An Energy-Efficient Greedy MapReduce Scheduler for Heterogeneous Hadoop YARN Cluster

Vaibhav Pandey(✉) and Poonam Saini

Department of CSE, Punjab Engineering College (Deemed to be University), Chandigarh, India
pandeyvaibhav51@gmail.com, nit.sainipoonam@gmail.com

Abstract. Energy efficiency of a MapReduce system has become an essential part of infrastructure management in the field of big data analytics. Here, Hadoop scheduler plays a vital role in order to ensure the energy efficiency of the system. A handful of MapReduce scheduling algorithms have been proposed in the literature for slot-based Hadoop system (*i.e.*, Hadoop 0.*x* and Hadoop 1.*x*) to minimize the overall energy consumption. However, YARN-based Hadoop schedulers have not been discussed much in the literature. In this paper, we design a scheduling model for Hadoop YARN architecture and formulate the energy efficient scheduling problem as an Integer Program. To solve the problem, we propose a *Greedy* scheduler which selects the best job with minimum energy consumption in each iteration. We evaluate the performance of the proposed algorithm against the FAIR and Capacity schedulers and find out that our greedy scheduler shows better results for both CPU- and I/O intensive workloads.

Keywords: MapReduce · Scheduling · Energy-efficiency

1 Introduction

Hadoop MapReduce is the most prevalent distributed computing framework inspired by Google's MapReduce programming paradigm. The framework was initially used for long batch processing of production jobs which are executed periodically in data centers. Essentially Hadoop is an ecosystem and MapReduce is one of its core components. The other components include HDFS [1], Pig [2], Hive [3], Mahout [4] and ZooKeeper [5] etc. Four versions of Hadoop framework have been released since inception, namely, Hadoop 0.*x*, Hadoop 1.*x*, Hadoop 2.*x* and Hadoop 3.*x*. The slot-based resource management is used in Hadoop 0.*x* and Hadoop 1.*x*. On the other hand, Hadoop 2.*x* and Hadoop 3.*x* use a fine-grained container based resource management system known as YARN.

The performance of Hadoop is greatly influenced by its scheduler. Initially, the main purpose of Hadoop to run large batch jobs such as web indexing and log mining. Hence, users submit jobs to a queue, and the Hadoop execute the jobs in FIFO order. However, the amount of data has increased substantially in Hadoop clusters that require

A. Mondal et al. (Eds.): BDA 2018, LNCS 11297, pp. 282–291, 2018.
https://doi.org/10.1007/978-3-030-04780-1_19

various complex algorithms to be executed faster. In the view of this, a MapReduce cluster is being shared among multiple users for a variety of workload. In a shared environment, the scheduler first selects a user who generally submits multiple jobs of different characteristics for execution. Hence, job selection is to be done next. Once the job is selected, its map, reduce, or speculative tasks are scheduled for further execution. It is not always true that a single scheduler schedules all three entities. At times, the scheduler allocates resources to one or more entities depending upon the scheduling policy being used. In July 2008, the scheduler in Hadoop became a pluggable component and triggered the innovation in this domain.

A MapReduce scheduler has to cater various quality of service (QoS) requirements of two stakeholders, namely, *Hadoop user* and *Hadoop administrator*. The QoS parameter includes makespan, response time, availability, throughput, energy efficiency, security and resource utilization etc. Here, energy efficiency is important for Hadoop system administrator in today's era of *Green Computing*. The U.S. Department of Energy stated in its report that data centers consumed about 70 billion kilowatt-hours of electricity in 2014 [6]. There is an increase in data center electricity consumption by about 4% from 2010–2014, a large shift from the 24% percent increase estimated from 2005–2010 and the nearly 90% increase estimated from 2000–2005. The Hadoop MapReduce framework is widely used in data centers to analyze the huge amount of data. Thus, there is a need to consider the energy efficiency of MapReduce clusters by designing energy-aware scheduling techniques to minimize energy consumption in a Hadoop system.

In this paper, we formulate an energy efficient MapReduce scheduling problem for YARN-based Hadoop framework as an Integer Program (IP). As the formulated problem is NP-hard, we propose a greedy technique which allocates a task with minimum energy consumption on a particular node. The rest of the paper is organized as follows. Section 2 presents a short survey of related work in the area of energy efficient scheduling in Hadoop framework. In Sect. 3 we prepare a scheduling model for YARN architecture and formulated an energy efficient scheduling problem. We then propose a greedy scheduler for the formulated problem in Sect. 4 and evaluate the performance in Sect. 5. Lastly, Sect. 6 concludes the paper.

2 Related Work

Energy-efficient job scheduling techniques help to reduce the energy consumption in a Hadoop system which can be further classified into two categories. In the first category, the partial or whole cluster is usually kept on low power state whenever not in use in order to reduce energy consumption. In the second category, map and reduce tasks are placed on appropriate nodes to achieve better energy efficiency. Few of the techniques [7, 8], that combine DVFS technique with task scheduling yields better results. A brief discussion of energy efficient MapReduce scheduling techniques is as follows.

Leverich et al. [9] identified that cluster nodes might remain idle for 20–38% of the time and hence proposed a method for energy management of MapReduce jobs by selectively powering down nodes with low utilization. Further, using idle and low utilization periods of cluster nodes, Lang et al. [10] proposed workload energy aware

All-in-Strategy (AIS). AIS use 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.

Chen et al. [11] designed an energy-efficient Map-Reduce workload manager called Berkeley Energy-Efficient MapReduce (BEEMR). In the scheme, the cluster is split into two zones i.e., a small interactive zone and a larger batch zone, each having different percentages of available capacity in terms of task slots, memory, disk and network. The interactive zone is always in full power state, while the batch zone oscillates between full and low power state. BEEMR segregates interactive and batch workloads into separate sub-clusters to improve energy efficiency.

Yigitbasi et al. [12] proposed an energy-efficient algorithm for scheduling heterogeneous workload to the heterogeneous cluster consisting of high and low power machines. This provides an opportunity to save energy by intelligently placing jobs on its corresponding energy-efficient machine. In case the available node does not match the energy efficiency threshold, fairness and data locality criteria are used to select a job and its task respectively for scheduling.

Mashayekhy et al. [13] proposed a framework for improving the energy efficiency of MapReduce applications over heterogeneous machines while satisfying the service level agreement (SLA). The problem of energy-aware scheduling of a single MapReduce job has been modeled as an Integer Program called Energy-aware MapReduce Scheduling (EMRS-IP) with a deadline as a constraint. Two heuristic algorithms have been proposed to solve the EMRS-IP, called energy-aware MapReduce scheduling algorithms (EMRSA-I and EMRSA-II). Both of these heuristics take the energy efficiency differences of different machines into account and use a metric called energy consumption rate of the map and reduce slots that characterize the energy consumption of each machine and induces an order relation among the machines.

Bampis et al. [8] proposed energy-efficient scheduling algorithms for two different scenarios to minimize the weighted completion time of a set of n MapReduce job with a constraint of the energy budget. In the first scenario, the authors formulated the problem as a linear program assuming that the order of the job execution is not fixed. A polynomial time constant-factor approximation algorithm was derived to solve the formulated problem. In the second scenario, the scheduling problem has been formulated as a convex program with the order of jobs is given.

Cai et al. [7] proposed a YARN scheduler to minimize the energy consumption with a deadline as a constraint unlike [8]. The proposed scheduler works at both, job level and task level. At the job level, the scheduler in highly inspired by ARIA [14] and competes for the jobs within its deadline. At the task level, energy efficiency has been targeted through the user-space DVFS governor. While scheduling at the task level, energy consumption of tasks at a specific node has not been considered as in [13].

3 System Modeling and Problem Formulation

A MapReduce job comprises a specific number of map and reduce tasks that are executed on a cluster composed of multiple machines. Cluster may consist of a different generation of machines implying heterogeneous hardware. The execution of jobs

consists of a map phase followed by a reduce phase. In YARN setting, a task (either map or reduce) may request a fixed amount of different resources available at a machine for its execution. Multiple types of resources on each machine are allocated to tasks in the form container. Each task request in YARN cluster is usually represented by a tuple <p, r, n, l, b>, where p represents the priority of a task, r gives the resource requirement vector of a task, n is the total number of tasks which have the same resource requirements r, l represents the location of a task's input data split, and b is a boolean value to indicate whether a task can be assigned to a *NodeManager* that does not have its input data split locally.

We consider a set of N MapReduce jobs $J = \{J_1, J_2, \ldots\ldots J_N\}$, which is submitted to a YARN cluster $S = \{S_1, S_2, \ldots\ldots S_M\}$ consisting of M machines. Each MapReduce job J_j consists of two distinct sets of the map and reduce tasks and precisely defined as $J_j = MT_j \cup RT_j$, where $MT_j = \left\{m_1^j, m_2^j, m_3^j, \ldots\ldots m_{m'}^j\right\}$ is a set of m' map tasks and $RT_j = \left\{r_1^j, r_2^j, r_3^j, \ldots\ldots r_{r'}^j\right\}$ is a set of r' reduce tasks. The task m_i^j represents the i^{th} map task of j^{th} job and similarly the task r_i^j represents the i^{th} reduce task of j^{th} job. Let the execution start time and processing time of map task m_i^j on k^{th} machine is sm_i^j and pm_{ik}^j respectively. Similarly, execution start time and processing time of reduce r_i^j on k^{th} machine is sr_i^j and pr_{ik}^j respectively. It is to be noted that execution start time of a map and reduce task is independent of machine on which it is scheduled.

Furthermore, we assume that K types of resources are available at each machine represented by r_1, r_2, ……. r_K. A two-dimensional matrix A of size $M \times K$ is used to represent the current availability of resources at each node. $A[i, j]$ indicates the total amount of j^{th} resource type r_j available at i^{th} machine S_i at a particular time instance $\emptyset$.

A matrix M of size $N \times K$ is also defined to store the amount of each resource type r_j required by map tasks of each job. The value $M[i, j]$ indicates the amount of resource type r_j requested by map tasks of i^{th} job for its execution. Similarly, a matrix R of size $N \times K$ is also defined to store the amount of each resource type r_j required by reduce tasks of each job and value $R[i, j]$ indicates the amount of resource type r_j requested by reduce tasks of an i^{th} job for its execution.

The YARN scheduler can assign a map task of i^{th} job to a worker node S_j for execution as long as $M[i,p] \leq A[j,p], \forall p \in [1,K]$. Similarly, the scheduler can assign a map task of i^{th} job to a worker node S_j for execution as long as $M[i,p] \leq A[j,p], \forall p \in [1,K]$.

We consider that the machines are heterogeneous and em_{ik}^j and er_{ik}^j represents the energy consumption of map task m_i^j and reduce task r_i^j respectively at machine S_k during the execution. We aim to minimize both, the energy consumption of whole YARN cluster and the completion time of the execution of all n jobs (i.e., makespan). We formulate the following energy-efficient MapReduce scheduling problem in YARN cluster as an Integer Program (IP).

$$\textbf{\textit{minimize}} \sum_{i=1}^{m'} \sum_{k\in[1,M]} em_{ik}^j X_{ik}^j + \sum_{i=1}^{r'} \sum_{k\in[1,M]} er_{ik}^j Y_{ik}^j, \quad \forall j \in [1,N]$$

$$\textbf{\textit{minimize}} \ \ max \sum_{i=1}^{r'} \sum_{k\in[1,s]} (sr_i^j + pr_{ik}^j) Y_{ik}^j, \forall j \in [1,n]$$

Subject to:

$$\max\{(sm_i^j + pm_{ik}^j)X_{ik}^j\} \geq sr_{i'}^j Y_{i'k}^j, \forall j \in [1,N], \forall i \in [1,m'], \forall i' \in [1,r'], \forall k \in [1,M] \quad (1)$$

$$\sum\nolimits_{i,i' \in J(\varnothing)} M[i,p]X_{ik}^j + R[i',p]Y_{i'k}^j \leq A[j,p], p \in [1,K], j \in [1,M] \quad (2)$$

$$X_{ik}^j = \{0,1\}, \forall j \in [1,N], \forall i \in [1,m'], \forall k \in [1,M] \quad (3)$$

$$Y_{ik}^j = \{0,1\}, \forall j \in [1,N], \forall i \in [1,r'], \forall k \in [1,M] \quad (4)$$

$$sm_i^j, sr_i^j \geq 0 \quad (5)$$

where $J(\varnothing)$ is a set of active jobs at particular time instance $\varnothing$.

In this formulation, there are two objective functions. First objective function minimizes the energy consumption while executing the MapReduce application whereas, the second objective minimizes the completion time of all jobs. Constraint (1) ensures that for all MapReduce jobs, reduce tasks start only when all map tasks have been completed. Constraint (2) requires that the resources consumed by all active tasks at a particular worker node S_j cannot exceed its resource capacity. Constraints (3) and (4) represents the integrality requirement for the decision variable. The decision variable X_{ik}^j takes the value 1 if map task m_i^j is assigned to machine S_k and 0 otherwise. Similarly Y_{ik}^j takes the value 1 if reduce task r_i^j is assigned to machine S_k and 0 otherwise. Lastly, constraint (5) requires decision variables sm_i^j and sr_i^j to be non-negative.

4 Proposed Solution

The MapReduce scheduling problem formulated in the previous section is NP-hard Integer Programming (IP) problem. There are mainly three approaches to solve NP-hard IPs. First one is Heuristic approach which solves the IPs usually in polynomial time with a sub-optimal result without any guarantee on the sub-optimality. The second one is known as approximation algorithms which provides a sub-optimal result with an assurance on the quality of the sub-optimal result. Finally, the third approach solves the IPs optimally, however, takes exponential time e.g., Branch and Bound (B&B), Branch and Cut, and cutting plane methods. We opt the first approach and use the energy consumption of a task on a particular machine as a heuristic. We further use that heuristic to greedily select a task to assign to a particular machine. The energy efficient *greedy* approach minimizes the total energy consumption of the set of n jobs. We now explain our proposed approach in detail.

4.1 Energy Efficient *Greedy* Approach

We propose a greedy approach without aiming makespan to schedule map and reduce task over worker nodes. The scheduler is triggered whenever it receives the heartbeat

message from any worker node *N* which is running the *NodeManager* demon. Upon receiving the heartbeat message, scheduler groups all jobs in two sets: NodeLocal_Jobs and RackLocal_Jobs. Jobs in the set NodeLocal_Jobs have a local copy of desired data split at node N, whereas, jobs in RackLocal_Jobs have a copy of data split at a different node which is in the same rack that of node N.

After grouping the jobs in two sets, the algorithm greedily selects a job first from the set NodeLocal_Jobs. Precisely, it picks a job which has the least energy consumption on node *N* and assigns as much map tasks as the available capacity of the node. If no local job exists then, the algorithm searches the set RackLocal_Jobs and greedily selects a job which has least energy consumption on node *N*. By this grouping of jobs, the proposed scheduling algorithm gives preference to NodeLocal_Jobs to achieve better data locality. This further improves energy efficiency because if a task gets its data from the local node, it does not need to perform energy consuming network I/O operation to move data split to its local node. The pseudo code of the proposed greedy approach for map tasks is given in Algorithm 1.

Algorithm 1:

```
Upon receiving a heartbeat from node N:
NodeLocal_Jobs ← GetJobsNodeLocal(J,N)
RackLocal_Jobs ← GetJobsNodeLocal(J,N)
while NodeLocal_Jobs ≠ NULL
    Best_Job ← SelectBestJob(NodeLocal_Jobs, N)
    NodeLocal_Jobs ← NodeLocal_Jobs – {Best_Job}
    TaskAssignment(Best_Job, N)
    If node N still has resource capacity to     a
    commodate new tasks
    then continue
while RackLocal_Jobs ≠ NULL
    Best_Job ← SelectBestJob(RackLocal_Jobs, N)
    RackLocal_Jobs ← RackLocal_Jobs – {Best_Job}
    TaskAssignment(Best_Job, N)
    If node N still has resource capacity to
     accommodate new tasks
    then continue
SelectBestJob(JobsList L, Node N)
    return argmin_{j∈ L} { em_iN^j }
TaskAssignment(Job J, Node N)
    Assign as much map tasks of job J as the total
    remaining capacity of node N
```

In case of reduce tasks, the algorithm does not consider any data locality metric while assigning tasks. Data locality for reduce tasks is hard to achieve because reduce task receives the output of various map tasks running on many different nodes. For assigning the reduce task, the algorithm considers the whole job set and pics the best job which has the least energy consumption on the node from which heartbeat message is received. The pseudo code of the proposed greedy approach for reduce task is given in Algorithm 2.

Algorithm 2:
Upon receiving a heartbeat from node N:
Let R_Jobs be set of jobs whose map tasks are finished
Best_Job ← SelectBestJob(R_Jobs, N)
TaskAssignment(Best_Job, N)
SelectBestJob(JobsList L, Node N)
return $\text{argmin}_{j \in L} \{em_{iN}^{j}\}$
TaskAssignment(Job J, Node N)
Assign as much map tasks of job J as the total remaining capacity of node N

5 Experimental Results and Discussions

In this section, we will evaluate the performance of proposed energy efficient greedy scheduler on the basis of total energy consumption and completion time and compare the results with FAIR and Capacity schedulers. Before illustrating the final results, we mention the cluster configurations, workload and energy model used in our experiments.

5.1 Cluster Setup

We use a heterogeneous YARN cluster which is composed of four high-end machines. We use three different configurations of the machine to build the cluster. Two of the nodes have 12 cores of 2.6 GHz (Intel Xeon E5-2690 v3) with 32 GB of RAM and 1 TB of hard disk. The third node has 4 core of 3.5 GHz (Intel Xeon E3-1270 v3), with 16 GB RAM and 500 GB of HDD and the fourth one has 4 cores of 3.8 GHz (Intel Xeon E3-1270 v6), with 8 GB RAM, and 500 GB of HDD. Energy consumption of various components in these nodes are shown in Table 1.

Table 1. Energy consumption of various components in cluster nodes

Node type	CPU power (*CPU_POW*)	Disk read/write power (*DISK_POW*)	NIC I/O power (*NIC_POW*)
Intel Xeon E5-2690 v3	135 W	4.5 W	1.2 W
Intel Xeon E3-1270 v3	80 W	3 W	1.2 W
Intel Xeon E3-1270 v6	72 W	3 W	0.8 W

5.2 Workload Mix Used

We use two different workloads; (i) CPU- intensive and (ii) I/O-intensive. For CPU-intensive workload, we use WordCount job which counts the frequency of words in a text file. Whereas, for I/O intensive workload we use TeraSort job which sorts the terabytes of numbers.

5.3 Energy Model Used for Profiling

We assume that energy consumption of map/reduce tasks at every node in the cluster are known to the proposed algorithm in advance that can be easily measured by energy profiling. In order to profile WordCount and Terasort jobs, we run the single job on every node multiple time and take mean of all recorded values. We note that certain activity like CPU processing, Disk I/O and network I/O consumes energy during the execution of any map and reduce task. In view of this, we devise the Eq. 6 for calculating the energy consumption of a map task. Energy consumption of reduce task can be measured in the same manner.

$$em_{ik}^{j} = C_{ik}^{j} * CPU\_POW_k + D_{ik}^{j} * DISK\_POW_k + N_{ik}^{j} * NIC\_POW_k \tag{6}$$

Where symbols have the following interpretations.

Symbol	Meaning
C_{ik}^{j}	CPU time in ms
CPU_POW_k	CPU power in watt
D_{ik}^{j}	Total disk input/output in bytes
$DISK\_POW_k$	Disk power consumption/byte read or written
N_{ik}^{j}	Number of shuffle bytes
NIC_POW_k	NIC power consumption/byte sent or received

5.4 Parameters Used for Evaluation

We use total energy consumption and completion time of all jobs (i.e., makespan) for comparing the greedy scheduler with FAIR and Capacity schedulers. We use both workloads for each of these evaluation parameters. During the experiments, we gradually increase the workload size for 5 jobs to 15 jobs. For each job, the input file size is kept 4 GB and block size at HDFS layer is kept 128 MB with a replication factor of 3. This creates 32 map tasks for each job and no of reducers explicitly set as 4.

5.5 Results

We perform two sets of experiments: one for measuring energy efficiency and another for measuring completion time. For energy efficiency experiments we perform a test on both workloads separately.

Energy Efficiency

We first evaluate the performance of the proposed scheme on the basis of total energy consumed during the execution of a set of jobs. We perform three sets of experiments with 5, 10 and 15 jobs individually for WordCount and TeraSort jobs. These experiments show (Figs. 1 and 2) that proposed Greedy scheme performs 15% and 17% better on average in comparison to FAIR and Capacity scheduling algorithm respectively for I/O intensive workload. In the case of CPU intensive workload, the proposed scheme performs up to 16% and 20% better in comparison to FAIR and Capacity algorithm respectively.

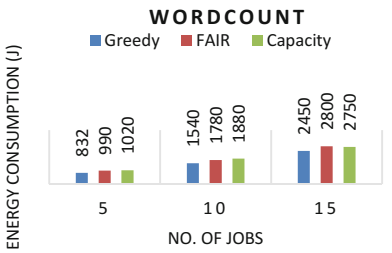

Fig. 1. Energy consumption of jobs for CPU-intensive workload

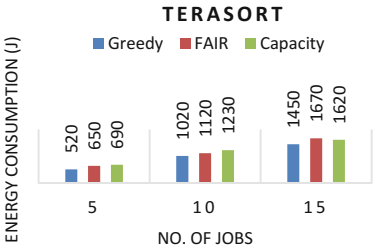

Fig. 2. Energy consumption of jobs for I/O-intensive workload

Completion Time

We further evaluate the performance of the proposed algorithm on the basis of completion time of all jobs. Here, in Figs. 3 and 4, we see that for both CPU- and I/O intensive workload, the greedy scheduler does not achieve better completion time than FAIR and Capacity scheduler. The reason for this degradation is due to consideration of only energy fitness metric during task assignment by the proposed greedy scheme. Hence we conclude that our greedy approach achieves better energy efficiency at the cost of job completion time. This motivates us to include the time parameter in our heuristic besides energy consumption.

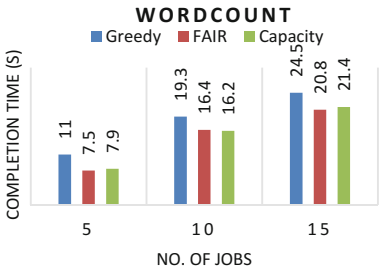

Fig. 3. Completion time of all jobs for I/O-intensive workload

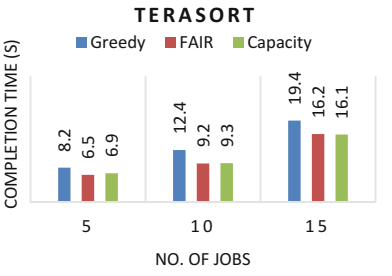

Fig. 4. Completion time of all jobs for I/O-intensive workload

6 Conclusion

Energy efficient scheduler in YARN system is a critical component in the era of Green Computing. In this paper, we presented a scheduling model for YARN architecture and formulated an energy efficient scheduling problem as an Integer Program at task level of scheduling hierarchy. We proposed a greedy scheduler which selects the best job in terms of energy efficiency upon receiving the heartbeat message. Experimental results show better energy consumption performance of the proposed scheme in comparison to both FAIR and Capacity scheduler. In future work, we aim to develop a scheduler which minimizes the energy consumption and completion time of the last job.

Acknowledgment. Authors would like to thank Ministry of Electronics and IT, Govt. of India for providing financial support to perform this work under the Visvesvaraya Ph.D. scheme.

References

1. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010 (2010)
2. Welcome to Apache Pig! https://pig.apache.org/. Accessed 25 June 2018
3. Apache Hive TM. https://hive.apache.org/. Accessed 25 June 2018
4. Apache Mahout: Scalable machine learning and data mining. http://mahout.apache.org/. Accessed 25 June 2018
5. ZooKeeper. https://zookeeper.apache.org/doc/trunk/zookeeperOver.html. Accessed 25 June 2018
6. Shehabi, A., et al.: United States Data Center Energy Usage Report, June 2016
7. Cai, X., Li, F., Li, P., Ju, L., Jia, Z.: SLA-aware energy-efficient scheduling scheme for Hadoop YARN. J. Supercomput. **73**(8), 3526–3546 (2017)
8. Bampis, E., Chau, V., Letsios, D., Lucarelli, G., Milis, I., Zois, G.: Energy efficient scheduling of MapReduce jobs. In: Silva, F., Dutra, I., Santos Costa, V. (eds.) Euro-Par 2014. LNCS, vol. 8632, pp. 198–209. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09873-9_17
9. Leverich, J., Kozyrakis, C.: On the energy (in)efficiency of Hadoop clusters. ACM SIGOPS Oper. Syst. Rev. **44**(1), 61 (2010)
10. Lang, W., Patel, J.M.: Energy management for MapReduce clusters. Proc. VLDB Endow. **3** (1–2), 129–139 (2010)
11. Chen, Y., Alspaugh, S., Borthakur, D., Katz, R.: Energy efficiency for large-scale MapReduce workloads with significant interactive analysis. In: Proceedings of the 7th ACM European Conference on Computer Systems – EuroSys 2012, p. 43 (2012)
12. Yigitbasi, N., Datta, K., Jain, N., Willke, T.: Energy efficient scheduling of MapReduce workloads on heterogeneous clusters. In: Green Computing Middleware on Proceedings of the 2nd International Workshop – GCM 2011, pp. 1–6 (2011)
13. Mashayekhy, L., Nejad, M.M., Grosu, D., Zhang, Q., Shi, W.: Energy-aware scheduling of MapReduce jobs for big data applications. IEEE Trans. Parallel Distrib. Syst. (1), 1 (2015)
14. Verma, A., Cherkasova, L., Campbell, R.H.: ARIA: automatic resource inference and allocation for MapReduce environments. In: Proceedings of the 8th ACM International Conference on Autonomic Computing - ICAC 2011, p. 235 (2011)