GMC: Greening MapReduce Clusters Considering both Computational Energy and Cooling Energy

Tarik Reza Toha[1], Mohammad M. R. Lunar[2], A. S. M Rizvi[3], Novia Nurain[4], and A. B. M. Alim Al Islam[5]
[1,4,5]Department of CSE, Bangladesh University of Engineering and Technology, Bangladesh
[2]Department of CSE, University of Nebraska-Lincoln, USA
[3]Department of CS, University of Southern California, USA
Email: [1]1205082.trt@ugrad.cse.buet.ac.bd, [2]mlunar@cse.unl.edu, [3]asmrizvi@usc.edu, [4]novia@cse.uiu.ac.bd, [5]alim_razi@cse.buet.ac.bd

***Abstract*—Increased processing power of MapReduce clusters generally enhances performance and availability at the cost of substantial energy consumption that often incurs higher operational costs (e.g., electricity bills) and negative environmental impacts (e.g., carbon dioxide emissions). There exist a few greening methods for computing clusters in the literature that focus mainly on computational energy consumption leaving cooling energy, which occupies a significant portion of the total energy consumed by the clusters. To this extent, in this paper, we propose a machine learning based approach named as Green MapReduce Cluster (GMC) that reduces the total energy consumption of a MapReduce cluster considering both computational energy and cooling energy. GMC predicts the number of machines that results in minimum total energy consumption. We perform the prediction through applying different machine learning techniques over year-long data collected from a real setup. We evaluate performance of GMC over a real testbed. Our evaluation reveals that GMC reduces total energy consumption by up to 47% compared to other alternatives while experiencing marginal throughput degradation in a few cases.**

I. Introduction

The emergence of computationally intensive applications paves the way for widespread deployment of cluster computing. For example, real-life applications such as traffic jam simulation deals with huge sample spaces of vehicles, roads, etc., demanding rigorous computations [1]. In order to address such rigorous computations, clusters of machines are often exploited [2]. In order to parallelize and distribute jobs across a cluster, a programming paradigm named as MapReduce is increasingly being used in recent times [3]. Due to the increasing deployment of MapReduce clusters, it is of utmost importance to minimize total energy consumption of clusters.

Existing studies in the literature mainly focus on reducing the energy consumption of clouds. However, the service characteristics of a cloud system is different from the cluster system [4]. Besides, most of the existing studies consider only computational power [5]–[7]. However, according to the study in [8], cooling energy occupies a significant portion of total energy consumption in parallel and distributed systems encompassing clusters. Research on energy saving schemes considering cooling power for cluster is still at an embryonic stage. There exist a few studies on the energy efficiency of clusters [9], [10]. However, similar to the earlier case, these studies focus on managing only computational power leaving the cooling power unexplored.

Therefore, to fill the gap in the literature, in this paper, we propose a new approach to reduce total energy consumption of a MapReduce cluster considering both computational energy and cooling energy. To do so, first, we predict the number of machines that results in minimum total energy consumption. We perform the prediction task through model fitting over a data set on total energy consumption collected from a real setup having different number of machines. Our method realizes different environmental and weather conditions through being fitted over a year-long collected real data. We evaluate efficacy of our proposed method against two state-of-the-art greening methods [7], [10] and an energy-oblivious method, i.e., static method [7]. Our method outperforms all the approaches in terms of minimizing total energy consumption. Based on our work, we make the following set of contributions: 1) We utilize machine learning algorithms over a year-long collected real data to predict response time, computational power, and cooling power, which are required to calculate total energy consumption in a cluster. K-Nearest Neighbor, Support Vector Regression, and Additive Regression show highest accuracies for these prediction tasks respectively exhibiting an overall accuracy of 97% for the prediction of total energy consumption. 2) Based on the predictions, we propose a method to determine the number of machines that are needed to be activated for consumption of a minimum total energy. We name our proposed method as GMC. 3) We compare the performance of our GMC framework with that of three different state-of-the-art methods. Our comparison reveals that GMC reduces total energy consumption by up to 47% at the expense of marginal throughput degradation only in a few cases.

Section II discusses the works related to this research. We explain the detail of our approach in section III, and the necessary testbed setups in section IV. The results are presented in section V followed by conclusion in section VI.

II. Related Work

The power hungriness of distributed systems motivates researchers to invest their efforts on energy efficiency in addition to performance improvement. However, most of them focus on cloud computing environment having a little focus on cluster computing environment. Since clouds possess different architectures from clusters [11], solutions for clouds are not generally applicable for the cluster environment.

Most of the cloud based approaches minimize computational energy by maintaining the performance within a certain level. For example, Guenter et al., [6] design an automatically-provisioned servers to meet workload demand while minimizing the energy cost and the reliability cost due to duty cycling. Chen et al., [5] propose energy-aware server provisioning and load dispatching for connection intensive Internet services. Zhang et al., [7] propose a dynamic energy aware capacity provisioning algorithm for cloud computing environment through adjusting active servers in a data center according to demand fluctuation, variability in energy prices, and the cost of dynamic capacity reconfiguration. Abbasi et al., [12] propose a thermal-aware server provisioning that minimizes cooling and computational energy cost. This method does not consider environmental conditions. The method works well even in absence of the consideration as it is difficult to collect environmental weather data of a spatially distributed machines in a cloud. On the contrary, in a co-located cluster environment environmental conditions play a significant role in cooling energy minimization, which in turn substantially affect the total energy consumption.

There exist a few studies on energy minimization of clusters. For example, Pinheiro et al., [9] propose a cluster operational mode controller to save energy by dynamically turning cluster nodes on and off in a way that efficiently matches load demand. Sunuwar et al., [10] propose power management in heterogeneous MapReduce clusters through exploiting a trade-off between energy efficiency and throughput considering only computational power. Similar to this study, all the existing studies for cluster ignore cooling energy [13]. In order to overcome the limitation of existing studies we propose a new approach to minimize total energy consumption of cluster including cooling energy with a minimal overhead while considering environmental conditions.

III. Our Proposed Methodology

The total energy consumption of a MapReduce cluster comprises computational energy and cooling energy. Here, computational energy refers to the energy consumed by computing machines of a cluster. On the other hand, cooling energy is the energy consumed by the cooling machines of the cluster that are required to externally cool computing machines of that cluster to maintain the room temperature within a certain limit.

To do so, first, we formulate the total energy consumption (E) of a MapReduce cluster as follows:

$$E = E_{CPU} + E_{AC} = (P_{CPU} + P_{AC}) \times t \qquad (1)$$

Here, E_{CPU} and E_{AC} denote computational energy and cooling energy of the cluster respectively. Besides, P_{CPU} and P_{AC} denote the total power consumed by computing machines and cooling machines for completion of a job in time t. That means t is the required time to complete the job. We define it as response time in this paper. We propose to use different modules for estimating the components of our formulation. Next, we illustrate the different modules.

A. Modules in Our Proposed Framework

GMC framework consists of three modules namely Sensing module, Learning module, and Capacity Provisioning module. Fig. 1 delineates a block diagram of our proposed GMC framework depicting the modules and their inter-operability.

1) Sensing Module

Sensing module collects sensor data pertinent to room temperature, environment temperature, and power consumption of both computing and cooling machines. It also collects starting and ending time for a specific job. This information is then stored in the database and feed to the Learning module as training data.

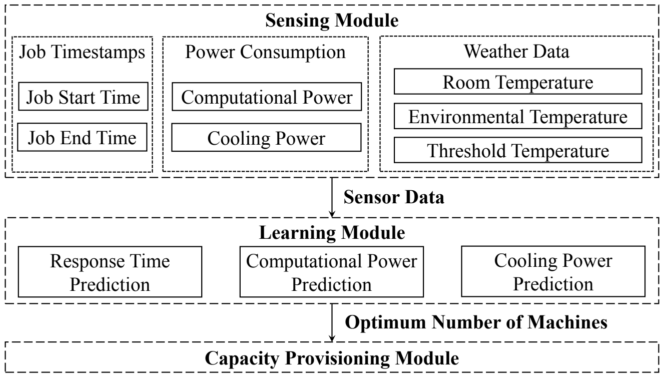


Fig. 1. Block diagram of operations in our proposed framework

2) Learning Module

Learning module utilizes the training data to learn for generating predicted total energy consumption for all possible numbers of computing machines in a cluster for an incoming job. Then the Learning module return the optimum number of machines for the next incoming job based on it's training. The best algorithm for out data set is decided by Auto-WEKA 2.0 [14]. Then that decided algorithm is implemented to our data set for learningusing Weka 3 [15].

3) Capacity Provisioning Module

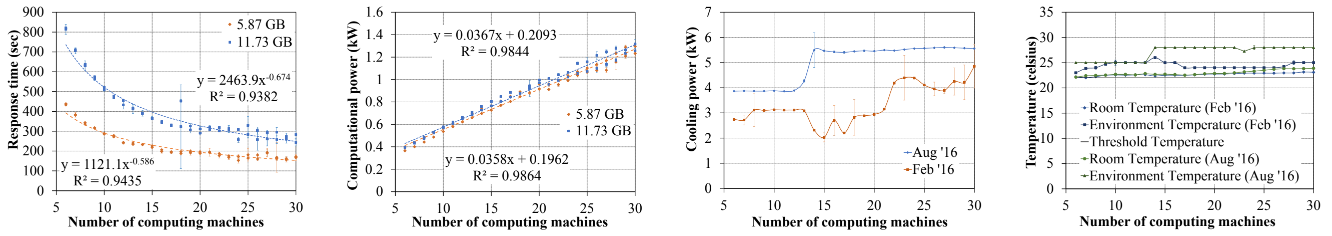
This module utilizes outputs generated from the Learning module to modify operational size of the cluster through activating the required number of computing machines. If the cluster is homogeneous, it randomly sends machines in sleep or wake state to activate the required number of machines. If the cluster is heterogeneous, we need another mechanism to select the best combination for the required number of machines. Here, we can use an evolutionary approach to select the best combination. Irrespective the nature of the cluster, the key operation performed by the modules is energy prediction done by the Learning module. We elaborate the process of energy prediction in our framework in the next subsection.

B. Energy Prediction through Different Machine Learning Techniques

In order to predict minimum total energy for an incoming job (shown in Equation 1), we need to predict both computational energy and cooling energy. To do so, we predict computational power, cooling power, and response time for a specific job. Next, we describe different mechanisms adopted for predicting each of them.

1) Prediction of Response Time

Response time depends on two factors namely size of the job and the number of operational computing machines. For a specific job size, an increment in the number of machines will increase throughput of the cluster. Therefore, response time will decrease with the increase in number of computing machines. Fig. 2a confirms the trend in response time, which we empirically find for different number of machines with two job sizes (we present experimental setup in section IV). Here, R^2 is the correlation coefficient, which represents how well the equation describes the data. We deploy 1-nearest neighbors (1-NN) [16] classifier to predict the response time for an incoming job.



(a) Trend in response time (b) Trend in CPU power (c) Trend in AC power (d) Trend in weather data

Fig. 2. Trends in response time, computational power (CPU power), and cooling power (AC power) along with weather data for different numbers of machines with two different job sizes in different seasons

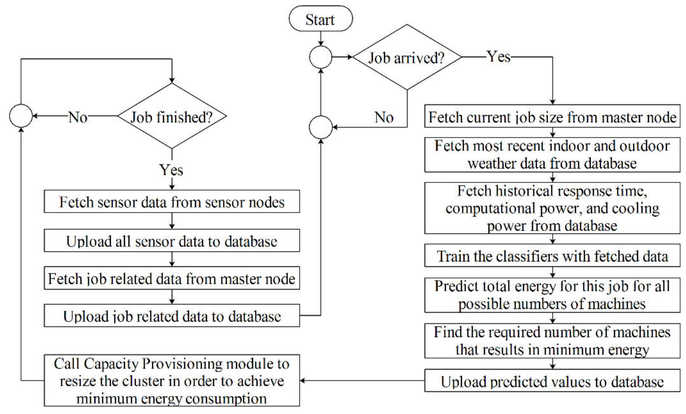

Start
Job arrived?
Yes
No
Fetch current job size from master node
Fetch most recent indoor and outdoor weather data from database
Fetch historical response time, computational power, and cooling power from database
Train the classifiers with fetched data
Predict total energy for this job for all possible numbers of machines
Find the required number of machines that results in minimum energy
Upload predicted values to database
Job finished?
No
Yes
Fetch sensor data from sensor nodes
Upload all sensor data to database
Fetch job related data from master node
Upload job related data to database
Call Capacity Provisioning module to resize the cluster in order to achieve minimum energy consumption

Fig. 3. Working process of our proposed GMC framework

2) *Prediction of Computational Power*

Computational power depends on the number of machines and job size. We also use CPU usage and memory usage for each cluster to predict the computational power. The power exhibits a linearly increasing trend for a homogeneous cluster. Moreover, it also increases for larger job sizes. Fig. 2b demonstrates the trend in computational power empirically found over different number of machines for two job sizes. We utilize Support Vector Machine (SVM) for regression [17] with Pearson VII function-based universal kernel [18] in our SVM classifier for predicting of computational power of a cluster exhibiting the trend.

3) *Prediction of Cooling Power*

Cooling power depends on the number of operational computing machines, job size, room temperature, environmental temperature, and expected temperature. The computing machines generates heats during their active period. Therefore, the number of active machines and the response time of each job has a significant impact on cooling energy consumption. Also the cooling work varies form one season to another and from day to night. Fig. 2c exhibits empirical trends of cooling power over the number of machines empirically found for a particular job size in two different seasons. Fig. 2d delineates the corresponding room temperatures, environment temperatures, and threshold temperatures for fall and spring seasons. We use a meta classifier called Additive Regression [19] having Random Forest [20] as the base classifier for predicting cooling energy exploiting the empirical data.

TABLE I
CONFIGURATION OF OUR HADOOP CLUSTER

Processor	Cores	RAM	# of machines
Intel Core 2 Duo E4600 @ 2.40GHz	2	1 GB	1
Intel Core 2 Duo E7300 @ 2.66GHz	2	2 GB	11
Intel Core 2 Duo E7400 @ 2.80GHz	2	2 GB	17
Intel Core i5-3470 @ 3.20GHz	4	4 GB	1

C. *Working Process*

Fig. 3 demonstrates overall working process for our proposed GMC framework. Here, upon arrival of a job at the master machine, the framework fetches required data for performing prediction. Then, it finds out the optimum number of machines and the job is then executed with that optimum number of machines. Next, the database is updated with current job data to make a better prediction for the next job.

IV. EXPERIMENTAL SETUP

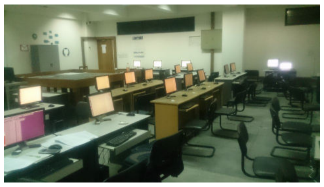
Our testbed deployment consists of a cluster of 30 computing machines and three cooling machines. We implement MapReduce [21] in our cluster using Apache Hadoop [22]. Our GMC framework works on top of Hadoop. We have three sensor nodes to collect data for our Learning module.

A. *Hadoop Cluster*

Our Hadoop cluster has 29 Intel Core 2 Duo slave nodes and one Intel Core i5 master node. Master node can also work as data node, hence, we have a total of 30 slave nodes. We use Ubuntu 14.04 LTS ($x86$) and Hadoop 1.0.3 in all the computing nodes. Table I exhibits detail configuration of our computing nodes. We use default HDFS block size as 64 MB and default replica factor 3 in our Hadoop configuration. Fig. 4a delineates the Hadoop cluster in our testbed.

B. *Sensor Nodes*

We use DHT22 Temperature and Humidity sensors [23] and Arduino Uno R3 for collecting room temperature. Besides, we collect environment temperature from web using Dark Sky API [24]. Fig. 4b shows a snapshot of our weather sensor node. Additionally, the power distribution unit of our cluster connects all computational nodes' CPU through a network of multi-plugs. We deploy an Aduino energy monitor near the supply line, which uses the Open Energy Monitor library [25]. Fig. 4c shows a snapshot of our CPU power sensor node. We have three split-type 2-ton air conditioners (ACs) in our testbed. We use another Ardunio energy monitor to collect power consumption of cooling machines. Fig. 4d shows a snapshot of our AC power sensor node.


(a) Our Hadoop cluster

(b) Weather sensor node

(c) CPU power sensor node

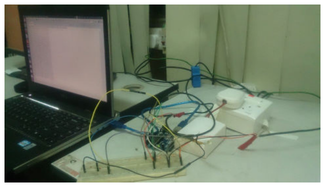

(d) AC power sensor node

Fig. 4. Snapshots of our experimental setup

C. Implementation of Our GMC Framework

We implement our GMC framework in the master node. We create our standard workload by following a similar approach mentioned in [10]. We use MapReduce wordcount application as our workload. We provide benchmark data set [26] as input file of the wordcount application. Our job size varies from 1-10 map tasks, where 64 MB data is equivalent to one map task. Since machines in our testbed deployment are old-aged and exhibit poor performance while executing a job, we face constraint pertinent to cluster capacity. Accordingly, we restrict ourselves to within 10 map tasks. We upload all the jobs in HDFS prior to running a job scheduler. We use total 68 jobs with different number of map tasks. We submit jobs in a FIFO scheduler having different lengths of inter-arrival time, which follows an exponential distribution with a mean of 6.3 second. The average job size in our job scheduler is 147.45 MB, therefore, our arrival rate is 23.4 MBps. Next, we present performance of our proposed GMC framework.

V. Performance Evaluation

We implement our GMC framework in our test-bed with static method and two state-of-the-art green methods [7], [10]. In all experiments, GMC outperforms all existing approaches.

A. Performance of Learning Module

We perform experiments to evaluate efficacy of our Learning module over the same series of jobs. We perform three iterations and take the average value over the iterations for each experiment. Before experimentation, we need to select the best combination of the hyper-parameters to fit the model perfectly. Therefore, we use Auto-WEKA to select the best fitted learning algorithm with its hyper-parameters. Here, we use Root Mean Squared Error as a metric to get the better model from Auto-WEKA and allow it to run around 15 minutes by using 10-fold cross-validation in each combination. We provide our historical data collected in spring and fall seasons as the training data set in this regard. We utilize the best fitted model with best settings of hyper-parameters suggested by Auto-WEKA and evaluate the performance.

Fig. 5a, 5b, and 5c illustrate the performance of response time, computational (CPU) power, and cooling (AC) power over the same series of jobs, respectively. Our evaluation reveals that all the predicted metrics follow the actual trend mostly remaining very close to the actual values irrespective of temperature variations as shown in Figure 5d. The overall prediction performance of our learning module is summarized in Table II. It is to be noted that our Learning module can predict the total energy consumption with 97.2% accuracy, although it exhibits some more deviations in predicting individual components of the total energy.

TABLE II
PERFORMANCE OF PREDICTION USING OUR LEARNING MODULE

Prediction Variable	Actual Mean	Predicted Mean	Accuracy (%)
Response Time (sec)	65 (± 7.6)	57 (± 2)	87.3
CPU Power (watt)	333 (± 77)	338 (± 73)	98.6
Cooling Power (watt)	1672 (± 40)	2211 (± 327)	67.8
CPU Energy (kJ)	26 (± 6.9)	21 (± 3.5)	83.1
AC Energy (kJ)	110 (± 14)	125 (± 18)	86.4
Total Energy (kJ)	143 (± 18)	147 (± 18)	97.2

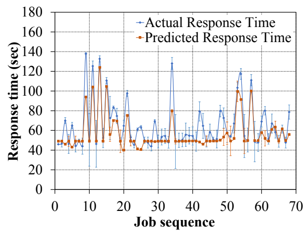
In this table we can see that, the accuracy of cooling power consumption is significantly low compared to others. Because the cooling power consumption depends on some additional factors e.g. humidity of the surroundings, cooling machine operations in nearby places of the test area etc. which we could not take account in this work.

B. Implementing Issues

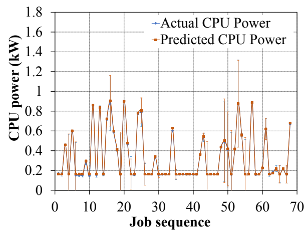
Popular state-of-the-art approaches attempt to achieve a trade-off between different performance metrics. For example, Zhang et al., [7] perform a trade-off between computational energy and waiting time. Besides, Sunuwar et al., [10] perform a trade-off between computational power and throughput. Similarly, in our GMC method, we attempt to achieve a trade-off between total energy and throughput. Here, throughput denotes the amount of data processed per second and waiting time represents the time a job need to wait in queue before starting its processing. Additionally, we consider a non-green method, i.e., static method [7] in our evaluation. The reason behind such a choice is that static method provides the maximum possible performance in terms of throughput enabling us to compare the throughput of GMC against the best-possible case. Furthermore, to resemble a real data center, we use a cluster of 28 slave nodes and one cooling machine to maintain a ratio of cooling power and computational power close to a real data center [27].

1) Static Method

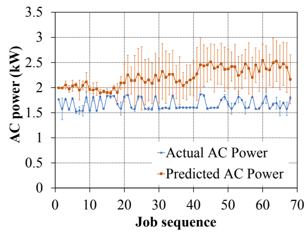
Static method provides the lowest response time, highest throughput, and lowest waiting time. One advantage of static method is that it does not need to run a data processing node in the master machine. Since it does not dynamically provision capacity, there is no chance to face data unavailability. On the contrary, our GMC method provisions capacity dynamically in a MapReduce environment, which exhibits a chance a great chance to face data unavailability [28]. Hence, if some machines having some required file blocks for a particular job get turned off, those blocks will be unavailable for that time. We address this problem by running a data processing node in our master machine. Since Hadoop's policy is to supply missing blocks from the master machine if it runs a data



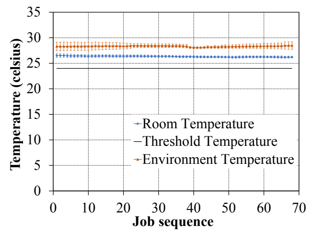
(a) Response time prediction



(b) CPU power prediction

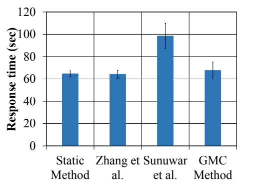


(c) Cooling power prediction

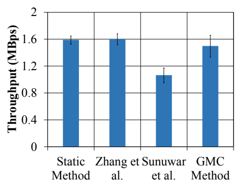


(d) Temperature condition

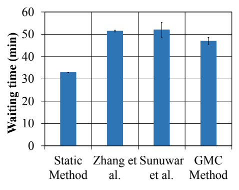
Fig. 5. Response time, computational power, and cooling power prediction using our learning module for a series of jobs



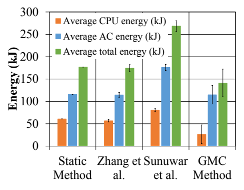
(a) Response time



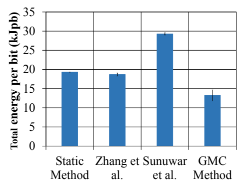
(b) Throughput



(c) Waiting time



(d) Energy components



(e) Total energy per bit

Fig. 6. Performance and energy consumption comparison among static method [7], Zhang's method [7], Sunuwar's method [10], and GMC method

processing node, Hadoop's master machine copies one set of blocks to its data node during data uploading. Therefore, we need to run a data processing node in our master machine. Thus, we use 28 slave nodes including the master machine for our GMC method and excluding the master machine.

2) *Dynamic Capacity Provisioning for Homogeneous Cloud*

Zhang et al., [7] propose dynamic energy-aware capacity provisioning for homogeneous cloud computing environments according to both demand and resource price. We predict its operational parameters namely cluster utilization, waiting jobs, and electricity cost using auto.arima function of R language [29]. Besides, we use Psensor [30] application installed in every machine to log real-time CPU usage data needed in the approach. Additionally, for electricity cost, we use electricity tariff 2015 [31] for Bangladesh, a developing country of South Asia. In addition, we set unit penalty cost as one. Further, we measure the slope of waiting time using the similar approach as mentioned in existing studies [32] and get 190.24. We also measure the idle power and get 33.97 watts. By using all these values, we get the optimum number of machines for the next job. Besides, we set $Q = 1$ and $R = 0.001$ as the hyper-parameters of the cost function.

3) *Static Resource Provisioning for Heterogeneous MapReduce Clusters*

Sunuwar et al., [10] propose a power management technique for heterogeneous MapReduce cluster. Since we have heterogeneity in our cluster, we implement this method in addition to our framework. They address the data unavailability problem of dynamic capacity provisioning by turning all machines on and provision CPU usages of the machines through imposing an upper bound of the CPU usage. Note that, we address this problem by running a data processing node in our master machine. Besides, Sunuwar et al., distribute workload over heterogeneous groups of servers in such a way that total power consumption gets minimized while keeping throughput within a certain level for a particular incoming workload arrival rate. To implement this, first, we develop individual power model for each heterogeneous group and assume that power equation for all nodes in a group remains identical. Besides, we divide our slave nodes into two groups based on processor frequency (Table I) namely, Dell group of 17 nodes and HP group of 11 nodes. We distribute workload in a particular group depending on a descending order of disk size, since larger disk size implies ability to contain higher volume of data enabling the machine work more. Additionally, we measure power consumption and log response time for different CPU usages to formulate the power models and throughput model of the two groups as follows: $P_{HP_i} = 15.11u_{HP_i} + 34.33$, $P_{Dell_i} = 18.32u_{Dell_i} + 33.32$, and $\tau = 18.4u_{HP} + 33.86u_{Dell}$. Here, we use cgroup [33] to limit the CPU usages.

C. *Performance Comparison*

We consider three metrics namely response time, throughput, and waiting time in our performance comparison. Fig. 6a, 6b, and 6c demonstrate the performance comparison in terms of response time, throughput, and waiting time. Since response time and throughput are correlated to each other, lower response time produces higher throughput for a particular job. Our evaluation reveals that static method outperforms other approaches in terms of the three performance metrics and the approach presented by Sunuwar et al., exhibits the worst performance. Our evaluation exhibit that our proposed GMC and Zhang's methods are very close to the static method in terms of response time and throughput. However, Fig. 6d confirms that our proposed GMC method outperforms all other methods in term of different energy consumption, i.e., computational, cooling, and total energy consumption.

Static method requires high computational energy to keep all machines operational for a job. Besides, Zhang's method requires high computational energy, since it powers on a large number of machines to reduce waiting time, which is caused due to sensor data collection overhead. Besides, Sunuwar's method requires higher energy, since it exhibits high response time caused due to CPU usage limitation. Additionally, cooling energy consumption of Zhang's method, static method, and

GMC are almost similar whereas Sunuwar's method consumes much higher cooling energy. Overall, GMC outperforms two state-of-the-art green methods through lowering total energy consumption by 19% and 47% over Zhang's method [7] and Sunuwar's method [10] respectively. Moreover, GMC maintains almost similar performance compared to the best performance provider, i.e., static method, in terms of throughput with 5.7% degradation. Nonetheless, our GMC method achieves the best trade-off between total energy consumption and throughput in terms of total energy consumption per bit (Fig. 6e) through achieving 32%, 29%, and 55% improvement over static, Zhang's, and Sunuwar's methods respectively.

VI. Conclusion

MapReduce clusters consume substantial amount of total energy covering both computation energy and cooling energy. However, little effort has been spent to minimize the total energy consumption considering both the energy components. Therefore, in this paper, we propose Green MapReduce Cluster (GMC) framework to determine the number of machines needed in a cluster to be activated for a job so that minimum energy consumption for the input data set and current weather condition gets guaranteed. For this purpose, we develop an experimental setup for investigating both computational and cooling energy consumption of a MapReduce cluster and collect real data by performing several series of experiments over a year in two different seasons. Subsequently, based on the collected data, we propose different machine learning based approaches to determine the number of machines. Our testbed implementation of GMC framework along with three state-of-the-art methods demonstrate an improvement of up to 47% reduction in total energy consumption, while experiencing marginal degradation in throughput in a few cases. In future, we plan to simulate our environment in large heterogeneous clusters and use context dependent classifiers in this case such as Kalman filter. Furthermore, we envision to explore many-objective optimization based approaches to minimize energy consumption in addition to optimize all other performance metrics in parallel.

Acknowledgment

This work was supported by the ICT Division, Government of the People's Republic of Bangladesh. The authors also would like to acknowledge the department of CSE, BUET for providing the necessary machines to make the parallel computing testbed of this work.

References

[1] B. Barney, "Introduction to parallel computing," https://computing.llnl.gov/tutorials/parallel_comp/, accessed: Oct. 28, 2017.

[2] Y. Wang, J. Li, and H. H. Wang, "Cluster and cloud computing framework for scientific metrology in flow control," *Cluster Computing*, Sep 2017.

[3] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: a survey," *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11–20, 2012.

[4] N. Sadashiv and S. M. D. Kumar, "Cluster, grid and cloud computing: A detailed comparison," in *2011 6th International Conference on Computer Science Education (ICCSE)*, Aug 2011, pp. 477–482.

[5] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services." *NSDI*, vol. 8, pp. 337–350, 2008.

[6] B. Guenter, N. Jain, and C. Williams, "Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning," *IEEE INFOCOM*, pp. 1332–1340, 2011.

[7] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein, "Dynamic energy-aware capacity provisioning for cloud computing environments," *ACM ICAC*, pp. 145–154, 2012.

[8] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[9] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Dynamic cluster reconfiguration for power and performance," *Compilers and operating systems for low power*, pp. 75–93, 2003.

[10] R. Sunuwar, "Power management in heterogeneous mapreduce cluster," Master's thesis, University of Nebraska-Lincoln, US, 2016.

[11] S. M. I. Omer, A. Mustafa, and F. Alghali, "Comparative study between cluster, grid, utility, cloud and autonomic computing," *IOSR Journal of Electrical and Electronics Engineering*, vol. 9, no. 6, pp. 61–67, 2014.

[12] Z. Abbasi, G. Varsamopoulos, and S. K. Gupta, "Thermal aware server provisioning and workload distribution for internet data centers," *ACM HPDC*, pp. 130–141, 2010.

[13] R. A. Shetu, T. Toha, M. M. R. Lunar, N. Nurain, and A. A. Al Islam, "Workload-based prediction of cpu temperature and usage for small-scale distributed systems," in *Computer Science and Network Technology (ICCSNT), 2015 4th International Conference on*, vol. 1. IEEE, 2015, pp. 1090–1093.

[14] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 826–830, Jan. 2017.

[15] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[16] D. Aha and D. Kibler, "Instance-based learning algorithms," *Machine Learning*, vol. 6, pp. 37–66, 1991.

[17] S. Shevade, S. Keerthi, C. Bhattacharyya, and K. Murthy, "Improvements to the smo algorithm for svm regression," in *IEEE Transactions on Neural Networks*, 1999.

[18] B. Üstün, W. J. Melssen, and L. M. Buydens, "Facilitating the application of support vector regression by using a universal pearson vii function based kernel," *Chemometrics and Intelligent Laboratory Systems*, vol. 81, no. 1, pp. 29–40, 2006.

[19] J. Friedman, "Stochastic gradient boosting," Stanford University, Tech. Rep., 1999.

[20] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[21] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[22] M. G. Noll, "Running hadoop on ubuntu linux (multi-node cluster)," https://goo.gl/yX6thp, accessed: Oct. 28, 2017.

[23] N. Honcho, "Dht22 temperature and humidity module," https://goo.gl/c1g8BR, accessed: Oct. 28, 2017.

[24] D. Ervideira, "Dark sky api," https://darksky.net/dev, accessed: Oct. 28, 2017.

[25] G. Hudson, "Openenergymonitor," https://learn.openenergymonitor.org/electricity-monitoring/ctac/how-to-build-an-arduino-energy-monitor, accessed: Oct. 28, 2017.

[26] "Data dumps," https://dumps.wikimedia.org/enwiki/, accessed: Oct. 28, 2017.

[27] A. Rizvi, T. R. Toha, M. M. R. Lunar, M. A. Adnan, and A. A. Al Islam, "Cooling energy integration in simgrid," *IEEE NSysS*, pp. 132–137, 2017.

[28] J. Leverich and C. Kozyrakis, "On the energy (in) efficiency of hadoop clusters," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 61–65, 2010.

[29] R. Ihaka and R. Gentleman, "R: a language for data analysis and graphics," *Journal of computational and graphical statistics*, vol. 5, no. 3, pp. 299–314, 1996.

[30] J. Orsini, "Psensor," https://wpitchoune.net/psensor/, accessed: Oct. 28, 2017.

[31] B. P. D. Board, "Electricity tariff 2015," https://goo.gl/tYpKgG, accessed: Oct. 28, 2017.

[32] Q. Zhang, J. L. Hellerstein, and R. Boutaba, "Characterizing task usage shapes in googles compute clusters," *LADIS*, pp. 1–6, 2011.

[33] D. Howard, "Implementing cgroups on ubuntu or debian," https://goo.gl/sjfk8E, accessed: Oct. 28, 2017.