

# LONGEST APPROXIMATE TIME TO END SCHEDULING ALGORITHM IN HADOOP ENVIRONMENT

R.Thanga selvi,  
M.E: Department of Computer Science,  
VV College of Engineering,  
Tirunelveli, India  
[golda.selvi@gmail.com](mailto:golda.selvi@gmail.com)

R.Aruna,  
M.E: Department of Computer Science,  
Jayamatha Engineering College ,  
Kanyakumari, India  
[arunartr@gmail.com](mailto:arunartr@gmail.com)

**Abstract**— MapReduce has become far and wide for processing large data volume jobs. As the number and deviation of jobs to be executed across different clusters are increasing, so it is the complexity of scheduling them efficiently to meet required objectives of performance. This paper presents a LATE MapReduce scheduling algorithms expected for such complicated scenarios. A taxonomy is furnished for Map-reduce algorithms based on their runtime nature. The algorithms considered for each hierarchical level of MapReduce scheduling are described in detail. Some pointers for future research to improve the efficiency some scheduling techniques are provided. Another aspect of MapReduce is that the size of their clusters is regularly in hundreds and thousands, at the same time it is used for processing infrequent batch and interactive tasks in parallel across these machines. Thus there is a need to detect at energy efficiency of MapReduce clusters. A survey on some of the algorithm is proposed to improve the energy efficiency of MapReduce . The studied techniques have been classified based upon the MapReduce element they work-on. Details of the technique in each category is provided. Few suggestions for future research are given based on the gaps observed in these work.

**Keywords**— MapReduce, Scheduling, LATE, heterogeneous, hadoop, DataNode, NameNode

## 1 . INTRODUCTION

With the current trend in increased use of World Wide Web in everything, lot of data is generated and analyzed. Web search engines and social networking sites discover and segregate every user action on their sites to recover site design, recognize spam and malfeasance, and advertising opportunities. Facebook collects 15 Tera Bytes of data each second into its Peta Byte scale data warehouse [26]. Powerful telescopes in astronomy, genome sequencers in biology, and particle accelerators in physics are churning out huge amounts of data for scientists. Key scientific breakthroughs are proposed to determine computational experiment of such data. Many basic and applied science disciplines rapidly have computational areas, for example computational economics, computational biology, and computational journalism. Given these many use cases, there is a need to preserve improving the BIG Data management techniques. The processing of this can be best done by Distributed computing and parallel processing mechanisms.

Map-reduce is one of the most popularly used such technique. MapReduce breaks a computation into little tasks that run in parallel on multiple machines, and scales easily to indeed large clusters of inexpensive commodity hcomputers. MapReduce is becoming a overall programming model. The best example is Google, which uses its MapReduce context to process 20 petabytes of data per day [7]. Other instance are of Mars [5] that harnessed computer graphics processors power for MapReduce. Hadoop, an open-source MapReduce implementation, has extensively been adopted by industries such as Facebook, and academia. Hadoop [2] is being deployed in many cloud platforms also. Example, Amazon has equipped their software stack with Hadoop to facilitate continually large-scale data applications on Amazon Elastic Cloud Computing [1]. The New York Times rented 100 virtual machines for a day to shift 11 million scanned articles to PDFs [10]. In addition, researchers from University of Maryland and PARC are starting to consider Hadoop for seismic simulation, natural language processing, and web data mining [11, 25]. Due to its great adoption, the attitude of Hadoop in particular (and MapReduce in general) has become an consistent 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 track 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 [8] has indicated that the power consumption of data centers 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 datacenters process 170 petabytes of data on cluster of 38000 servers [3]. Such large clusters are created to support peak workloads. It was observed that Facebook workload had high peak-to-average ratios [29] then the large clusters remain under utilized consuming peak power most of the time leading to energy in-efficiency. 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 [13]. This also causes energy wastage. Over the lifetime of IT equipment, the operating energy cost is comparable to the initial equipment acquisition cost [6] and constitutes a significant part of the TCO of a datacenter [4]. The 3 years TCO for hybrid cluster of 2600 nodes itself has been shown to be around \$15.1million analytically [24]. 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 datacenters, encompassing government reports [8], standardization efforts [9], and research projects in both industry and academia [18, 13, 17, 30, 29]. 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 MAPREDUCE PROGRAMMING MODEL

Most common huge volume data processing programs do counting, sorting, merging etc. Such programs require to perform first 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 [7] 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.

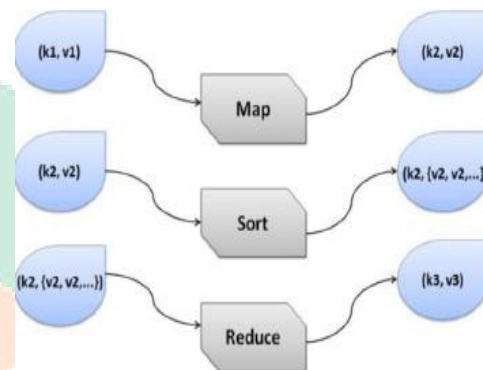


Figure 1: Map-Reduce program workflow.

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.

## 3.OVERVIEW OF HADOOP[ MAPREDUCE SCHEDULING ALGORITHM

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 so set 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.

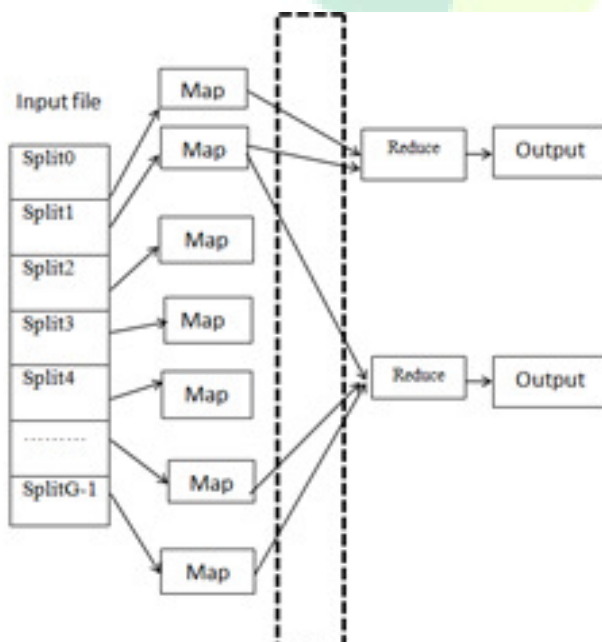


Fig 2: MapReduce framework

### 3.1 FIFO scheduling algorithm

Hadoop by default uses FIFO scheduling algorithm. The main objective of FIFO scheduler to schedule jobs based on their priorities in first-come first serve basis. FIFO stands for first in first out which in it Job Tracker pulls oldest job first from job queue and it doesn't concern about priority or size of the job [5]. FIFO scheduler have many limitations such as: poor response times for short jobs compared to large jobs, Low performance when run multiple types of jobs and it persevere good result only for single type of job. to address these problems round robin scheduling algorithm was introduced.

### 3.2 Round Robin scheduling algorithm

In round robin technique each tasks are given extend priority [11]. Round-robin (RR) is one of the algorithm engrossed by the processors and network schedulers in computing.[1] As the term is commonly used, has a head start slices are sitting each practice in extend portions and in circular sending up the river, handling all processes without priority (also experienced as cyclic executive). Round-robin scheduling is easily done, inconsequential to enforce, and starvation-free. Round-robin scheduling can also be applied to distinctive scheduling problems, such as data packet scheduling in computer networks. It is an Operating System concept. The cast of the algorithm comes from the round-robin principle known from other fields, where each person takes a uniform share of something in turn.

### 3.3 Fair scheduling algorithm

Fair scheduling is a manner of assigning resources to jobs such that all jobs gain, on average, an equal share of resources during time [12]. If there is a single job running, the job uses the entire cluster. When disparate jobs are submitted, free job slots are assigned to the new jobs, so that each job can get closely the same amount of CPU time. It lets short jobs meticulous within a competitive time while not starving for long jobs. The circumstance of Fair scheduling algorithm is to do a equal distribution of compute resources among the users/jobs in the system [17], [12]. The scheduler typically organizes jobs by resource pool, and shares resources fairly among these pools. By default, there is a separate pool for each user. The Fair Scheduler can impose the number of concurrent running jobs via user and per pool. Also, it can inflict the number of concurrent running tasks via pool. The traditional algorithms have high data transfer and the execution time of jobs. Tao et al. [13] introduced an improved FAIR scheduling algorithm, which takes into account job characteristics and data locality, which decreases both data transfer and the execution time of jobs. Thus, Fair scheduling can cover some limitation of FIFO such as: it can works abundantly in both small and large clusters and less complex. Fair scheduling algorithm does not consider the job weight of each node, which this is an important disadvantage of it.

### 3.4 Capacity scheduling algorithm

The study of capacity scheduling algorithm is very redolent to fair scheduling. But used of queues instead of pool. Each queue is allocated to an organization and resources are divided bounded by these queues. Scilicet, Capacity scheduling algorithm puts jobs into multiple queues in accordance with the conditions, and allocates indisputable system capacity for each queue. If a queue has heavy load, it seeks unallocated resources, then makes redundant resources allocated evenly to each job [13], [6]. For maximizing resource utilization, it empowers re-



allocation of resources of free queue to queues via their full capacity. When jobs arrive in that queue, running tasks are completed and resources are given back to original queue. It also empowers priority based scheduling of jobs in an organization queue [3]. The capacity scheduler empowers users or organization to simulate a separate MapReduce cluster with FIFO scheduling for each user or organization [4]. Generally, capacity scheduling algorithm labels the FIFO's limitation such as the low utilization rate of resources.

### 3.5 Weighted Round Robin scheduling algorithm

The weighted round-robin scheduling is introduced to better use servers by the whole of different processing capacities. Each server boot be given away a saddle, an integer worth that indicates the processing capacity. Servers by the whole of higher weights feed new connections willingly than those by all of less weights, and servers with higher weights win more connections than those by all of less weights and servers with admit of comparison with weights earn equal connections. The Weighted Round-Robin (WRR) Scheduling algorithm [14] is based on the round-robin and advantage scheduling algorithms. The WRR retains the body of round-robin in eliminating hunger and by the same token integrates pride of place scheduling. WRR is a pre-emptive algorithm that will realize several techniques. WRR will fit a long in the tooth style, bend to a well-known will the anticipate slice for each fashion contained in each "weight", and furthermore reorder the fashion chain according to "weight". The WRR by the same token will achieve the work of genius of aging. Aging is when a process, that renew the trade queue will be subject to a "bump" in weight. In the status of WRR the algorithm will pick up the load of that engagement in activity application by a figure of one for separately three realized job queue cycles it is processed.

### 3.6 Improved Weighted Round Robin scheduling algorithm

The Improved Round Robin (IRR) scheduling algorithm works similar to Round Robin (RR) with a small improvement [15]. IRR selects the first process from the ready queue and assign the resource to it for a time interval of up to 1 time quantum. After realization of the process's time quantum, it calculates the remaining burst time of the on-going process. If the remaining burst time of the on-going process is lesser than 1 time quantum, the processor further allocated to the on-going process for remaining burst time. In this case this process will finish execution and it will be moved from the ready queue. The scheduler formerly proceeds to the next process in the ready queue. Otherwise, if the remaining burst time of the on-going process is greater than 1 time quantum, the process will be placed at the tail of the ready queue. The scheduler will formerly select the next process in the ready queue.

### 3.7 Hybrid scheduling algorithm

Nguyen et al. [16] proposed a Hybrid Scheduler algorithm based on dynamic priority in order to reduce the delay for variable size concurrent jobs, and recuperate the order of jobs to uphold data locality. Also it provides a user-defined job level value for QoS. This algorithm is designed for data intensive workloads and tries to uphold data locality during job execution [11]. The average response time for the workloads is approximately 2.1x faster over the Hadoop Fairs with a standard deviation of 1.4x. It achieves this improved response time by means of relaxing the strict proportional fairness with a simple exponential policy technique. This algorithm is a fast and flexible scheduler that improves response time for multi-user Hadoop environments.

### 3.8 Self-adaptive Reduce scheduling (SARS) algorithm

Tang et al. [3] intended an optimal reduce scheduling procedure for reduce tasks start time in Hadoop, which called SARS. This procedure works by slow down the reduce processes. The reduce tasks are scheduled, when not all of the map tasks have finished. The purpose of the scheduling algorithm, SARS is to shorten the copy duration of the reduce process, decrease the task complete time, and save the reduce slots resources. Due to The experimental results in [3], a time when comparing SARS with the FIFO, the reduce completion time is decreased sharply. And it is further proved that the average response time are diminished 11% to 29% when SARS algorithms are applied traditional job scheduling algorithm: FIFO, Fair Scheduler and Capacity Scheduler. The limitation of this algorithm is that only focus on reduce process.

## 4. PROPOSED METHODOLOGY

The proposed Longest Approximate Time to End (LATE) algorithm 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  $\text{ProgressScore} = \text{ExecutionTime}$ . The time left parameter for a task is estimated based on the Progress Score provided by Hadoop, as  $(1 - \text{ProgressScore}) / \text{ProgressRate}$ .

The LATE algorithm works as shown in Algorithm 1.

Algorithm 1 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 Hadoops 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 Hadoops 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.

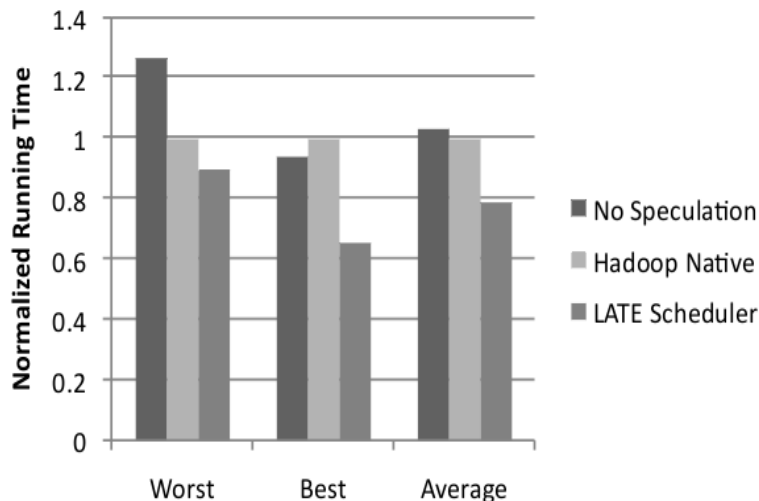


Fig 3: LATE Running time

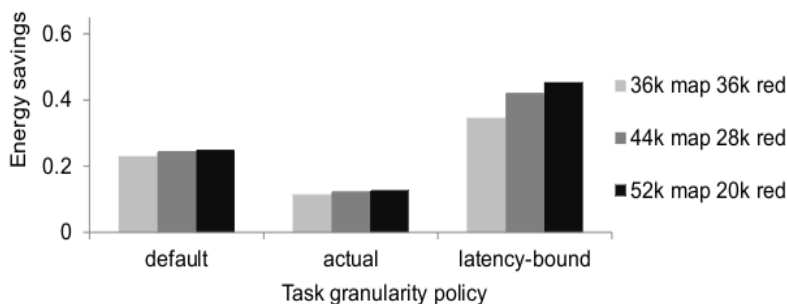


Fig 4: Energy savings at different values of mapreduce ratio

## 5. CONCLUSION

In this paper, we have evaluated the performance of our approach LATE (Longest Approximate Time to End) scheduling algorithm which improves the performance of hadoop. It works better than existing map reduce scheduling

algorithms by taking less amount of computation and gives high accuracy. The results demonstrate that the algorithm is both more accurate and efficient in comparison to other algorithms in literature. In the future we have planned to further improve the efficiency by reducing the execution time.

## 6. REFERENCES

- [1] Amazon Elastic Compute Cloud (EC2). <http://aws.amazon.com/ec2/>.
- [2] The Apache Hadoop Project. <http://www.hadoop.org>.
- [3] E. Baldeschwieler. <http://developer.yahoo.com/events/hadoopsummit2010/agenda.html>.
- [4] C. Belady. In the data center, power and cooling costs more than the IT equipment it supports. Electronics Cooling Magazine, Feb. 2010.
- [5] B. He, W. Fang, N. K. Govindaraju, Q. Luo, and T. Wang. Mars: A MapReduce framework on graphics processors. In Proc. 17th Int'l Conference on Parallel Architectures and Compilation Techniques, pp. 260-269, 2008.
- [6] Y. Chen, A. Ganapathi, A. Fox, R. H. Katz, and D. A. Patterson. Statistical workloads for energy efficient mapreduce. Technical report, UC, Berkeley, 2010.
- [7] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In Communications of the ACM, 51 (1): 107-113, 2008.
- [8] U.S. Environmental Protection Agency. Report to Congress on Server and Data Center Energy Efficiency, Public Law 109-431, 2007.
- [9] The Green Grid. The Green Grid Data Center Power Efficiency Metrics: PUE and DcIe. 2007.
- [10] D. Gottfrid. Self-service, Prorated Super Computing Fun!. <http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/>.
- [11] Applications powered by Hadoop. <http://wiki.apache.org/hadoop/PoweredBy>.
- [12] J. Hamilton. Overall Data Center Costs. In Perspectives. <http://perspectives.mvdirona.com/>. September 18, 2010.
- [13] Jacob Leverich, Christos Kozyrakis. On the Energy (In)efficiency of Hadoop Clusters. In ACM SIGOPS Operating Systems Review. Volume 44 Issue 1. Pages 61-65. 2010.
- [14] J. Jeery Hanson. An introduction to the Hadoop Distributed File System. IBM DeveloperWorks, 2011. <http://www.ibm.com/developerworks/web/library/wa-introhdhfs/index.html>
- [15] Jordà Polo, David de Nadal, David Carrera, Yolanda Becerra, Vicenc Beltran, Jordi Torres and Eduard Ayguade. Adaptive Task Scheduling for MultiJob MapReduce Environments. In Proceedings of Jornadas de Paralelismo conference (JP), pp 96-101A Corua. 2009.
- [16] L. A. Barroso and U. H. Izle. The Case for Energy-Proportional Computing. IEEE Computer, Vol 40 (no. 12), pages 33-37. 2007.
- [17] Kamal Kc and Kemafor Anyanwu. Scheduling hadoop jobs to meet deadlines. In Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science. CLOUDCOM '10. pages 388-392. IEEE Computer Society, 2010.
- [18] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, Ion Stoica. Improving MapReduce performance in

heterogeneous environments. In Proceedings of the 8th USENIX conference on Operating systems design and implementation, p.29-42, December, 2008.

[19] Mohammad Hammoud, M. Suhail Rehman, and Majd F. Sakr . Center of Gravity reduce task scheduling to lower Map-Reduce network trac. In Proceedings of IEEE 5th International Conference on Cloud Computing (IEEE Cloud). Pages 49-58, 2012.

[20] Nan Zhu, Lei Rao, Xue Liu , Jie Liu, Haibin Guan. Taming Power Peaks in MapReduce Clusters. In Proceedings of the ACM SIGCOMM conference. Pages 416-417, 2011.

[21] Nedeljko Vasic, Martin Barisits , Vincent Salzgeber and Dejan Kostic . Making Cluster Applications Energy-Aware. In ACM Proceedings of the 1st workshop on Automated control for datacenters and clouds (ACDC). Pages 37-42, 2009.

[22] Nezih Yigitbasi , Kushal Datta, Nilesh Jain, and Theodore Willke . Energy E-fficient Scheduling of MapReduce Workloads on Heterogeneous Clusters. Proceed-ings of the 2nd International Workshop on Green Computing Middleware. Pages 1-6, 2011.

[23] Nitesh Maheshwari, Radheshyam Nanduri, Vasudeva Varma . Dynamic Energy Efficient Data Placement and Cluster Reconfiguration Algorithm for MapReduce Framework. Future Generation Computer System (FGCS) Journal. Vol. 28, No. 1. Pages 119-127. 2012.

[24] Rini T. Kaushik , Milind Bhandarkar . GreenHDFS: Towards An Energy-Conserving, Storage-Ecient, Hybrid Hadoop Compute Cluster. In Proceedings of the International conference on Power aware computing and systems (HotPower). Pages 1-9. 2010.

[25] Presentations by S. Schlosser and J. Lin at Hadoop Summit. 2008.  
[tinyurl.com/4a6lza](http://tinyurl.com/4a6lza)

[26] A. Thusoo et al. Hive - a warehousing solution over a map-reduce framework. PVLDB, 2(2):16261629, 2009.

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

[28] Willis Lang, Jignesh M. Patel . Energy Management for MapReduce Clusters. In Proceedings of the VLDB Endowment. Volume 3 Issue 1-2. Pages 129-139. 2010.

[29] Yanpei Chen, Sara Alspaugh, Dhruba Borthakur , Randy Katz. Energy Eciency for Large-Scale MapReduce Workloads with Signicant Interactive Analysis. In Proceedings of the 7th ACM european conference on Computer Systems (Eurosys). Pages 43-56, 2012.

[30] Zhenhua Guo, Georey Fox, Mo Zhou, Yang Ruan. Improving Resource Utilization in MapReduce. Indiana University Report. May 2012.