaskTracker executes map and reduce tasks, one per corresponding slot, on a DataNode as per master’s instructions. It also manages the data flow to/from map/reduce tasks. Hadoop assumes a tree-style network topology. Nodes are spread over different racks contained in one or many data centers. The bandwidth between two nodes is dependent on their relative locations in the network topology. So nodes that are on the same rack have higher bandwidth between them than those that are off-rack. This feature is used while scheduling data-local tasks among the nodes.

Hadoop MR provides a easy and quick implementation framework for efficiently processing huge volume of data on distributed cluster. Users submit jobs by specifying the data location, map/reduce implementations and consisting of a map function and a reduce function to Hadoop MR. Hadoop JobTracker distributes that program instance to TaskTrackers, breaks it into tasks, schedules them and tracks them till end. The high-level execution flow of a job in Hadoop MR is same as described in previous section.

Hadoop MR handles the straggler tasks (described in above section) by running their backup tasks called as speculative tasks on another machine. To identify straggler tasks Hadoop MR uses tasks’ progress score. For map tasks, the progress score is given by the fraction of input data read by that time. A reduce task’s execution is divided into three phases, and progress score is defined as fraction of phases completed (like 1/3 or 2/3). If the average progress score of a task is less than the average for its category by more than 0.2, and it has been running for at least one minute, then it is marked as a straggler.

Further as Hadoop Map-Reduce is used in shared cluster environments, to handle multiple jobs from multiple users on multi-core machines the Hadoop MR provides a few plug-gable/configurable job schedulers to enable different types of scheduling algorithms based on the data center’s needs. The scheduling algorithms used by these job schedulers are described below. (Please note that a scheduler is the controller that implements a particular scheduling algorithm, so scheduler and scheduling algorithms are interchangeably used in the report.)

### 2.2 Hadoop MR Scheduling Algorithms

In Hadoop, every TaskTracker sends frequent heartbeats containing the number of free map and reduce slots on that slave to the JobTracker. The JobTracker then assigns a task to the TaskTracker having free slots according to the configured scheduling policy. Hadoop schedulers mostly do a (2-level) hierarchical scheduling. First a job is selected from the pool of pending jobs, then that job’s task is scheduled. The task scheduling is done based on the type of the slot available. If its a map slot, then map tasks are scheduled as follows:

1. A map task bucket is selected from the 3 types of map tasks (3 buckets) in following order of priority: failed, normal, speculative.

2. From a bucket then a map task is selected based on data locality. The order of preference for selecting a task is a map task having local chunk on available node; a map task that has data on another node within the same rack; a task having data on another node outside the rack.

If free slot is a reduce slot, the reduce tasks from the queue are randomly selected and scheduled.

Multiple algorithms are available with Hadoop MapReduce like First-come First-server, Priority based, and Capacity based. As there are multiple objectives to be met and many variables
available for scheduling, there is scope of improvement in these algorithms. Many variations and improvements of the default scheduling algorithm are available.
3 Scheduling Algorithms for Hadoop Map-Reduce

3.1 Overview of Hadoop Map-Reduce Scheduling Problem

The key objective of Map-Reduce programming model is to parallelize the job execution across multiple nodes for execution. It creates multiple tasks to be executed and executes them on the multiple machines. Multiple combinations of task and machine are possible, scheduling policy is used to decide when and where (on which machine) a task is to be executed. The most common objective of scheduling is to minimize the completion time of a parallel application by properly allocating the tasks to the processors. Scheduling is a highly important factor, an inappropriate scheduling of tasks would fail to exploit the true potential of the system and offset the gain from parallelization.

The MapReduce tasks scheduling is a NP-Hard problem as it needs to achieve a balance between the job’s performance, data locality, user fairness/priority, resource utilization, network congestion and reliability. If scheduling policy considers data locality for selecting a task, it may have to compromise on the fairness as the node available may have data of some job which is not on head-of-line as per the fairness policy. Similarly if a task is scheduled based on job’s priority, it is not necessary that it would have local data on the available node. This would impact job’s performance, network utilization and thus also other job’s performance. The speculative tasks, executed for improving the reliability of a job, can cause resource wastage and hamper other job’s performance. Given the widespread utility of MapReduce programs in data analysis, they are used for long analytical jobs, short batch jobs, quick interactive jobs and so on. All enterprise data being stored in DFS, all the jobs are run on the environment. Some data centers or users want to achieve higher performance, some want high data locality, some want to meet SLAs for workload mix, some want to improve resource utilization and so on. The scheduling policies need to be designed differently for achieving different objectives in different scenarios.

The large Map-Reduce cluster is used to execute multiple jobs of different users. So, the scheduling policy needs to decide which user, which job and then which task to be executed on which machine. The users/jobs may have different priorities. The jobs would have different complexity, characteristics and requirements. The tasks would be of different type and have different data locations. The available node would have some speed, some capacity and other hardware characteristics. The scheduling needs to consider all these. In this scenario the task scheduling needs to be hierarchical, first select user, then his/her job, next the task for available node. Different policies can be used at each level. The scheduling algorithms proposed for each level are described here. Given the different objectives that need to be met, the multiple levels involved and the large number of variables available in MR environment, makes job scheduling an interesting problem for each combination.

3.2 Taxonomy for Hadoop Map-Reduce Scheduling Algorithms

In current large-scale multi-job clusters, Map-reduce scheduling algorithms are designed to meet objectives like SLA requirements, fairness or optimal resource utilization. To meet the desired objective scheduling algorithms use some decision making criteria and techniques while selecting a job or task on a particular machine. So, the taxonomy of MapReduce scheduling algorithms can be based on their runtime nature i.e. how they work. It could be based on if they consider the changing environment, scenarios etc or not; if they consider somethings, what all they consider
and so on.

3.2.1 Adaptive vs Non-adaptive

Based on their runtime flexibility, the algorithms can be termed as adaptive or non-adaptive. An adaptive scheduling algorithm uses the previous, current and/or future values of parameters to make scheduling decisions. A non-adaptive scheduling algorithm, on the otherhand, does not take into consideration the changes taking place in environment and schedules job/tasks as per a predefine policy/order.

For adaptive algorithms in MapReduce, the demand for scheduling adaptation comes from following points: the heterogeneity of cluster nodes, the data locality-awareness, the dynamism of workload arrival, and the diversity of applications’ execution times. So based on the key aspect considered for decision making at runtime, the adaptive algorithms can be further classified as shown in Figure 3. The scheduling algorithms falling in each category are also shown in the Figure 3. The algorithms of each class are described in later sections of the report.

![Figure 3: Taxonomy for Map-Reduce Scheduling Algorithms](image)

3.3 Hierarchical Scheduling Algorithms

In shared environment, the Hadoop Map-Reduce first selects a user whose job needs to be scheduled. Then from the list of jobs for the selected user it selects the job to be scheduled. Once the job is selected its map, reduce or speculative task is scheduled. Different decision making algorithms can be used at each level depending on the objective to be achieved. Multiple scheduling algorithms exists for each hierarchy level as shown in Figure 4. These algorithms are discussed in detail in this section.

![Figure 4: Hierarchical Scheduling policies proposed for Map-Reduce Clusters](image)
3.3.1 User Level

3.3.1.1 Fair Scheduling Algorithm

The objective of Fair scheduling (FS) algorithm is to do a equal distribution of compute resources among the users/jobs in the system [APHAD]. In case of multiple users, one pool is assigned to each user [APHAD]. The scheduling algorithm divides resources equally among these pools. The jobs within a pool can then be scheduled based on priority, FCFS or FS basis. Next the data locality is considered for selecting a task of the selected job.

FS scheduling algorithm provides a minimum share guarantee for each pool. If a pool does not get its minimum share for long time, FS preempts the most recently started task of an over allocated job from some other pool. This ensures that long batch jobs do not block the execution of some production jobs and also lesser amount of resources are wasted (that spent in the preempted task). The FS provides option to limit the number of jobs submitted in a pool to avoid concurrent execution of large number of jobs competing for cpu and memory resources. When Fair Scheduling algorithm is configured to consider job priorities, the priorities are used to assign weights to the jobs and resources are allocated as per the normalized fractions to the jobs.

Fair scheduling algorithms are useful in environments with different types of jobs. So that a lengthy batch job submitted first does not continuously occupy all the resources, and a short job submitted later gets some amount of resources to finish off in relatively lesser time instead of having to wait for that long time.

3.3.1.2 Capacity Scheduling Algorithm

The objective of Capacity scheduling is to maximize the resource utilization and throughput in multi-tenant cluster environment. The design of capacity scheduling algorithm is very similar to Fair scheduling [APHAD]. Here instead of pools, queues are used. Each queue is assigned to an organization and resources are divided among these queues. Additional security mechanisms are built for control access to the queues, so that each organization can access only its queue and cannot interrupt with other organization’s queues, jobs or tasks.

Similar to Fair scheduling, this algorithm offers minimum capacity guarantee by having limits on running/pending tasks and jobs from a single user/queue. For maximizing resource utilization, it allows re-allocation of resources of free queue to queues using their full capacity. When jobs arrive in that queue, running tasks are completed and resources are given back to original queue. It also allows priority based scheduling of jobs in an organization queue.

3.3.2 Job Level

The job level scheduling policies available with default Hadoop Map-Reduce are First-Come First serve (FCFS) and Priority based. These may not be efficient for all the scenarios and requirements. Therefore some new policies have been proposed for selecting a job for meeting the different users/administrators requirements. The preliminary algorithms, their improved versions and some newly proposed scheduling algorithms are described in this section.

3.3.2.1 First-Come First-Serve Scheduling Algorithm

The objective of FIFO scheduler to schedule jobs based on their priorities in first-come first serve order. FIFO is the default Hadoop scheduler which includes five priority levels as well.
When the scheduler receives a heartbeat indicating that a map or reduce slot is free:

- It scans through list of jobs to find a job with highest priority and then oldest submit time. If this job has a pending task of the slot type (map/reduce) then that job is selected. Else next job in this order is picked. This goes on till a matching task (type as per slot) is found.

- Next if its a map slot, then to achieved data locality the scheduler consults NameNode and picks a map task in the job which has data closest to this free slave (on the same node, otherwise on the same rack, or on a remote rack).

- If its a reduce slot, any of the reduce task is scheduled on the node. Hadoop MapReduce doesn’t wait for all map tasks to finish for scheduling a reduce task, so the map task execution and shuffling of intermediate data can run in parallel to have better turn-around time. This is known as early-shuffle.

### 3.3.2.2 Fairness Scheduling Algorithm

The naive Hadoop MapReduce Fair Scheduling at Job level always assigns free slots to the job that has the fewest running tasks to ensure that cluster is shared fairly between jobs. The newly submitted jobs are considered by fairness criteria whenever the next task-scheduling is to be performed. Once the job is selected, one of its tasks which has local data on the node, rack or within LAN is launched on the available node.

Two locality problems can occur with this naive fair scheduling [MAT10] -

- The head-of-line scheduling - The head-of-line problem is that the job at head-of-line has low data locality then most of its tasks cannot be scheduled on local nodes. This problem mostly occurs for small jobs, which have small input files and hence has less number of data blocks to read. The probability of finding their data on a given node is low whenever they come to head-of-line, still some of their tasks get scheduled.

- The sticky slots - This problem can happen for both small as well as large jobs. The problem is that there is a tendency for a job to be assigned the same slot repeatedly. Suppose multiple jobs each having 10 tasks are submitted together on 10 node cluster. So each job is allocated 1 node, in order of their arrival. When job J finishes a task on node N, Node N requests a new task. At this point, J has 9 running tasks while all the other jobs have 10. The naive Fair scheduling algorithm assigns that slot on node N to same job J again. Consequently, in steady state, jobs never get to leave a slot/node. This leads to poor data locality as the input files are striped across the cluster while all tasks of job are executed on same machine.

To avoid the above mentioned data locality problem simple strategy to delay the execution of the job till it can’t be executed on local data is proposed [MAT10]. This delay scheduling algorithm is works as shown in Algorithm 1.

This algorithm uses a threshold D to restrict the number of time a job can be skipped to avoid starvation. Another point to be noted is that once a job has been skipped D times, it can be launched on many non-local data nodes. So D is based on the overall delay a job can wait for. This policy is used to handle following 2 long lasting conditions:
Algorithm 1 Delay Algorithm

1: a node N requests for a task
2: repeat
3: select the next job J from the fairness based queue
4: if J has local data on Node N then
5: set number of times J’s skipped to 0
6: launch J’s task on node N
7: else
8: if number of times J’s skipped < threshold D then
9: increment number of times J’s skipped by 1
10: else
11: launch any of J’s task on node N
12: end if
13: end if
14: until all available slots are filled OR all tasks have been launched

- If all the slots on a node are filled with long tasks, the node may not free up quickly enough for other jobs to achieve locality.

- Presence of Hotspots, i.e. nodes which have local data for many jobs. Then multiple jobs would be trying to read a same small input file.

Later whenever a job manages to launch a local task again, its skipcount is set back to 0. This can help avoid future non local tasks as if sticky slot problem occurs then this job will keep getting assigned to same node whose data it has already read. An approximate analysis of how to set D to achieve a desired level of locality is provided based on the observations that (1) non-locality decreases exponentially with D and (2) the amount of waiting required to achieve a given level of locality is a fraction of the average task length and it decreases linearly with increase in number of slots per node.

The proposed algorithm was incorporated in Hadoop MR’s Fair Share (HFS) algorithm by using skipping time threshold instead of skipping count. The comparison of the naive HFS and fair share with delay algorithms was done using Facebook workload mix. It is observed that small jobs were benefited more at the cost of longer jobs. Also the large IO intensive jobs, with more number of map tasks, were benefited more from delay scheduling as compared to jobs with higher number of reduce tasks. It improved response times for small jobs by 5 time, and doubled throughput for IO-heavy workload.

3.3.2.3 SLA-based Scheduling Algorithms

The huge enterprise data is now being stored in distributed file systems for higher reliability and availability. This data is required as input not only for long batch jobs but also for many user jobs which many be online jobs or interactive jobs. As these are user tasks, meeting their SLAs is important. The default Map-Reduce implementation does not have SLA as a input/condition to be met. The default scheduling algorithm therefore does not consider these while scheduling jobs. So few SLA-based scheduling algorithms have been proposed for scenarios.

One such algorithm is the constraint based hadoop scheduling algorithm proposed by Kamal et. al. in [KAM10]. This algorithm maintains the minimum number of map/reduce tasks, provided by the job execution cost model, in the cluster to meet the SLA. The job cost execution cost model determines the minimum number of map/reduce tasks required to meet deadlines
based on the deadline, arrival time, input data size and map/reduce execution costs. The notations that will be used in expressing the job execution cost model and derived constraints are as follows:

Table 1: Notations used in deadline constraint scheduling algorithm in [KAM10]

<table>
<thead>
<tr>
<th>Notations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J (t_{m1}, t_{m2}, t_{m3}, ..., t_{mu}, t_{r1}, t_{r2}, ..., t_{rv})</td>
<td>Job J with u map tasks and v reduce tasks. t_{mi} is the i^{th} map task and t_{rj} is the j^{th} reduce task where 1 \leq i \leq u and 1 \leq j \leq v.</td>
</tr>
<tr>
<td>A</td>
<td>Job’s arrival time</td>
</tr>
<tr>
<td>D</td>
<td>Relative deadline of Job</td>
</tr>
<tr>
<td>(\alpha = (\alpha_1, \alpha_2, ..., \alpha_u))</td>
<td>Map data distribution vector (\alpha_i) is the data fraction allocated to (i^{th}) map task. Map input data is equally distributed among the map nodes so, (\alpha_i = 1/u).</td>
</tr>
<tr>
<td>(\sigma)</td>
<td>Filter ratio. The fraction of input that the map process produces as output. For most practical purposes (0 \leq \sigma \leq 1) holds</td>
</tr>
<tr>
<td>(c_m)</td>
<td>Cost of processing a unit data in map task.</td>
</tr>
<tr>
<td>(c_r)</td>
<td>Cost of processing a unit data in reduce task.</td>
</tr>
<tr>
<td>(c_d)</td>
<td>Communication cost of transferring unit data</td>
</tr>
<tr>
<td>(s_m)</td>
<td>start time of the first map task for the job</td>
</tr>
<tr>
<td>(s_r)</td>
<td>start time of the first reduce task for the job</td>
</tr>
<tr>
<td>(n_m)</td>
<td>Number of map slots assigned for the job in cluster</td>
</tr>
<tr>
<td>(n_r)</td>
<td>Number of reduce slots assigned for the job in cluster</td>
</tr>
<tr>
<td>(n)</td>
<td>Total number of slots for the job in cluster; (n_m + n_r)</td>
</tr>
</tbody>
</table>

The job execution cost model expresses Job’s execution time as combination of its map completion time, reduce completion time and data transfer during reduce copy phase as follows:

\[
\frac{\sigma c_m}{n_m} + \frac{f\sigma c_r}{n_r} + f\sigma c_d \tag{1}
\]

Now to meet the deadline constraint time when job finishes should be less or equal to expected deadline. This can be expressed as:

\[
s_m + \frac{\sigma c_m}{n_m} + \frac{f\sigma c_r}{n_r} + f\sigma c_d \leq A + D \tag{2}
\]

Thus the maximum time when a reduce task should be started is given by

\[
s_{r}^{max} = A + D - \frac{f\sigma c_r}{n_r} - f\sigma c_d \tag{3}
\]

Combining equations (2) and (3), we get

\[
s_m + \frac{\sigma c_m}{n_m} \leq s_{r}^{max}
\]

\[
n_m \geq \frac{\sigma c_m}{(s_{r}^{max} - s_m)}
\]
\[ n_m^{\text{min}} = \left\lceil \frac{\sigma \cdot c_m}{s_m} \right\rceil \] (4)

\[ n_r^{\text{min}} = \left\lceil \frac{f \sigma \cdot c_r}{A + D - f \sigma \cdot c_d - s_r} \right\rceil \] (5)

The modifications suggested for considering heterogeneous environment and partition-skew for reduce tasks are follows:

- The unit cost value for map and reduce task execution corresponding to the slowest node of the heterogeneous nodes is considered in the constraint equations.
- The worst case data skew is considered for reduce task, then 1 reduce task has to process all the records. In such cases, \( s_r^{\max} = A + D - f \sigma \cdot c_r - f \sigma \cdot c_d \)

The proposed constraint scheduler uses above criteria to decide to schedule a job or not using following algorithm 2. The user specified runtime values of map and reduce tasks.

**Algorithm 2** Schedulability Test Algorithm

1: a job is submitted with a deadline
2: calculate minimum number of map tasks required using equation 4
3: if number of map slots available < minimum number of map tasks then
4: reject the job, as deadline cannot be met
5: exit
6: else
7: calculate \( s_r^{\max} \) based on number of reduce tasks configured for the job
8: calculate number of reduce task slots at time \( s_r^{\max} \)
9: if no. of reduce slots at time \( s_r^{\max} \) < specified no. of reduce tasks then
10: reject the job
11: exit
12: end if
13: determine \( n_r^{\text{min}} \) using equation 5 to have at least \( n_r^{\text{min}} \) reduce tasks
14: end if

Once job is found to be schedulable, it is added to the constraint scheduler’s priority queue, which contains jobs arranged in increasing order of their deadlines. Thereafter constraint scheduler ensures that the minimum task count is met during the entire job execution to meet the SLA using the algorithm 3 shown below.

**Algorithm 3** Constraint Scheduling Algorithm

1: a TaskTracker reports N free slots
2: repeat
3: select the next job J in the priority queue
4: if no. of J’s map tasks running < J’s minimum no. of map tasks then
5: launch J’s map task on a free slot
6: reduce number of available free map slots
7: end if
8: if (J’s all map tasks are completed) AND (no. of J’s reduce tasks running < J’s minimum no. of reduce tasks) then
9: launch J’s reduce task on a free slot
10: reduce number of available free reduce slots
11: end if
12: until (map/reduce slots are available) OR (Jobs present in priority queue)
**Observations and Suggestions:** The constraint algorithm sacrifices data locality while scheduling tasks, still the evaluation results for a single word count application shows that it was able to meet the desired SLAs. However, the head-of-line scheduling and sticky slot problems [MAT10] can occur in this algorithms as data locality is not considered and minimum number of jobs need to be maintained. One suggestion for extension of this paper [KAM10] is to do a workload mix test to understand how the suggested estimation model and constrain scheduler will work in such scenario. Because the estimation model is based on isolated workload it may not work well in the workload mix scenario. The formula could be modified to include the waiting or delay introduced by other job’s tasks. As when multiple jobs are running each may have different costs and thus time for map/reduce tasks. If tasks of Job submitted at later point having longer task times get scheduled, their minimum number of tasks being less will keep occupying more number of slots introducing wait time for the jobs submitted earlier.

An adaptive task scheduling algorithm was proposed to meet the SLAs for Map-Reduce Jobs [JOR09]. This algorithm predicts remaining runtimes of the jobs in execution and dynamically adjusts the resources/slots allocated to them meet their SLAs. The authors studied multiple methods for estimating the job/task execution time at runtime. One based on number of tasks completed, elapsed time and number of remaining tasks. Another one based on input data size, elapsed time and data size of finished map tasks. But then an approach which considers the finished as well as partially completed tasks was selected. This estimated average task time is used to calculate number of slots required to finish the job based on the remaining number of tasks at a given point in time. The various formulas and notations used are given below.

\[
CT_{avg} = \sum_{i=1}^{N} (FT_i - ST_i) / N
\]

\[
PT = \sum_{i=N+1}^{M} (CurT_i - ST_i)
\]

\[
ET_{avg} = ElapT / N + \frac{PT}{CT_{avg}}
\]

\[
SR = \frac{RT}{RemT} \times ET_{avg}
\]

\[
Need = SR - M
\]

Here, The proposed deadline scheduling algorithm [JOR09] allocates the resources, in form of slots, according to their estimated need. Jobs with largest need gets the highest number of slots to finish as per their SLAs. The algorithm works as follows: The newly submitted jobs are classified as NODATA jobs (not enough data to estimate the jobs requirements). After execution of first task, job is moved to ADJUST class, whose jobs can be allocated slots as per

<table>
<thead>
<tr>
<th>CTavg</th>
<th>completed tasks’ average time</th>
<th>PT</th>
<th>total partial time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT_i</td>
<td>finish time of ith task</td>
<td>CurT_i</td>
<td>current time of ith task</td>
</tr>
<tr>
<td>ST_i</td>
<td>start time of ith task</td>
<td>M</td>
<td>no. of running tasks including partial tasks</td>
</tr>
<tr>
<td>N</td>
<td>no. of completed tasks</td>
<td>ElapT</td>
<td>elapsed Time</td>
</tr>
<tr>
<td>ETavg</td>
<td>estimated average task time</td>
<td>SR</td>
<td>slots required to meet SLA</td>
</tr>
<tr>
<td>RT</td>
<td>remaining tasks</td>
<td>RemT</td>
<td>remaining time to SLAs</td>
</tr>
<tr>
<td>Need</td>
<td>additional no. of slots required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
their Need. If Need of a Job is found to be more than available slots, it is moved to UNDEAD class which identifies jobs with past deadlines. Once all tasks of a Job are finished it is removed from the Job queue. The various classes of jobs are scheduled in following order of priority UNDEAD, NODATA and then ADJUST. The Need for each Job in ADJUST and UNDEAD class is updated at regular intervals. The Job with highest need is selected for scheduling within a given class. A workload mix experiment containing same type of jobs with different SLAs, showed that the algorithm was able to meet the SLAs using only the required number of slots for both while leaving some slots for other jobs.

Observations and Suggestions: For algorithm proposed in [JOR09] some experiments with mix of different types of jobs should be done to see what percentage and type of jobs are able to meet SLAs in such interfering environment. Scalability and overheads measurements could be done using more experiments. The estimations done for a job can be reused for similar jobs. As the accuracy of estimation increases towards the end, it will be high for later jobs from starting itself. Ateleast when same job is already running (if dont want to use a repository). This will also help in minimizing the estimation overheads.

3.3.3 Task Level

The third level of hierarchical scheduling is to select a task for scheduling on the available node. MR programming model creates two types of tasks map and reduce which need to be scheduled in respective slots. One more type of task created by MR during runtime is speculative task to handle straggler map/reduce tasks. These need to be scheduled on scheduled in a map or reduce slot depending on the type of task it is duplicating. Different scheduling algorithms have been proposed for selecting a task for the given slot type available to achieve different objectives like data locality, performance improvement and improve resource utilization. The section describes some of the scheduling algorithms proposed for each of the task type.

3.3.3.1 Map-Task Level

The default MR scheduling algorithm uses data locality criteria to select a map task for a given node. Initially JobTracker assigns the map tasks to the slaves depending on the TaskTracker slots capacity, considering data locality. At run time, when a TaskTracker reports an empty slot to process map task, the JobTracker checks for map tasks in its pool and consults the storage meta-data service to get the hosted chunks. It selects a map task for this node in following order of preference (1) a map task that has local chunk on that node, else (2) a map task that has data on another node within the same rack, else (3) a map task that has data on another node outside the rack. The map tasks pool has three kinds of map tasks: failed map tasks which have highest priority, normal map tasks and speculative map tasks with lowest priority. Some of the variations suggested for map task scheduling to improve the overall job performance are described below.

3.3.3.1.1 Replica-aware scheduling algorithms

The experiments done by Shadi et. al. [SHA12] show a high percentage (23%) of non-local map tasks executions with default map scheduling algorithm. This impacted task execution time and resulted in large percentage of speculative tasks executions (55%) out of which only 50% were successful. All this increased the job response time. It was shown that the local map
tasks cause speculation with a much lower probability as compared to non-local tasks. Also it was observed that non-local executions caused significant imbalance of the successful map tasks among different identical nodes. The possible reason for this was stated as follows. The native scheduling algorithm randomly selects a local task (out of all available local task) for the node. It doesn’t consider the consequences of current task scheduling on the next task in terms of the possibility of non-local map task execution. For example, consider a situation with two data nodes, A (hosts two chunks r1, r2) and B (hosts r1); at run time, if node A reports an empty task slot, the current Hadoop scheduling algorithm will blindly select one chunk without considering the consequences of processing each hosted chunk; say it schedules processing of r1 on node A then later it will cause a non-local map task execution on node B. To avoid such situations of non-local task executions in future a probability based, fine grained replica aware, Maestro algorithm is proposed [SHA12]. The proposed Maestro scheduler stores one additional factor for the chunk-level hosting status—the expected (remaining) number of map tasks executions to be launched. Maestro keeps track of the chunks locations along with their replicas locations and the number of other chunks hosted by each node. So that it can efficiently schedule the map task on a data local node which causes minimal impacts on other nodes local map tasks executions. It does map task scheduling in two waves, initially at the start of job and then at run time whenever a TaskTracker reports idle slot. It uses following heuristics to have lesser non-local map tasks and improve map tasks load balance across nodes.

During initial wave

- select the data node (of all available the chunk replicas) which has the lowest probability of executing local map tasks in future.

- select the data node which has minimal impact on other nodes local map tasks executions. That is a local node which has the lowest share rate with other nodes. Share rate of a node is the maximum number of shared chunks it has with any other node. \[ShareRate_{N_j} = \max_{1 \leq i \leq N, i \neq j} S_{c_j},\] where \(S_{c_j}\) is the number of shared chunks between \(n_i\) and \(n_j\), \((S_{c_j} = n_{i} \cap n_{j})\).

During runtime wave

- select the chunk which has maximal probability of not being processed locally.

Some of the parameters used in algorithm are:

- A cluster consists of k servers: \(n_1, n_2, ..., n_k\).

- Number of hosted chunks in node \(n_j\) (denoted as \(HCN_j\)). The HCN value indicates the number of unprocessed chunks hosted by a node. So after a task \(m_t\) is launched, the HCN of the all nodes hosting a replica of this specific chunk \(c_i\) are decreased by 1.

- The chunk weight (denoted as \(Cw_i\)). The \(Cw\) value indicates the probability of processing \(c_i\) on none of its hosts (i.e. non-locally). Thus select a local chunk with highest chunk weight, it will reduce the risk of executing it on a non-local node later. It is given by: \(Cw_i = 1 - (1/HCN_{i1} + 1/HCN_{i2} + ... + 1/HCN_{ir})\), where \(r\) is the number of replications of \(c_i\) and \(HCN_{ij}\) is the number of the chunks hosted by the data nodes which have \(c_i\).

- To incorporate fault handling, the priority of the chunk having a copy on failed node is increased by modifying the chunk weight as follows:
$C_{w'} = 1 - \left( \frac{1}{HCN_1} + \frac{1}{HCN_2} + \ldots + \frac{1}{HCN_{r-1}} \right)$, now $C_{w'} > C_w$ so that chunk will be picked for local data with higher probability than others.

- To incorporate cloud heterogeneity the chunk weight is calculated as follows:

$$C_w = 1 - \left( \frac{s_1}{HCN_1} + \frac{s_2}{HCN_2} + \ldots + \frac{s_r}{HCN_r} \right)$$

where $s_j$ denotes the slots capacity of node $j$. Thus select the chunk shared with nodes with smaller slots capacity (as higher $s_j$ leads to smaller $C_w$). Therefore by using the nodes with lower capacity for local maps, the higher computation capacity nodes are left out for other chunks possible non-local maps.

- A combination of chunk weight and share rate, NodeW$_i$.

$$\text{NodeW}_i = \sum_{j=1}^{HCN_i} \left( 1 - \sum_{k=1}^{r} \frac{1}{(HCN_k + Sc_k)} \right)$$

where $r$ is the number of replication of chunk $i$. This is used to find the nodes with higher potential in processing less local map tasks which could be due to them having relatively lesser number of chunks or having low sharing rate. By sorting nodes in ascending order of NodeW$_i$, it orders them by the sum of their hosted chunks weights, while prioritizing nodes which share chunks with more nodes.

The first wave of scheduling is used when the job starts. It schedules map tasks on the nodes which have higher potential of processing less local map tasks. For all nodes, select the node with minimal $\text{NodeW}_i$ value, on this node schedule map for the chunk with maximal chunk weight. The detailed algorithm is shown in algorithm 4. The run time scheduler is used when a node reports an empty task slot. Then JobTracker checks for un-launched map task whose data is hosted on this specific node. For this Maestro also computes the $C_w$ of each local chunk and processes the chunk with the highest weight. If nodes report an empty slot and no local map task is found, then first local speculative map tasks are looked for and if they are not found the non-local map tasks are selected. Detailed algorithm is given in Algorithm 5. Evaluations done using sort benchmark in local cluster showed that maestro performs better than default by 25% and 11.8% for 40-maps and 160-maps, respectively. InGrid5000 homogeneous environment Maestro reduced response time as compared to Hadoop by 34% and 6% for 200-maps and 1600-maps respectively. As with Maestro, the number of non-local map tasks is 16% in Hadoop, and it is only 3% in Maestro. In heterogeneous environment, Maestro outperformed Hadoop by 24% and 4% for 200-maps and 1600-maps respectively for sort application. The wordcount

---

**Algorithm 4 Maestro First wave Scheduling Algorithm**

1: while slots available do
2: find $\max_{1 \leq k \leq NS_k}$
3: sort nodes according to their $\text{NodeW}_j$
4: while $s_j < \max_{1 \leq k \leq NS_k}$ do
5: select next node \{select node with max share rate\}
6: end while
7: find $\max_{1 \leq i \leq HCN_j} C_{w_i}$ \{find chunk with maximum chunk weight\}
8: Launch $\text{map}^{NL}(c_i, n_j)$ \{Normal local map task for chunk $c_i$\}
9: $s_j = s_j - 1$ \{reduce share count\}
10: for all $n_j$ hosting $c_i$ do
11: remove $c_i$
12: end for
13: for all $n_j$ do
14: calculate $\text{NodeW}_j$
15: end for
16: end while
Algorithm 5 Maestro Runtime Scheduling Algorithm

1: a heartbeat is received from node $n_j$ reporting free slot
2: if local tasks $Map_{N_j}$ are available then
3: find $\max_{1 \leq i \leq HC_{N_j}} C_{w_i}$ (find chunk with max chunk weight)
4: launch $map_{N_i}^{L}(c_i, n_j)$ {schedule normal local map task for max chunk}
5: else
6: if $n_j$ hosts speculative map task’s input then
7: launch $map_{N_i}^{SL}(c_i, n_j)$ {schedule speculative local map task}
8: else
9: launch $map_{N_i}^{SN}(c_i, n_j)$ {schedule speculative non-local map task}
10: end if
11: end if

application response time was improved by 13% and 6% for 200 and 1600 maps as shown in Figure 5.

Figure 5: Percentage improvement in response time with Maestro in Heterogeneous environment Source: [SHA12]

Observations and Suggestions: One observation from the various experiment results was that the maestro algorithm [SHA12] doesn’t seem to be scaling well with increase in number of maps and nodes. As the percentage improvement in response time is reducing with increase in number of maps in same environment and relatively lesser with increase in number of nodes/maps. This could be due to for loops (of order of number of nodes and maps) used in first wave specially.

3.3.3.2 Reduce-Task Level

Once even a single map task of a job is completed, Hadoop MR starts scheduling its reduce job so that the map task execution and shuffling of intermediate data can run in parallel to have better turn-around time. This is known as early-shuffle. Hadoop MR randomly selects the reduce the task of the selected job for scheduling on the available reduce slot. Very few improvements have been suggested in literature for reduce task scheduling. Some of the modifications suggested were to meet jobs SLA as discussed in above section 3.3.2.3. A reduce task scheduling algorithm is described below.

3.3.3.2.1 Locality-aware scheduling algorithms

As the reduce task involves collecting data from the map nodes, some network congestion and performance problems can occur due to random reduce task scheduling by the Hadoop Job tracker. Consider the case when the node providing the input to a given reduce task is not near-by it, then this can cause lot of data shuffling and network traffic. Another problem could
be of partitioning skew, i.e. the intermediate keys frequencies and distribution across nodes can be different causing high volume of data for some reduce tasks. Both these would impact the application performance. To address these issues, a locality-aware skew-aware Center of Gravity reduce task scheduler (CoGRS) is proposed [MOH12]. Here a reduce task is scheduled on node considering nearness to its feeding nodes and the skew in its partitions. In Hadoop tree-style network topology the bandwidth between two nodes is dependent on their relative locations in the network topology. So nodes that are on the same rack have higher bandwidth between them than those that are off-rack. The bandwidth between nodes is represented as distance; distance between 2 nodes is measured by adding distances to a common ancestor and distance of a node to its parent is said to be 1. Using this, total network distance of a reduce task, R (TNDR), is defined as $\sum_{i=1}^{n} ND_i R$ where n is the number of partitions that are fed to R from n feeding nodes and ND is the network distance required to shuffle a partition i to R.

A metric called Weighted Total Network Distance (WTND) Per Reduce Task is designed considering the location of feeding map nodes, in form of network distances, and relative partition size generated by them, as weights for those map nodes. This metric used is for determining Center of Gravity (CoG) node for scheduling reduce tasks. It is given by $\sum_{i=1}^{n} ND_i R \times w_i$, where n is the number of partitions needed by R, ND is the network distance required to shuffle a partition i to R, and $w_i$ is the weight of a partition i. The node having minimum value of WTND is designated as the CoG node for the given reduce task. This WTND can be precisely calculated when all the feeding map tasks are completed. But Hadoop MR uses early-shuffle. It is observed that Hadoop performs much better with early shuffle ON rather than with it as OFF. So WTND needs to be calculated in intermediate state itself, i.e. without deferring shuffling data to reduce tasks until the map phase is fully done. It is assumed that with a certain probability all or some of the receiver reduce tasks will appear/receive their partition at an early period of Ms processing time. Accordingly determination of CoGR is promoted to be done after a (say alpha) percentage of map tasks commit, and then start shuffling Rs required partitions. This is known as pseudo-asynchronous map and reduce phases approach. The value of alpha is application dependent and determined by doing multiple rounds of experiments. The CoGRS algorithm using the above metrics and measures is described in Algorithm 6.

**Algorithm 6** Center of Gravity Reduce Scheduling Algorithm

1: a node N polls for a reduce task \{JT checks for reduce task for TT\}
2: for all reduce task R do
3: find CoG node = $\min_{\text{WTND}_{R}}$
4: end for
5: select R which has CoG node as N
6: if no. of Rs having CoG as N > 1 then
7: select the reduce task R that consumes largest input partition at N
8: launch R on N
9: else
10: if no. of Rs with CoG as N = 0 then
11: find R having $N_R$ with max progress score < $\beta$ \{select R whose COG remain busy for longest\}
12: if If no. of such Rs > 1 then
13: select R whose COG node $N_R$ is the closest to N
14: end if
15: launch R on N
16: end if
17: end if
Experiments showed that CoGRS shortened the map phase as compared to early-shuffle on case. It decreases the shuffle phase time for benchmarks with higher partition skew. But in case of uniform data it was observed to have slightly increased shuffle phase due to its processing overheads. It increased and decreased the reduce phase depending on time when the data will be available for a reduce phase. So overall CoGRS decreased the execution times only by 3.2% to 6.3% as compared to default Hadoop MR with early-shuffle on. It did increase the node-local data significantly by 1%, 32%, and 57.9% from native Hadoop on 8, 16 and 32 node clusters respectively in EC2 cluster.

**Observations and Suggestions:** Experiments of CoGRS [MOH12] show that the main advantage of this algorithm is in saving network bandwidth by having higher node-local data, not in execution time improvement. This brings up a question as to when to use this algo (for which workload for what configurations) as it doesn’t seem to be beneficial in all cases. Some work can be done in this direction. A workload mix testing can also help understand its use case as one expectation is that it would further improve the overall performance due to increase in network availability.

### 3.3.3.3 Speculative-Task Level

In Hadoop, if a node is available but is performing poorly, a condition that we call a straggler, MapReduce runs a speculative copy of its task (also called a backup task) on another machine to finish the computation faster. The goal of speculative execution is to minimize a jobs response time. A speculative task is run based on a simple heuristic comparing each tasks progress to the average progress. Hadoop monitors task progress using a parameter called Progress Score which has value between 0 and 1. For a map, the Progress Score is the fraction of input data read. For a reduce task, the execution is divided into three phases, each of which accounts for 1/3 of the score. Hadoop looks at the average Progress Score of each category of tasks (maps and reduces) to define a threshold for speculative execution. When a tasks Progress Score is less than the average for its category minus 0.2, and the task has run for at least one minute, it is marked as a straggler.

The speculative task scheduling in Hadoop is based on multiple assumptions, one is that data center is homogeneous, all tasks progress at same rate (while some may be local, some remote, some more compute intensive etc), and all reduce tasks process same amount of data. So if any of these is invalidated, their execution can cause competition and may cause Hadoop to perform poorly. Thus scheduling of speculative tasks which actually help minimize delay is complex. First because it is difficult to select the task for which to run speculative task as it would be difficult to distinguish between nodes that are slightly slower than the mean and stragglers especially in heterogeneous environment. Then it is useful in decreasing response time only if stragglers are identified as early as possible, so it needs to scheduled at right time. Few other points that need to be considered while deciding and scheduling them would be the competition of resources (network, cpu etc) created by speculative (nothing but duplicate) tasks and selecting node to run them. Some of the improvements suggested to make the speculative tasks serve their purpose when the some of the assumptions are invalidated and to improve the accuracy with they can be scheduled so that they don’t cause unnecessary competition are described below.
3.3.3.3.1 Latency-aware scheduling algorithms

Zaharia et al. address the problem of robustly performing speculative execution to maximize performance. The proposed Longest Approximate Time to End (LATE) algorithm [MAT08] is based on three principles: prioritize tasks to speculate, select fast nodes to run on, and cap speculative tasks to prevent thrashing. To realize these principles LATE algorithm uses following parameters:

- **SlowNodeThreshold** - This is the cap to avoid scheduling on slow nodes. Scores for all succeeded and in-progress tasks on the node are compared to this value.
- **SpeculativeCap** - It is the cap on number of speculative tasks that can be running at once.
- **SlowTaskThreshold** - This is a progress rate threshold to determine if a task is slow enough to be speculated upon. This prevents needless speculation when only fast tasks are running. Progress Rate of a task is given by \( \frac{ProgressScore}{ExecutionTime} \).
- **The time left parameter for a task is estimated based on the Progress Score provided by Hadoop, as \( (1 - \frac{ProgressScore}{ProgressRate}) \).**

The LATE algorithm works as shown in Algorithm 7.

**Algorithm 7 Longest Approximate Time to End (LATE) Scheduling Algorithm**

```
1: a node N asks for a new task
2: if number of running speculative tasks < SpeculativeCap then
3:   if nodes total progress < SlowNodeThreshold then
4:     ignore the request
5:   else
6:     rank currently running tasks that are not currently being speculated by estimated time left
7:     repeat
8:       select next task T from ranked list
9:       if progress rate of T < SlowTaskThreshold then
10:          Launch a copy of T on node N
11:       exit
12:     end if
13:   until while ranked list has tasks
14: end if
15: end if
```

Authors have done exhaustive experiments for LATE algorithm in EC2 heterogeneous cluster. One experiment showed that in a cluster with non-faulty nodes experiment (without stragglers), LATE finished jobs 27% faster than Hadoop native scheduler and 31% faster than no speculation. LATE provides gains in heterogeneous environments even if there are no faulty nodes. For Sort with stragglers, on average, LATE finished jobs 58% faster than Hadoop native scheduler and 220% faster than Hadoop with speculative execution disabled. The comparison of worst, best and average-case performance of LATE against Hadoops scheduler and no speculation for runs without and with stragglers are shown below in Figure 6. Sensitivity analysis to SpeculativeCap done in test environment showed that response time drops sharply at SpeculativeCap = 20%, after which it stays low. And a higher threshold value is undesirable because LATE wastes more time on excess speculation. Experiments for Sensitivity to SlowTaskThreshold (percentile of progress rate below which a task must lie to be considered for speculation) show that small threshold values harmfully limit the number of speculative tasks, values past 25%
all work well. Sensitivity analysis to SlowNodeThreshold (percentile of speed below which a node will be considered too slow for LATE to launch speculative tasks on) show that as long as SlowNodeThreshold is higher than the fraction of nodes that are extremely slow or faulty, LATE performs well.

Another small strategy Benefit Aware Speculative Execution (BASE) was proposed by Zhenhua et. al. in [ZHE12]. According to this, speculative tasks are launched only when they are expected to complete earlier than the original tasks. When a node $N_i$ becomes free, a slow task $T$ of job $J$ is identified (based on its progress score using same approach as in Hadoop and Late), then decision to run a speculative task on this Node $N_i$ is taken using this algorithm:

- If some tasks belonging to job $J$ are running or have run on node $N_i$, the mean of their progress rates is calculated and used as the progress rate of speculative task $T_0$.
- Else, progress rates of all scheduled tasks of job $J$ are gathered and normalized against the reference baseline to eliminate the effect of hardware heterogeneity. Then the mean of normalized progress rates is calculated. Then to compute the expected progress rate of $T_0$ on $N_i$, mean is de-normalized against the specification of node $N_i$.
- Using the estimated progress rate of $T_0$, the execution time ($1/$progress rate) is calculated. If the difference of the estimated execution time of $T$ and $T_0$ is larger than a preset threshold, $T_0$ is launched on $N_i$.

Additionally they proposed an approach of Resource Stealing to steal residual resources if corresponding map/reduce slots are idle, and hand them back whenever new tasks are launched to occupy those slots. It is adaptive to resource utilizations as it runs periodically with the up-to-date information of task execution and system status. Resource stealing is applied on each slave node locally, it is transparent to the task scheduler and can be used in combination with any existing Hadoop scheduler directly such as fair scheduler and capability scheduler. Following policies are suggested to distribute residual resources among running tasks:

- Even Evenly allocate residual resources to running tasks
- First Come First Server The task that started first/earliest gets the resources
- Shortest Time Left Most The task with shortest remaining time gets the resources
- Longest Time Left Most The task with longest remaining time gets the resources
- Laggard Task Most - Speculative Tasks running on the node gets the resources
During experiments with map-only benchmarks following observations were made: It was observed that only Resource stealing shortens run time by 5-58%, with Even policy. Also it was seen that lower the workload is, the more resource stealing outperforms native Hadoop. With both resource stealing and BASE enabled it was observed that native Hadoop performs the worst, the performance superiority of resource stealing decreases with increasing system workload and BASE slightly shortens run time. Here LTM gives the best performance. It was observed that BASE drastically eliminates the launches of non-beneficial speculative tasks. For workload 75% and 90%, almost all of them are removed. With reduce-mostly jobs following observations were done: Only Resource stealing (LTM policy) substantially shortens job run time by 71% and 44% respectively and thoroughly eliminated non-beneficial speculative tasks. Additionally enabling BASE shortens job execution marginally, but reduces the number of non-beneficial speculative tasks by up to 90% compared to native Hadoop. There were some overheads of doing multi-threading. For network intensive workloads, resource stealing (with speculative execution disabled) shortened the run times. For IO intensive workloads, Resource stealing performs only slightly better than native Hadoop scheduling, although resource stealing achieves higher parallelism within each task due to the (IO read/write) resource contention.

Observations and Suggestions: As BASE algorithm [ZHE12] build upon LATE algorithms’ [MAT08] methods experiments to compare LATE and BASE algorithms (without resource stealing) could have been done to provide the benefits of the additional logic added on it. An approach to run BASE algorithm after LTM resource stealing should be explored using experiments.

3.4 Evaluations done for different Map-Reduce Scheduling Algorithms

The efficacy i.e. the effectiveness and sufficiency of any approach can be proved by showing the results of its extensive evaluation. The algorithms discussed above were also evaluated in respective papers with different rigors. The Table 3 below summarizes the various evaluation aspects covered by each of the paper. As seen in the table the experiments with mix workload were not performed by most of the papers, which is required in the current scenarios where data centers are shared multiple users for running different types of jobs. Also it was observed that not many papers exclusively performed the tests for studying the overheads of the proposed algorithms and their scalability with increasing number of nodes to thousands as seen in current Map-reduce clusters.

Table 3: Summary of experiments done for evaluating scheduling algorithms in various papers
3.5 Comparison of Scheduling Algorithms

The job and task scheduling for different users in a shared Hadoop environment is done hierarchically for meeting objectives like reducing the latencies, meeting SLAs, improving data locality and improving resource utilizations. To meet these requirements, many improvisations/changes have been proposed for the default Hadoop scheduling techniques. As different algorithms work towards different goals and work at different levels, they are not compared for efficiency related metrics. The algorithms studied here are compared based on the objectives targeted, the type of tasks scheduled and the key factors considered. The Table 4 below shows objectives considered in the algorithms in their priority order. The type of task - Map, reduce or speculative, whose scheduling is improved by the given algorithms. The heterogeneous column is used to show if the algorithms consider the presence of heterogeneous nodes in cluster or not.

Table 4: Comparison of Scheduling Algorithms based on the criteria used by them

<table>
<thead>
<tr>
<th>Scheduling Algorithm</th>
<th>Minimizing latency</th>
<th>Meeting SLA</th>
<th>Maximizing Data Locality</th>
<th>Maximizing Resource Utilization</th>
<th>Fairness</th>
<th>Task Types (Map (M)/Reduce (R)/ Speculative (S))</th>
<th>Heterogeneous Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATE [MAT08]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
<td>Yes</td>
</tr>
<tr>
<td>Deadline-based [JOR09]</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>Yes</td>
</tr>
<tr>
<td>Delay [MAT10]</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>M</td>
<td>NA</td>
</tr>
<tr>
<td>Constraint-based [KAM10]</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>M</td>
<td>M, R</td>
<td>Yes</td>
</tr>
<tr>
<td>CoGRS [MOH12]</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>Yes</td>
</tr>
<tr>
<td>BASE [ZHE12]</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
<td>Yes</td>
</tr>
<tr>
<td>Maestro [SHA12]</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td>M</td>
<td>Yes</td>
</tr>
</tbody>
</table>

3.6 Conclusions and Future Directions for Research

3.6.1 Conclusions for Scheduling Algorithms

Need to process huge data volume is the key factor to use MapReduce which works on principle of moving process close to data. Local data processing takes lesser time as compared to moving the data across network. So to improve the performance of jobs, most of the algorithms work to improve the data locality. To meet the user SLAs, scheduling algorithms use prediction methods based on the volume of data to be processed and underlying hardware. The remaining task times are used to prioritize the task executions. For efficiently using the available resources, algorithms to prevent unnecessary speculative task executions have been proposed.

3.6.2 Directions for Future Work for Scheduling Algorithms

- The Hadoop MR has a centralized global scheduler which needs to track all the jobs and tasks running in the cluster. As the workload intensity and size of the clusters are increasing, the single job scheduler may get overwhelmed with the volume of work it has to do. As interactive, short response time jobs are being increasingly used, the scheduler needs to
make quicker decisions at least before a small task gets completed. Also to achieve different objectives of decreasing task times, increasing data locality, improving resource utilization, meeting SLAs of different jobs etc. complex decision making algorithms need to be implemented. This would definitely increase the processing time on scheduler. Therefore, distributed scheduling algorithms for clusters consisting of thousands of nodes should be looked into.

- Another option for improving performance of algorithms could be to make the cluster hierarchical and develop hybrid scheduling techniques. These hybrid algorithms would then use global information for local cluster optimization, to keep a balance of accuracy and efficiency.

- The existing algorithms use simple task time estimation techniques based on the job’s tasks progress rate. The average execution time thus obtained may not be applicable in heterogeneous environment. Also as data locality is achieved by only limited tasks, the execution times may or may not be able to factor it in, as calculation may happen after or before the tasks running on a local data node. Thus performance prediction models considering the data locality and node heterogeneity can be developed to improve the accuracy. They can be then plugged into various SLA aware scheduling algorithms.

- Another aspect could be to do data placement based on workload analysis and design the scheduling algorithms accordingly. Such combination of scheduling with data distribution strategies can help improve overall performance by reducing the interference of segregated workloads. So different workload analysis and data placement techniques need to studied.
4 Energy Efficiency in Hadoop Map-Reduce Systems

4.1 Overview of Hadoop Map-Reduce Energy Efficiency Problems

The data centers have been growing size to support large number of online user and process their huge data for different analytical purposes. Their operational costs including power, cooling etc has been increasing accordingly. The report from U.S. Environment Protection Agency (EPA) showed that the energy consumed by data centers has doubled from 2000 to 2006 and projected that it is expected to double again from 2007 to 2011 [EPA07]. So, the increasing operational costs of growing data centers have attracted a lot of attention over past decade. For their massive data processing many industries use map-reduce for processing huge volumes of data spread across large number of clusters consisting of hundreds and thousands of machines. These clusters consist of commodity hardware, medium size servers and a few higher performance servers in different ratios. Such large-scale infrastructure consumes lot of power. Thus power management for MR clusters has also become important.

The data centers have large number of machines as they are sized for handling peak loads which occur rarely, so the machines lie under utilized consuming power. A good option would be to have power consumption proportional to the throughput got from machine, that is its utilization. But the commonly used commodity machines in clusters have very narrow power ranges, so they consume peak power at low utilizations. The server machines have higher power range still can operate only at 3 power levels for different utilization ranges. This shows that the hardware is not yet power-proportional. Thus have low (approx. 50%) energy efficiency at low utilizations. The MR applications use data replicated across multiple machines for fault tolerance and the jobs arrival have variable distributions, so the average utilization of these machines in large data centers is observed to be 20-30%. Thus the MR clusters exhibit low energy efficiency. A common approach to improve the energy efficiency is to increase the amount of work done per unit energy. A simple way to realize this is to consolidate workloads on fewer servers and put idle servers on sleep. To realize this multiple dynamic workload placement and VM consolidation techniques have been proposed. Such redirection and server shutdown techniques work well for workloads which are not data intensive and access little required data from remote databases so are not bound to any machine. But these will not work so easily for MR applications.

In case of Map-Reduce application workload consolidation can cause increase in latencies as parallelism would be impacted, data would have to fetched from remote locations; putting down of servers can cause data unavailability as data is distributed across servers. A detailed case study in [NED09] shows that when a power management module did power-down of 30% of HDFS nodes then due to the in-built fault-tolerance mechanisms NameNode started replicating all their data causing network traffic to peak upto 110 MB/s. The scaling down of nodes also caused a high 25% files unavailability as some of their blocks, distributed across machines were not available. The MR performance was also shown to be proportional to the number of machines, as it is parallel in nature. So if the underlying file distribution and programming frameworks are not considered or involved in energy management, the application performance can get impacted adversely. Also as the Map-Reduce clusters are mostly shared to run different types of jobs, some being CPU intensive, some IO intensive etc on same data set. When a given job is running, energy of the other idle component of node is wasted. Another MR programming model related challenge is that for ensuring reliability, speculative tasks are run which are not
always useful and cause high energy wastage. Thus achieving energy efficiency for the data intensive MR systems is challenging. As it is designed to be highly parallel, distributed, fault tolerant and run close to data. Also the type and volume of workloads on shared MR systems varies largely unpredictably from batch to time-sensitive interactive jobs, and from low mean arrival rates to high peak rates.

4.2 Classification of Energy Efficiency Techniques proposed for Map-reduce

Given the hardness of problem of achieving energy efficiency in MR environments, more intelligent and involved, MR specific techniques are required for improving MR systems’ energy efficiency. Most of these techniques work to improve efficiency by reducing ideal periods on nodes by having lesser number of active nodes in cluster. They achieve this using job consolidation, data re-distribution and nodes re-configuration. For doing so they alter the design of either one or both of the two underlying frameworks, HDFS cluster and Map-Reduce programming model. Thus proposed techniques can be classified based on the MR system modified by them for improving the energy efficiency. Thus they can be classified into following 2 categories:

1. Map-Reduce programming model modification techniques
2. HDFS cluster modification techniques

1. Map-Reduce programming model modification techniques - The techniques which modify Map-reduce component do workload consolidation/distribution either based on workload characteristics or hardware characteristics. So they can be further classified as workload-based and hardware-based techniques. The Figure 7 below shows the list of techniques in each sub-class of Map-reduce based techniques.

2. HDFS cluster modification techniques The techniques which modify HDFS component work by consolidating the data of use on fewer active nodes so that other nodes can be put to sleep. For this they alter the data placement i.e. modifying the data distribution strategy of the cluster. The data is either segregated either to ensure one replica or to ensure critical data availability. So these techniques are said to be replica-based and data characteristics based. Figure 7 shows the list of energy efficiency techniques in each of these sub-classes.

![Figure 7: Classification Map-reduce Energy Efficiency Improvement Techniques](image)
4.3 Techniques for improving Energy Efficiency

Some of the energy efficiency improvement techniques proposed for MR cluster systems are described in this section.

4.3.1 Map-reduce programming model modification techniques

The techniques in this category modify the Hadoop MR Job and Task tracker components for energy aware job scheduling. The idea here to schedule jobs based on their jobs’/tasks’ characteristics and/or hardware characteristics so that energy consumption can be reduced. Based on the key criteria used for scheduling jobs, the techniques in this category can be further classified into following categories Workload energy-aware and Hardware energy-aware scheduling techniques.

4.3.1.1 Workload energy-aware scheduling techniques

As most common workload on Map-Reduce clusters are the random batch jobs, significant inactivity periods can occur on nodes like idle for 120 secs 20% of the time, idle for 40 secs 38% of the time and so on [JAC10]. And low utilization of 10-30% is observed most of the time. So with idea to use these idle and low utilization periods of MR nodes, Willis et. al. proposed a workload energy aware All-in-Strategy (AIS) [WIL10]. AIS is to run jobs on all nodes and power down all nodes when no work. AIS uses batching for consistently low utilization periods. It powers down all nodes during low utilization periods, batches the jobs and powers on all nodes, performs all jobs and again power down all when all jobs are completed.

Here the energy savings can be increased by putting more idle nodes in low power state, but that would increase the workload time thus increase the energy consumption during high power time. This increase would depend on the batched workloads’ computational complexity. The time to transition from one power state would also have significant impact on energy consumption. Especially if its much higher than the workload’s execution time then energy consumed during transitions will increase overall energy consumption. To capture these trade-offs between performance and energy savings an energy management framework is proposed. This framework finds out the impact of the workload characteristics, hardware characteristics and performance targets on energy consumption using a mathematical model shown in equation 1. Then using this it articulates the energy efficiency improvement as following constraint optimization problem shown in equation 8.

\[ E(\omega, \upsilon, \eta) = (P_{tr} \cdot T_{tr}) + ((P^m_w + P^m_w) \cdot T_w) + ((P^m_{idle} + P^m_{idle}) \cdot T_{idle}) \]  
\[ \eta = T_{tr} + T_w + T_{idle} \]  
\[ \min(E(\omega, \upsilon, \eta)) \cdot T_w \leq \tau \]

Here,

The duration for batching jobs could be determined solving the above constraint problem based on how much delay workloads can tolerate. The increase in time with changing number of nodes can be predicted using the workload complexity model. In addition workload prediction models can also be used to guide this energy management framework.

Observations and Suggestions: Currently the paper [WIL10] mainly uses only on the energy consumption model. The proposed response time constraint problem should be worked on to...
Table 5: Notations used in Energy Framework of AIS techniques proposed in [WIL10]

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>total nodes in the cluster</td>
</tr>
<tr>
<td>n</td>
<td>online nodes running the job</td>
</tr>
<tr>
<td>n'</td>
<td>N-n, offline nodes during job processing</td>
</tr>
<tr>
<td>m</td>
<td>online nodes during the idle period</td>
</tr>
<tr>
<td>m'</td>
<td>N-m, offline nodes during the idle period</td>
</tr>
<tr>
<td>υ</td>
<td>total observation time</td>
</tr>
<tr>
<td>T_tr</td>
<td>total transitioning time in υ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_w</td>
<td>workload runtime</td>
</tr>
<tr>
<td>T_idle</td>
<td>idle time</td>
</tr>
<tr>
<td>P_tr</td>
<td>average transitioning power</td>
</tr>
<tr>
<td>P_w[n,n']</td>
<td>on/off-line workload power</td>
</tr>
<tr>
<td>P_m[m,m']</td>
<td>on/off-line idle power</td>
</tr>
<tr>
<td>E(ω, v, η)</td>
<td>total energy consumption by workload in time, on hardware</td>
</tr>
<tr>
<td>τ</td>
<td>workload SLA</td>
</tr>
</tbody>
</table>

find the optimal configurations for different workload patterns. Simplest way could be to study it using experiments/model. This would also establish the feasibility and applicability of the proposed constraint problem.

With increasing demand for interactive and intelligent systems, data query and are becoming important part of all users and in-house applications. To support these the MR programs have evolved from being large batch jobs to being time-sensitive interactive jobs. The AIS will not work well for environments having such workload and almost no idle time, especially if shared.

Yanpei Chan et al did a detailed study of Facebook MapReduce workload in [YAN12] and found that their workload consists of long batch jobs, latency sensitive batch jobs and user interactive jobs. They called this combination as Map-Reduce with Interactive Analysis (MIA) workload. Some of the key observations during workload analysis were as follows: A high percentage jobs were small interactive jobs. There were some long jobs with small task times, their long duration was due to lack of parallelism among their tasks. The data sizes for most of the workloads were in KB to GB range. The input path accesses of workloads followed a Zipf distribution, i.e., large fraction of workloads accessed same few input paths. And most of the time small data size of 10s of GB was accessed per input path. All these observations characterized the MIA workload to be consisting mainly of interactive jobs which run for short periods and access a relatively small amount of common data. Based on these characteristics of workload Berkeley Energy Efficient MapReduce (BEEMR), an energy efficient Map-Reduce workload manager was designed which segregates interactive and batch workloads into separate sub-clusters to improve energy efficiency. The interactive jobs are served by a fraction of cluster with their dedicated storage. The less time sensitive batch and long jobs are batched and served.

Figure 8: Log-log plot of workload input file path access frequency; CDF of input size per job and size per input path. Source: [YAN12]
at intervals by the remaining larger cluster with their full storage and processing bandwidth. This increases the cluster utilization and results in energy saving between batch jobs executions. The cluster is split into 2 zones: a small interactive zone and a larger batch zone, each having different percentages of available capacity (task slots, memory, disk, network). The interactive zone is always in full power state, while the batch zone oscillates between full and low power states. The interactive zone has all, the input, shuffle and output, data associated with interactive jobs. This is achieved by making it function as a data cache. It stores most of the data required by interactive jobs. If there is miss, required data is migrated from batch zone (either immediately or in next batch execution) whenever it is in full power state. Both the zones store the error correcting code (ECC) or replicated blocks for their respective data. A key feature of BEEMR is that despite of selective switching off of nodes, there is no impact on write bandwidth and capacity of MR cluster. As full write bandwidth and capacity is available during batch mode when whole cluster operated in high power state. The small writes of interactive jobs can be well performed on the interactive cluster available in high power state all the time. The energy savings are achieved by workload based data segregation and batching without modifying the replication/ECC mechanisms used for fault tolerance in distributed file system. BEEMR implementation required Hadoop Job Tracker to be extended to have a wait queue, interruptible jobs handling and power-switching of batch cluster nodes. The scheduler was changed to perform interactive/batch/interruptible jobs scheduling on respective zone. Few changes in NameNode were required to ensure output storage in corresponding zone nodes and data fetching from other zone.

The authors have done lot of experiments using an empirically validated simulator of the proposed approach. Energy efficiency achieved with different combinations of cluster parameters (like total number of slots, ratio of map and reduce slots, tasks per job); and this technique’s parameters like batch interval length, job classification threshold were studied in great detail. Overheads in terms of increase in response time of various types of jobs was also studied. Following are some of the key observations:

- Use of latency-bound algorithm for calculating the number of tasks in a job, provided the higher energy savings than use of default algorithm. It also resulted in higher cluster utilization and lesser stragglers.

- Higher ratio of map to reduce slots (instead of default 1) resulted in higher energy savings, even more with latency-bound algorithms as shown in Figure 9(1). It also improves the slot occupancy.

- A lower interruptible threshold provides higher energy savings with default and actual tasks per job calculation algorithms. As it puts more jobs into interruptible category which can be completed in multiple batch rounds while each batch round has only faster batch jobs.

- The overheads of energy savings were observed in form of increased job latencies. But interactive jobs had least overheads as 40% had lesser or same latencies as default. Some batch jobs had really long delays (more than 100 times increase in latencies). Overhead on batch jobs was higher than that on interruptible jobs. In case of interruptible jobs it was observed that higher the energy savings (by having higher map/reduce ratio and lower interruptible threshold), higher the overheads (i.e. increase in latencies).
Simulation shows that idle and BEEMR energy savings are higher in larger clusters (with more slots). Here idle energy savings is based on the maximum slot occupancy. Idle energy saving = 1 - ((avg. no. of map tasks + avg. no. of reduce tasks)/total number of available slots. Still the gap between idle and BEEMR energy savings increase with cluster size as shown in Figure 9(2).

The best combination of parameters provided 0.55 fraction of energy savings. Paper shows empirically that BEEMR provides 40-50% energy savings.

![Energy savings at different values of map/reduceratio for different tasks/job selection policies. Ideal and observed energy savings for different cluster sizes. Source: [YAN12]](image)

### 4.3.1.2 Hardware energy-aware scheduling techniques

A map-reduce cluster can consist of different kinds of machines, some could be high and low power machines. This provides an opportunity to save energy by intelligently placing jobs on its corresponding energy efficient machine. Following key observations were made by Nezih Yigitbasi et al [NEZ11] during the experiments conducted on performance equivalent clusters of low and high power machines:

- Overall the low power machines had lesser dynamic power range and their cluster consumed higher power than the higher power machines cluster for all workloads.

- The low power machines cluster had 1.3 times higher response time for cpu bound workloads. The high power consumption and higher response time made low power machines cluster 2 times lesser energy efficient than high power machines cluster for cpu bound workload.

- The response time for IO bound workload was 3.5 times lesser on low power machines cluster than that on high power machine cluster. The immensely lower response time made low power machines cluster 2.5 times more energy efficient (despite of high power consumption) than high power machines cluster for IO bound workloads.

Based on the above observations, a heuristic was derived that IO bound workloads have better energy efficiency on low power nodes while cpu bound workloads achieve better energy efficiency on high power nodes. Another heuristic was that a map task is more CPU intensive while a reduce task is more IO intensive. The energy efficient scheduler proposed in [NEZ11] used the above 2 heuristics to select an energy efficient node for a given task. The metrics used for measuring energy efficiency of nodes are defined as records/joule and IOPS/watt. The Task Tracker module is modified to calculate these metrics using the monitoring agent on their node and send them to the Job tracker along with the heartbeats. The Job Tracker module is modified
to use energy efficiency as the first criteria for task scheduling. It schedules Map tasks on node with records/joule value higher than a map threshold and Reduce task on node with IOPS/watt value higher than a reduce threshold. If available node does not matched the energy efficiency condition, then fairness and data locality criteria are used to select a job and its task respectively for scheduling.

Another approach for scheduling jobs could be to ensure that all nodes run in medium power state and the peak power states of servers where high power is consumed is avoided. It would require to regulate the workload on the system so that few peaks observed in power consumption can be removed. This will help in saving the peak power provisioning and operational costs. A power model based adaptive approach is proposed in abstract paper [NAN11]. The power cost model is given for capping the system power consumption. The optimization and evaluation of the proposed approach in the abstract paper will be done in future.

4.3.2 HDFS cluster modification techniques

Data being at the heart of MR jobs, drives most of the optimization techniques for them include energy efficiency. Data locality is key for achieving good performance and distributed data replication for fault-tolerance is one of the main cause of its energy inefficiency. Therefore a lot of techniques work on the trade-off between MR performance and energy efficiency. These techniques capitalize on the opportunity provided by data volume, distribution and redundancy across nodes in HDFS. The nodes are partitioned into zones. These zones are periodically or based on some criteria disabled (but not all together) to save energy and enabled when required to serve different jobs. To ensure that unavailability of certain nodes at different times does not hamper data availability and performance of application, different data placement or redistribution and zoning strategies have been proposed. Some of these strategies ensure that at least one copy of every data block is always available. These can be called as Replica-binning based data placement and zoning strategies. Some the strategies consider the data access patterns based on time and/or workload to create the zones so that frequently accessed data is always available. These can be called as Temporal-binning based data placement and zoning strategies. A few MR energy efficiency improvement techniques using these two types of strategies are described below.

Default HDFS data placement strategy

HDFS mostly stores 3 copies of every data block using following default principles no two copies can be on same node and same data block should be there at least on 2 different racks. Experiments done with HDFS having 3 as replication factor showed that removing any 3 nodes caused from a single rack configuration made some of the data unavailable [JAC10]. So at most 2 nodes could be disabled in this scenario. In case of multiple racks configuration, a single rack can be powered down to ensure that all remains available.

4.3.2.1 Replica binning-based data placement and zoning techniques

To allow switching down of higher number of nodes without impacting data availability Jacob et. al. [JAC10] proposed another data placement principle for HDFS. They defined a subset of nodes as covering subset. And as per their principle at least 1 copy of each data should be present on one of the node of covering subset nodes. So now all data remains available till all nodes outside the covering subset of nodes are disabled. The size of the covering subset
is suggested to be 10 to 30% of the cluster size, so that a balance is maintained between the number of disabled nodes and the storage capacity, and the IO operations etc are not impacted. Covering set can be specified on file by file basis by the users, so when a user/application is inactive corresponding covering set can be disabled without impacting other users/applications.

The proposed design was realized by doing changes in HDFS. The master component i.e. the NameNode was modified to maintain at least one copy of each data block in covering subset as follows: Function for distribution of new data blocks need to work as per the strategy of having at least 1 replica in covering subset of nodes. The function was modified to distribute data block stores the first copy of data on the node which created/ is writing the data, second copy on a node in covering subset and third copy on a node which is not on same rack as the first copy node. The strategy for removing excess replicas (say when a faulty node re-joins) was changed so that it leaves at least one replica (of the data blocks found on re-joined node) in covering subset.

The strategy for adding more replicas in case of node failure / decommissioning was changed to consider the replicas in disabled nodes. For this two lists are maintained one internal list of data blocks containing mapping of each data block of active nodes to their nodes and one offline storage location list which contains the mapping of each data block to its location on the disabled node. So on decommissioning/disabling of nodes, the corresponding data blocks information and their locations are removed from internal list and written to the offline storage list. Also the execution of queries, checking for the number of copies of data, is modified to refer to both the lists.

Observations and Suggestions: A significant degradation of performance was observed in the experiment results [JAC10] which was not given due importance. It was mentioned that each node contributes lesser towards performance than what it contributes to energy consumption. But the performance is mostly more important to application owners than energy so high performance degradation may not be acceptable for energy efficiency. More analysis of test results should have been done to understand the reasons for the high degradation in performance. Like the utilizations, data locality during map operations (by studying map task latencies) in different test scenarios could have been studied to see if lesser compute or lesser data locality caused the performance degradation. As approximately only 1 copy of data is available the techniques for achieving data locality while task scheduling would need to be modified so that the performance is not impacted much.

Another approach for carefully placing the replicas of data blocks across distributed nodes, to allow certain number of nodes to be put in low power mode, was proposed by Nedeljko et. al. [NED09]. A meta data seggroup was used to store the list of machines containing replicas of a data block. The strategy used was to have 1 machine only in 1 seggroup. The available nodes would be divided into certain number (number of nodes/replication factor) of groups. Each group would have unique combination of nodes, which will be used to store copies of a certain data blocks depending on the replication factor. This multiple data blocks would have same seggroup and all but one node of each seggroup could be put to sleep mode for saving energy. Using the above data-placement policy along with other collaborative power management components a energy-aware Hadoop MR system design was presented. It consists of a common control plane connected with power management module HDFS and Hadoop MR services. The common panel is central manager who receives the requests for saving energy, co-ordinates with HDFS and Hadoop MR and communicates the strategy to power management model. The power
management actions like put 30% nodes on sleep can be initiated by human or automated. The common control plane receives these, communicates to cluster services and receives their feedback on which machines can be put to sleep. The HDFS and Hadoop MR provide this feedback based on their analysis that what cluster configuration will impact the performance and data availability least. The common control plane then consolidates the feedbacks from various services and forward the final decision to power management component about which machines to be put to sleep. To allow the power management policies to implemented in MR clusters without affecting the jobs performance, the HDFS and Hadoop MR were made energy aware by doing certain changes.

The default block replication policy of HDFS was changed to seegroup based policy described above. A new status called sleeping is added for DataNodes, along with the existing dead and active. The block inspection function was modified, so that when it finds that a data block is not available and its node status is sleeping then replication is not triggered. In addition a function is implemented to determine the nodes whose unavailability will not impact data availability based on seegroups. Similarly a function was implemented in Hadoop MR to determine nodes whose unavailability will not impact performance. This function was designed to select the remote data machines, so that more tasks process on local data and speculative tasks executions are avoided.

### 4.3.2.2 Temporal binning-based data placement and zoning techniques

Most of the user data has temporal quality, i.e. data created / accessed once is referred to frequently for sometime. The data caching and memory management techniques save the most recently used data and flush the least recently used one. The workload analysis of Facebook cluster [YAN12] showed similar quality for HDFS data, that same set of data is used by most of the interactive jobs as discussed in section 4.3.1.1. The workload analysis of Yahoo! data cluster [RIN10] also showed the temporal qualities of HDFS data and files (Figure 10):

- 90% of data was accessed quickly within 2 days of its creation.
- 89.61% of data was hot (last read) for less than 10 days after its creations.
- 40-60% data lied untouched for more than 20 days. This data can be classified as cold data.

![Figure 10: Cumulative Frequency Distribution of Lifespan metrics in the cluster, i.e. time between: (1) file creation and first read, (2) file creation and last read, (3) last file read access and file deletion. Source: [RIN10]](image)

In steady state 60% of data was in cold state. This data amounted to 70% of files in the system. Using their temporal quality data can be classified into hot/active and cold/passive zones. This
can be termed as temporal binning-based data placement and zoning strategy. The cold zone can be put to sleep while the hot zone can be used to serve most of the requests. The jobs requesting cold zone data can be batched and served together utilizing full capacity of nodes in short durations.

GreenHDFS [RIN10] uses the temporal binning-bases data placement and zoning strategy. Cluster is divided into Hot (high-temperature) and Cold (Low temperature) zones. Data is placed initially in these based on high-level data classification policies and later transitioned from one to another based on its access frequencies. Hot zone consists of high performance, high power and high cost CPUs. All nodes in Hot zone remain in high power mode at all times to provide maximum performance. Majority of the servers in cluster are assigned to Hot zone to ensure higher performance and higher data availability. Cold Zone consists of machines with higher number of disks to store large amount of rarely accessed data. Aggressive power management policies are used in cold zone to keep nodes in low, inactive power mode as long as possible. Nodes are powered-on only on-demand. To reduce the number and duration for which they need to be powered-on following policies are used:

- The data in cold zone is not replicated, so that when data is requested a single node needs to be transitioned to active power mode.
- The In-order data placement policy is used to write data, where a node is powered-on and filled to full capacity, then moved to inactive power state and next set of nodes are filled.

Once data setup is done, during runtime following policies are used:

- File migration policy - Files are moved from Hot zone to Cold zone as their temperature changes. A file's age, defined by the last access time of file, is used as a measure of its temperature. So the old files are moved to Cold zone. This creates more space for highly accessed files.
- Server power conservation policy. Most of the servers consume 50% of the peak energy even when idle. So idle servers are completely moved into cold zone. Servers are moved to Hot zone when a data access, data migration or disk scrubbing event is received.
- File reversal policy A file is moved from cold zone to hot zone if its number of accesses becomes more than a threshold number.

Experiments done using a simulated GreenHDFS showed that energy consumption for the workload of a period of 3 months, created using yahoo traces, was just 26% of the baseline energy consumption.

**Observations and Suggestions:** Server power conservation policy is not clearly explained, exactly when at what utilization server is transitioned to inactive power state/cold zone is not explained. If given 50% is to be used, then it is not mentioned how/where that 50% load is transferred before transitioning the node to inactive power state. Server power conservation policy seems to be orthogonal to the File migration policy which migrates old data [RIN10]. This is not brought out in paper at all. As paper talks only about hot/cold zones based on data classification. While server power conservation policy is based on server utilization levels. Time factor should also be included along with the number of access of a file in the file reversal policy else file migration may happen due to just a single job.
A completely different perspective to look at temporal binning-based data placement zoning could be to have dynamic HDFS cluster. Here the HDFS cluster configuration in terms of number of nodes varying depending on current workload intensity and the data is re-distributed on modified cluster. Nitesh et. al. proposed the algorithms based on such a strategy in [NIT12]. A cluster power controller interacts with HDFS to maintain optimal number of nodes in cluster for meeting the workload demands. The cluster is started with certain minimum number of nodes, then as the workload increases more number of nodes are included in cluster. When the average utilization of cluster goes below a threshold, certain nodes are put into sleep node to maintain cluster utilization above certain threshold. The difference between scale-up and scale down utilization thresholds needs to be large to avoid the workload jitter impact of frequently scaling up/down the cluster. Every time the cluster is scaled up/down, the data is re-distributed in a rack-aware manner. To control the network load due to re-balancing of cluster following heuristics are used:

- Maximum amount of data is transferred within the same rack as it has higher network bandwidth.
- The data currently being used by any tasks is not moved during re-balancing to avoid any impact on performance and fault tolerance. It is moved after that tasks processing is over.

To ensure that the rack-aware data distribution policy of HDFS is not violated following checks are used:

- While doing intra-rack transfer, a data block from a node to another node only if a replica of that block does not exist on the destination node.
- In the case of inter-rack transfer, a data block is moved from a node to another node only if a replica of that block does not exist on any nodes in the destination rack.
- The amount of data transferred, both inter-rack and intra-rack is controlled based on the utilization and disk capacity of the source and destination nodes.

Simulation based experiments done for proposed method showed that it was able to do scaling up and down of the system with varying workload so that the average node utilizations are maintained at higher energy efficiency levels. During these experiments 27-36% energy savings were observed based on the average number of inactive nodes.

Observations and Suggestions: The proposed approach of re-balancing the cluster [NIT12] can be a huge overhead on the cluster increasing its energy consumption every time. This impact needs to be considered while calculating the energy efficiency improvements. The impact of re-balancing on performance of jobs running in the cluster should also be studied. The experiments used write intensive jobs while read workloads are very common in map-reduce cluster, so experiments need to done with those workloads to see the impact of re-balancing on their performance and energy savings.

4.4 Evaluations done for different Energy Efficiency techniques

The efficacy i.e. the effectiveness and sufficiency of any approach can be proved by showing the results of its extensive evaluation. The techniques discussed above were also evaluated in respective papers with different rigors. The Table 6 below summarizes the various evaluation aspects covered by each of the paper.
Table 6: Summary of experiments done for evaluating proposed energy efficiency techniques in various papers

<table>
<thead>
<tr>
<th>Energy-efficiency Techniques</th>
<th>Metrics Studied</th>
<th>Implementation used</th>
<th>Input used</th>
<th>Cluster Used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Energy Efficiency</td>
<td>Response Time</td>
<td>Data Availability</td>
<td>Simulation</td>
</tr>
<tr>
<td>Segroup-based (NED09)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Metron nodes (NEZ[1])</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>iREM (YAN[2])</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CS (JAC10)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AM (IR[10])</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GreenHDFS (RIN10)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DynamicHDFS (NIT[12])</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

4.5 Comparison of MR Energy Efficiency Improvement Techniques

The various techniques studied above are compared against following key properties expected from them in Table 7 below.

- Energy efficiency - The energy efficiency metrics shows the percentage improvement reported by each of the paper over the native Hadoop Map-reduce. Though each one has different environments like simulation, private cloud and public cloud, different mechanism for power measurements like meter reading or counting number of active nodes, all have reported percentage energy efficiency improvement over the native Hadoop Map-reduce.

- Performance - As improving energy efficiency involves trade-off with performance, the proposed techniques are compared for their impact on application performance.

- Data availability - Data availability is an important aspect of Hadoop HDFS and MR systems, at the same time data replication is one of the factor for increased energy consumption by Hadoop MR. So various techniques are compared on how they handle this aspect by comparing them for impact on data availability and possibilties of getting hotspots criteria.

- Scalability - Considering the size of Map-reduce clusters and volume of workload they need to handle, any technique used need to be scalable to thousands of servers and high workload intensities. So the techniques are compared for their scalability.

- Overheads - Also the application performance should not get impacted much due to the optimization activities like improving energy efficiency, so the overheads introduced by each technique are listed.

4.6 Conclusions and Future Directions for Research

4.6.1 Conclusions for Energy Efficiency Techniques

Energy efficiency is mostly achieved at the cost performance and availability. Data distribution strategy is one of the key factor for improving the Hadoop MR energy efficiency. Workload intensity and mix analysis can significantly contribute towards energy efficiency strategy. As they can be used for intelligent deciding data placement and job scheduling strategies.
Few generic shortcomings observed during literature survey were that overheads for achieving energy efficiency were not analyzed in most of the papers. Scalability of the selected approach was not talked about in most of the papers. Power consumption has been actually measured only in few papers to find out the energy efficiency improvements, other represent energy efficiency in some abstract way like number of nodes active/inactive.

4.6.2 Directions for future research in Energy Efficiency improvement techniques

- **Study of Power consumption by Map-reduce cluster**
  
  All the papers have used the 2007 EPA [EPA07] report’s power consumption values and estimates to motivate the cost factor. And the non power proportional nature of machines proved in [LAB07] to show the energy wastage and thus need of research to improve energy efficiency of Map-reduce. Jacob et. al. studied the ideal server durations and their distributions for mixed map-reduce workloads to motivate their work of covering set [JAC10]. Yanpie et. al. analyzed the Facebook’s workload variation, and input data size/path for finding the server utilizations across the clusters [YAN12]. They found it to be low for most of the servers for most of the time, so designed BEEMR workload manager. Nezih et. al. [NEZ11] did experiments on small 1 and 7 nodes clusters of high performance and low power machines respectively to see the difference in performance and power consumption for a set of cpu, I/O and mix intensive benchmarks. Thus, there seems to be lack of detailed study done on energy consumed in Hadoop MapReduce data centers. An in-depth study of energy consumed by map-reduce clusters for different types of workloads, data volumes and their combinations is required. This would help underline the motivation to do work to improve energy efficiency for map-reduce, specifically.

- **Energy aware map-reduce job/task scheduling algorithms**
Some work has been done on energy aware scheduling on low-power and high performance machines in [NEZ11]. They observed that CPU consumed almost half of the total power for all workloads and that power ranges of the two machine types were very different. Based on power consumed and response time comparison it was found that energy efficiency of low power nodes was better for I/O bound jobs while that of high performance nodes was better for CPU bound jobs. Using these heuristics task scheduling algorithms was designed. More experiments to understand the applicability of these for different machines and types of jobs/tasks should be done. Then they can be used for scheduling those jobs/tasks accordingly.

A range of scheduling algorithms have been designed to improve the performance and resource utilizations of map-reduce clusters, as studied. These can thus also help improve the efficiency of map-reduce clusters also when clubbed with power management and other techniques like CS or AIS. Speculative tasks executions occur frequently in map-reduce cluster which result is resource wastage. Using the scheduling algorithms if they can be reduced, the power consumption can be controlled. Thus some research can be done to study how the use of these efficient scheduling algorithms designed to meet SLAs, ensure data locality and control number of speculative tasks execution, can improve Map-reduce energy efficiency. Further in the course new energy aware job and task scheduling algorithms can be developed.

- **Energy consumption and efficiency models for Map-reduce jobs**
  There is a need of detailed energy efficiency model for Map-Reduce environments to predict the energy consumed for mix workload scenarios. It should also consider the background HDFS activities carried out for availability checks. It should be able to incorporate the idle nodes energy as well.

  Map-Reduce performance models can be used to predict the map and reduce task timings depending on the data volume, their distribution, underlying hardware etc. Energy and performance models can be combined to evaluate various scheduling algorithms for predicting the energy consumptions and thus decide which one maximizes the performance and energy efficiency.

- **Power models for Map-reduce tasks**
  For finding the energy consumption and efficiency, power measurements are required. The large-scale map-reduce clusters would require more power measurement equipments adding up to data center costs. Most of the papers have used utilization proportional power models for estimating the power consumed in various exercises. The authors in [NEZ11] measured power using internal set of software tools for each component and used total to show that power consumed. These may have their own overheads as well.

  So for cost and time benefits, instead of actual equipments or software tools, power models can be used. Thus there is a need to develop accurate power models for Hadoop map-reduce tasks. These power models would predict the power consumed by a given task (map or reduce or heartbeat) for a given data size on a machine operating at a frequency.

- **Use of Power management techniques for Map-reduce cluster**
The energy is defined as product of Power and Time. Current work has considered only two power states of the hardware, peak power state and lowest power state. Consequently they have been concentrating on reducing durations of the peak power state to reduce energy consumption and improve energy efficiency. The hardware has evolved overtime to have up to 6 power states. Some of the OS level tools are available which can help control the power consumed by a machine dynamically. The use such methods in Hadoop cluster can be studied. Say idle nodes required only for data availability can be put on lower power states and brought to higher power state whenever a task is scheduled. It can be further extended to use the more fine grained Dynamic Voltage Frequency Scaling methods. So that nodes can be put on a frequency suitable for the type, priority and deadline of task scheduled on it.

- **Power-performance models for Map-reduce tasks**

The power performance models for map-reduce should also be developed. These would be used to predict the performance of different map-reduce jobs when run on different machines at different power frequencies. These models would complement the power models. They can be designed to include the map-reduce cluster and job’s configurations (like number of map/reduce tasks, data split size) to provide higher accuracy predictions. The models should also predict performance for mix workload scenarios. They can provide the input of time taken by various tasks/jobs, at different frequencies on different hardware for varying input data sizes, to energy efficiency models when power management techniques are being used.
5 References


[BEL10] C. Belady. In the data center, power and cooling costs more than the IT equipment it supports. Electronics Cooling Magazine. Feb. 2010.


[SCH08] Presentations by S. Schlosser and J. Lin at Hadoop Summit. 2008. tinyurl.com/4a6lza


[TOM09] Tom Wheeler. Hadoop in 45 minutes or less. OCI 2009.


Acknowledgements

I would like to thank my supervisor, Dr. Umesh Bellur for his directions and valuable inputs for this work.