MapReduce Service Provisioning for Frequent Big Data Jobs on Clouds Considering Data Transfers

Seyed Morteza Nabavinejad, Maziar Goudarzi, and Saeed Abedi

Abstract

Many companies regularly run Big Data analysis, and need to optimize their resource usage considering cost, deadline, and environmental impact simultaneously. The cloud allows choosing from various virtual machines (VM) where the number and type of VMs affect the outcome such as the time for data placement and data shuffle phases, a task's energy consumption and execution time, and the makespan of jobs. We provide provisioning and scheduling algorithms to minimize environmental impact, considering the above factors, for frequently executed MapReduce jobs. To mathematically model the problem and obtain the optimal solution, we present an Integer Linear Programming (ILP) model and then continue with two heuristic algorithms. We compare proposed algorithms against a number of rivals using extensive simulations based on publicly available real-world data. The results demonstrate that our algorithms can achieve near-optimal solutions, e.g., sometime even within 0.39% of the optimal solution obtained by ILP regarding energy consumption.

Keywords

Big Data, MapReduce, Cloud Computing, Hadoop, Energy Efficiency

1 Introduction

The volume of data is steadily increasing. Forecasts such as [1] predict that the volume of digital data in 2020 will be 300 times larger than 2005, furthering the significance of Big Data as well as Big Data Analytics which are already important needs. MapReduce [2], and its open source implementation Hadoop [3], are among the prevalent frameworks for implementing Big Data Analytics and applications. With the rapid growth of popularity in big data analytics, cloud providers have even launched dedicated services for MapReduce applications such as Amazon Elastic MapReduce (EMR) service [4]. Private and hybrid clouds as well as MapReduce services have also gained traction due to the advantages the cloud paradigm provides. Big Data jobs of companies usually involve financially and/or strategically invaluable data of the company, and hence, transferring this data to public clouds for processing is a constant source of security concerns for them;

- *The authors were with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, when this work was done. Maziar Goudarzi is the corresponding author. Seyed Morteza Nabavinejad is currently a Postdoc at School of Computer Science, Institute for Research in Fundamental Sciences (IPM). Saeed Abedi is currently a Ph.D. student at the Department of Computer and Information Science, University of Pennsylvania.*
  E-mail: nabavinejad@ipm.ir, goudarzi@sharif.edu, abedi@seas.upenn.edu

consequently, in-house Big Data processing over private clouds is becoming more widespread and has gained even more significance and attractiveness. Our work deals with such cases of in-house Big Data processing on a private cloud where the company's social responsibilities and obligations require it to reduce its environmental impact in addition to meeting more traditional requirements on cost and deadline.

Using a hybrid/private cloud introduces a number of concerns that arise from the on-demand provisioning nature as well as other features of the cloud: in addition to considering pay-per-use cost structure of the cloud, the amount of resources in the cloud is often limited as well as shared among other tasks of the company; consequently, the number and types of available VMs dynamically changes and depends on the cloud utilization at the time of Big Data job submission. In addition to above cloud-induced concerns, specific needs of the MapReduce paradigm introduces two further data-related concerns to address: a typical MapReduce job starts by *distributing the data* over employed processing nodes, then the *Map phase* starts which produces *intermediate (key, value)* data, afterwards a *shuffle phase* redistributes these data among processing nodes, and finally the Reduce phase is done which produces final results; prior art have focused mainly on the *Map and Reduce phases* for optimization, but the initial data distribution as well as intermediate data shuffle phases are also important especially in the case of a cloud implementation where the number and place of employed VMs are determined at run-time depending on the status of the cloud at the time of the Big Data job submission. We particularly consider these concerns as part of our major contributions in this paper.

Prior research addresses one or the other concerns of energy, cost, and deadline; while meeting the deadline of jobs is the sole concern of [5], there are other works [6], [7] that allocate resources and schedule the jobs considering both monetary cost and deadline, whereas considering deadline along with energy consumption is the theme of [8], [9]. However, if the above-mentioned issues imposed by cloud implementation of MapReduce are not taken into account, the obtained results would be unusable in this context: solely scheduling Map or Reduce tasks to available machines, without considering prior placement of the data to work on, could result in extra waiting times for data to be transferred from other machines, and consequently, not only the task execution time as well as the makespan are lengthened, but also this could result in speculative execution of extra Map or Reduce tasks to compensate for the slow task, worsening the situation even further. Data locality importance is a well known topic [5], [10]. However, as far as we know the state of the art task assignment and scheduling approaches ,that take into account both cloud provider and user concerns (e.g., cost, deadline, and energy), do not consider it. But, we consider data locality in our approach.

We first provide a mathematical model of the above provisioning and scheduling problem. Then we develop an Integer Linear Program (ILP) to solve the problem in small-scale cases and obtain the optimal solution to evaluate quality of results of our two heuristics, namely Double Stage Provisioning and Scheduling (DSPS) and Machine Valuation Provisioning and Scheduling (MVPS) which solve the problem with polynomial time complexity. For larger-scale cases where the ILP execution time renders it unusable, we relax it to a Linear Program (LP) to obtain a lower-bound for the optimal solution so as to obtain a sense of how closely our heuristics can approach the optimal solution.

For evaluating the proposed algorithms, we have considered eight MapReduce applications from PUMA benchmark suit [11] as Big Data jobs and nine VM configurations from Google Cloud [12] as the type of VMs

available in the cloud.We formulate the execution time of each MapReduce job to evaluate the performance of proposed algorithms and compare their solutions with the optimal or the lower bound ones depending on the problem size. We also have compared our algorithms with several rivals. The results illustrate that our proposed algorithms can find near optimal solutions; our DSPS and MVPS algorithms can find solutions as close as 0.39% to the ILP solution on account of energy consumption.

Our major contributions in this work are as follows:

- We take into account the initial data distribution phase as well as the intermediate data shuffle phase which are particularly relevant when the MapReduce job is running in cloud. Not considering these items in prior works renders them unusable for such implementations.
- We simultaneously consider energy, cost, and deadline for environment-aware provisioning and scheduling of frequently executed MapReduce jobs on the cloud.
- We quantify available saving opportunities for improving the energy efficiency of provisioning and scheduling techniques.
- We mathematically model the optimization problem for clear, concise, and unambiguous representation, and develop an ILP model to obtain the optimal solution in smaller problems, and a relaxed LP version of the same model to obtain a lower bound of the optimal solution in larger scales. These serve as the golden models to evaluate the quality of the heuristics we provide.
- We contribute two heuristic algorithms, DSPS and MVPS, to efficiently solve the problem in polynomial time and find near optimal solutions for online usage. Then, we analyze the sensitivity of our approach to cost limit, deadline, and network speed.

The rest of the paper is organized as follow. In section 2, related works are discussed. System model we considered in our work is presented in section 3. We also have motivated our work in this section. In section 4, we discuss the problem formulation of resource allocation and task scheduling of MapReduce applications in the cloud. Section 5 is dedicated to the heuristic algorithms. In section 6, we present the experimental results and evaluate the algorithms. Finally, section 7 is dedicated to discussion about results and avenues for future research.

2 Related Work

We divide the researches on MapReduce task assignment and scheduling into three categories: Deadline aware approaches, Energy and Deadline aware approaches, and Cost and Deadline Aware approaches.

Deadline aware MapReduce task assignment and scheduling. The primitive proposed algorithms only consider deadline or response time of jobs when scheduling them [5], [13]. Considering homogeneous environments, [13] proposes scheduling approach that regards Map and Reduce phases of MapReduce applications as different stages and schedules them separately. The reason behind this separation is different code of each phase. The average time of tasks is estimated through a one-to-one sampling method. It also considers the jobs priority in scheduling. Unlike [13] which considers the homogeneous environments, [5] pays attention to the heterogeneity of resources in cloud. This work mentions four concerns that were not considered in cloud-based deadline aware

scheduling algorithms: a) slot performance heterogeneity that originates from this fact that there are different types of machines with various hardware resources in cloud, b) adaptive task deadline setting for Map and Reduce phases, c) impact of data locality on deadline and d) reducing the number of jobs that have not met their deadline. According to these points, this work presents BGMRS scheduling algorithm based on bipartite graph modeling.

Cost and Deadline aware MapReduce task assignment and scheduling. Another category of related works consider the cost of resources in addition to deadline of jobs when assigning tasks. For example, same as [5] in previous category, [7] includes the hardware heterogeneity in decision making for task assignment. It mentions that in two situations the task placement must be done: when some jobs are finishing or when some new ones arrive, and the proposed method considers the latter. Several workers for each job are assumed and each worker must gain sufficient resources so the desired job QoS can be secured. Simultaneously, the cost of VMs executing each jobs workers must be as low as possible. For solving the problem, it first produces a set of possible placement of workers on VMs (Placement Patterns Generation). Then, the MapReduce Placement Problem (MRPP) Solution Building uses the sets generated in the first step to assign the workers to VMs.

Three operational model for cloud are introduced in [6]: Immediate execution, Delayed start, and Cloud managed where the provider is completely responsible for everything such as resource allocation and job configuration. The customer only submits the job with its desired deadline. The profit of cloud provider can be maximized using this model. Cura, introduced in [6], aims to increase the profit of cloud provider using the third model, i.e., Cloud managed. Cura is optimized for short workloads but long ones also can benefit from it.

Energy and Deadline aware MapReduce task assignment and scheduling. A subset of researches is conducted in this category such as [8], [9]. MapReduce production jobs that process the Big Bata periodically in clusters are the target of [9]. Because of their periodical nature, it is possible to profile them and find the energy consumption as well as execution time of their tasks for every machine slot. Using aforementioned information, this work tries to schedule the Map and Reduce tasks on the machine slots in order to reduce the energy consumption of cluster while the job meets its deadline. First, the problem is formulated as an integer linear program and optimal solution for small scale jobs with a few tasks is obtained. Since it is very time consuming to find the optimal solution for real world large scale jobs, two heuristic algorithms called Energy-aware MapReduce Scheduling Algorithms (EMRSA-I and EMRSA-II) are proposed. These algorithms try to assign the tasks to the most energy efficient slots while taking the deadline into account. Ref [8] also proposes an energy-aware task assignment approach for Hadoop jobs in heterogeneous clusters. The goal is minimizing energy consumption without performance degradation. Unlike [9] that supposes there is prior knowledge about energy consumption and execution time of tasks, it assumes that there is no such information. Employing the ant colony optimization, the proposed E-ant algorithm tries to schedule the tasks on the machines adaptively. The impact of jobs on each other is also considered in E-ant. The pivotal difference between these two works and ours is that they do not consider resource allocation and only focus on task assignment because they assume that all the resources in a cluster (machines and their specifications) are given. Furthermore, they do not take into account the cost limit set by the customer; deadline is the only constraint specified by costumer in these works. Not considering the data placement and data shuffle stages is another difference between these two and our approach.

3 System Model

In this section, we introduce MapReduce execution model we have considered in this work. An overview of model is depicted in Fig. 1. The model has four stages: *Creating HDFS and Copying Data to Allocated Nodes (Data Placement)*, *Executing Map Tasks*, *Shuffling Intermediate Data (Data Shuffle)*, and *Executing Reduce Tasks and Generating Final Output*. In the following we explain each stage in more details.

Creating HDFS and Copying Data to Allocated Nodes (Data Placement). There are two approaches regarding data placement. The first one supposes that data is already copied into HDFS. This approach is used for most on-premises MapReduce clusters, but is not a general rule. The other approach, however, considers data placement as a part of processing flow of MapReduce jobs and leverages it to improve the objective function. A large body of research such as [14], [15] has focused on the data placement phase of MapReduce jobs, both in physical clusters and cloud, which shows the importance and significance of the second approach. We also consider this approach in our work.

In our work, we assume that Hadoop creates HDFS as the file system on the allocated nodes and then copies the input data in the form of data blocks to it for processing. Same as several previous works [16], we assume that data is stored in local storage system or cloud storage such as Amazon S3 in first place, which have negligible processing power and are only used for storing data. Then, data is transferred from the storage system to HDFS created on computing nodes (i.e., VMs in our work) to be processed.

Two scenarios for copying data to nodes exist: Distributing the data blocks uniformly between the nodes or considering the computing capacity of each node and placing data blocks on it based on this capacity. In this work we use the second scenario. Our heuristic algorithms first assign tasks to each machine while considering *data placement* and *data shuffle* overheads, and then we arrange to copy the required data blocks of assigned tasks to that machine. An API introduced for specifying data blocks placement policy in HDFS [17], which is used in previous works such as [14], can realize the process of copying data blocks to specific machine we determine in our approach. Regardless of selected scenario, the process of copying the data into HDFS is time consuming and each node consumes energy during this process. Furthermore, the number of nodes allocated for processing the job determines the copy time and consequently energy consumption. So, this stage must be considered in resource provisioning and scheduling optimizations.

Executing Map Tasks. In the MapReduce programming paradigm, every job has two phases called Map and Reduce which consist of a number of tasks. Each Map task processes a data block and produces some intermediate data in the form of *(key, value)* pairs. The pairs with the same key are later combined by the so-called Reduce tasks in the Reduce phase. The operations to be performed by each Map and Reduce task are provided by the programmer. The intermediate data is initially stored on the node that has generated it. Each Map task is executed by a Map slot and each VM has several Map slots. The number of Map slots can be configured offline and remains fixed until the next offline change. It is commonplace to set the number of Map and Reduce slots same as the number of VMs cores [9]. If the number of available slots is equal to the number of tasks, then all the tasks can be executed in parallel. However, in most cases the number of slots is not sufficient for parallel processing of all tasks. So, the tasks need to be scheduled on the slots. Since we have considered in our work

that we first schedule Map tasks and then copy the data into nodes based on the arranged schedule, each node has the data blocks of its assigned Map tasks and there is no need for data transfer between nodes during Map phase.

Shuffling Intermediate Data (Data Shuffle). As briefly mentioned above, (key, value) pairs sharing the same key are combined in the Reduce phase. Consequently, unlike Map phase, where each task processes a specific block on the node and does not need data from other nodes, the tasks in Reduce phase may need intermediate data from other nodes in order to complete their processing. Hence, the intermediate data needs to be transferred among the nodes; this stage is called shuffle. Shuffling intermediate data imposes time overhead so we must consider it similar to HDFS creating and copying. Since we do not have control over data that each Reduce task must process, we consider the worst case scenario to make sure that deadline of job is met. The worst case scenario happens when none of the nodes has the intermediate data needed for the Reduce tasks that are assigned to it. Consequently, the data has to be transferred from other nodes, which leads to communication overhead and extra energy consumption.

Executing Reduce Tasks and Generating Final Output. Execution of Reduce tasks is similar to Map tasks. Similar to Map phase, Reduce phase also consists of several Reduce tasks which are executed by Reduce slots of VMs. Each Reduce task processes a portion of intermediate data to generate final output and stores it in HDFS. In this work, we assume that Reduce phase starts after both the Map phase and data shuffle phases are finished. While it is a common assumption [9], other approaches [18] exist that try to reduce execution time by overlapping Map phase and Reduce phase. It is an interesting direction for further investigations but is out of scope of this paper. The main challenge to cover in that case is the modeling of execution time and energy consumption of Map and Reduce tasks as well as the shuffle operation when overlapped; simple assumptions, such as their independence, could have been used here but further and deep investigation is required to validate that assumption or find out a proper model. Thereafter, our mathematical model in next section can be easily extended to cover the case of overlaps as well.

Motivation. In the following, we demonstrate the need, and evaluate the opportunity, for a provisioning and scheduling scheme that can minimize energy consumption considering cost limit, deadline and resource availability with respect to system model introduced above. We demonstrate the wide gap regarding energy consumption among the design points in the design space. We implemented a simple algorithm that repeatedly generates random resource provisioning and scheduling solutions and reports the energy consumption of the ones that meet the constraints (e.g., cost limit, deadline and VM availability). We ran this algorithm for all medium-scale workloads described in section 6 for 200 times. The results for energy consumption are presented in Fig. 2. Details of experiments setup for these results such as workloads, input data, VM specifications, power measurement method, etc., are presented in section 6.1.

The big difference in energy consumption among obtained acceptable solutions clearly states that there is great opportunity for energy saving within the valid solutions that can meet the constraints. The average difference between maximum and minimum energy consumption over all applications is around 22% and application InvertedIndex has the widest gap by 39.82%.

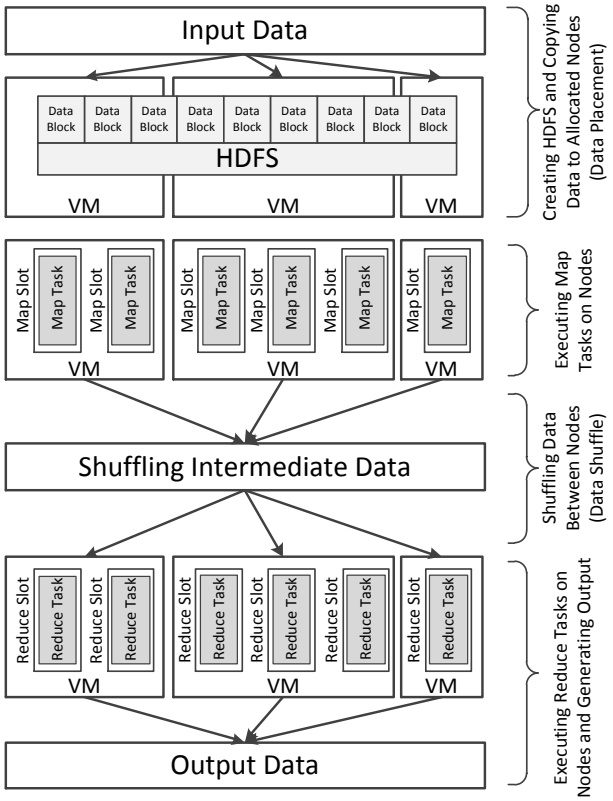


Fig. 1. Overview of system model.

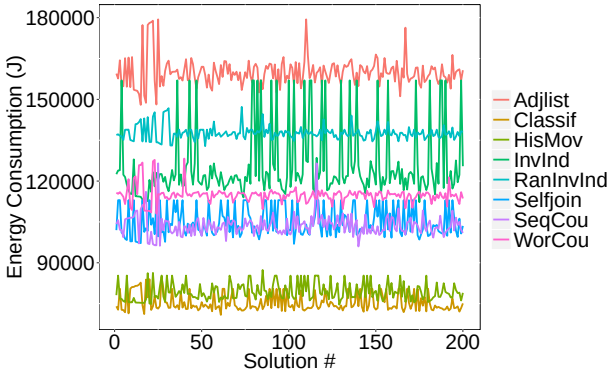


Fig. 2. Difference in energy consumption among various provisioning and scheduling solutions.

4 Problem Formulation

Considering the aforementioned system model in section 3, in this section we first present the notation used in problem formulation. After that, we clarify the problem statement. Finally, the problem formulation is represented.

TABLE 1
Notation

	Symbol	Definition
Given Parameters	SM	Set of map slots
	SR	Set of reduce slots
	M	Set of map tasks
	R	Set of reduce tasks
	SVM	Set of available VMs
	T_{ij}	The execution time of each task on each slot ($i \in M \cup R, j \in SM \cup SR$)
	E_{ij}	The difference between energy consumption of slot j when executing task i and its idle energy consumption ($i \in M \cup R, j \in SM \cup SR$)
	VMC_k	The cost of each VM ($k \in SVM$)
	D	Deadline of the job
	MC	Monetary cost limit of job
	S_{jk}	Whether slot j belongs to VM k,or not ($j \in SM \cup SR, k \in SVM$)
	IBS	Size of Input Blocks
	MtoR	Amount of intermediate data produced per unit input data
	RDS	Amount of data that each Reduce task must Process : $RDS = (\|M\| \times IBS \times MtoR)/\|R\|$
	IOP	Difference between power consumption of machine in data shuflle and data placement stages and its idle power consumption
	NS	Speed of network
Decision Variables	X_{ij}	Whether Map task i is assigned to slot j,or not,($i \in M, j \in, SM$)
	Y_{ij}	Whether Reduce task i is assigned to slot j,or not,($i \in R, j \in, SR$)
Determined During Optimization	DPE	Energy consumption of data placement stage
	DSHE	Energy consumption of data shuffle stage
	DPT_k	Data placement time of VM *k*
	$DSHT_k$	Data shuffle time of VM *k*
	UT	Up time of VMs cluste

Notation. The parameters we have used in our problem definition are presented in Table 1:

Problem Statement. We choose the type and the number of VMs of each type from among total VMs that the cloud can provide, and also select the proper Map and Reduce slots from available sets (SM and SR) on available VMs (SVM) for executing Map and Reduce tasks (M and R) and also schedule the tasks on selected slots in order to minimize total energy consumption for the given MapReduce job. The allocation and scheduling must be done considering job deadline (D) as well as the cost limit (MC) determined by the customer.

Similar to [9], we assume that energy consumption (E_{ij}) and execution time (T_{ij}) of each task on each slot is profiled prior to allocation and scheduling since the jobs are executed periodically. It is common practice to execute batch jobs periodically in order to process Big Data. For example, Google periodically creates the inverted index of web pages using MapReduce. This kind of batch jobs makes it possible to use job profiling approaches in order to extract the information about each Map and Reduce task of every individual job [9]. The execution time of a task on different types of VMs can be obtained via reports produced by Hadoop for every completed job.

While it is debatable how to profile the power consumption of VMs, several approaches are already proposed for it such as [19]. In our experiment setup, the power consumption is measurable through power meter tool embedded in servers that host the VMs. Finally, by having the execution time and power consumption, we can

calculate the energy consumption of each task on each type of VM. This information can be gathered because in the private or hybrid cloud, the cloud user is also (partially, in the hybrid case) the cloud provider and hence has access to physical servers and Hadoop logs and can measure the power consumption of tasks as well as their execution time. Note that a similar case occurs in a public cloud where the cloud provider provides Big Data processing service; the cost limit (actually the agreed total service fee for the given amount of Big Data) and the deadline are given by the service user, while the energy consumption and execution times of tasks are measurable by the provider, and the former needs to be minimized for the same reason as in private or hybrid clouds.

The cost of each virtual machine (VMC_k) and its slots information (S_{jk}) are the other given information. By using the size of input blocks (IBS) and number of Map tasks that each machine should execute, we calculate the amount of data that must be placed on each machine during data placement stage. MtoR estimates the volume of generated intermediate data in Map phase based on the size of input data. Amount of data that each Reduce task should process (RDS), is calculated using size of input data, MtoR and number of Reduce tasks. Obviously, speed of network links (NS) is also given. The last given parameter (IOP) is the power consumption of machines when they are in data placement or data shuffle stages since in these stages they only transmit and copy data.

The decision variables in the problem are X_{ij} and Y_{ij} which determine the proper slot for each task, and consequently the type and the number of corresponding VMs, in order to achieve the desired goal.

Problem Formulation.The goal is to minimize the energy consumption of the four stages of system model i.e., energy consumption of tasks execution plus energy consumption of data placement and data shuffle:

$$Minimize \sum_{i \in M} \sum_{j \in SM} E_{ij} X_{ij} + \sum_{i \in R} \sum_{j \in SR} E_{ij} Y_{ij} + DPE + DSHE \tag{1}$$

Subject to below constraints:

First, we must make sure that every task is assigned to one and only one slot: l

$$\sum_{j \in SM} X_{ij} = 1, \forall i \in M \tag{2}$$

$$\sum_{j \in SR} Y_{ij} = 1, \forall i \in R \tag{3}$$

After that, we should assure that execution time of tasks on the slots plus the time consumed for data placement and data shuffle is less than the deadline:

$$UT \leq D, \tag{4}$$

$$UT = Max(DPT_k) + Max(\sum_{i \in M} T_{ij} X_{ij}) + Max(DSHT_k) + Max(\sum_{i \in R} T_{ij'} Y_{ij'}), \forall j \in SM, \forall j' \in SR, \forall k \in SVM$$

According to system model introduced in section 3, all the stages should execute sequentially. First the data placement stage should finish and it finishes when the data placement of the last machine is done. After that the Map tasks should execute on the assigned slots. Then the intermediate data should be shuffled between all the machines. Finally, the Reduce tasks should process the intermediate data and generate the final output.

As we mentioned earlier in section 3, instead of the conventional data distribution that HDFS tools usually

apply when creating an HDFS file system, our optimization approach takes control of the data placement; we first assign the tasks to machines and then arrange to copy the data to machines based on the assignment so that each Map task has local access to the data it is to process. This is required for proper execution of Map tasks and avoids creating slow tasks that impose extra and speculative task executions; lack of paying attention to this important point is a major concern on validity of previously proposed techniques in this area. So we can calculate the data placement time for each machine by dividing the volume of data to speed of network as illustrated in (5):

$$DPT_{\mathrm{k}} = \frac{\sum_{i \in M} \sum_{j \in SM} X_{\mathrm{ij}} S_{\mathrm{jk}} \times IBS}{NS}, \forall k \in SVM \tag{5}$$

In the case that data is already copied in HDFS such as the first approach described in data placement stage of system model, we can simply assign zero to *IBS* parameter (Size of input block, see Table 1) which leads to zero data placement time for VMs ($DPT_{\mathrm{k}} = 0$)

In data shuffle stage of system model, we consider the worst case scenario as explained in previous section. So each machine should first send its generated intermediate data to other ones (first part of (6)) and then receive its required intermediate data from them (second part of (6)).

$$DSHT_{\mathrm{k}} = \frac{\sum_{i \in M} \sum_{j \in SM} X_{\mathrm{ij}} S_{\mathrm{jk}} \times IBS \times MtoR}{NS} + \frac{\sum_{i \in R} \sum_{j \in SR} Y_{\mathrm{ij}} S_{\mathrm{jk}} \times RDS}{NS}, \forall k \in SVM \tag{6}$$

After the data placement time and data shuffle time of each machine is obtained, we calculate the total energy consumption of data placement and data shuffle stages as in (7) and (8):

$$DPE = \sum_{k \in SVM} DPT_{\mathrm{k}} \times IOP \tag{7}$$

$$DSHE = \sum_{k \in SVM} DSHT_{\mathrm{k}} \times IOP \tag{8}$$

The next step is to calculate total cost of VMs and constrain it by the specified monetary cost. Since we should pay for all the VMs from the beginning of execution time until the end, we multiply the uptime of whole cluster by the monetary cost of each VM as in (9). Note that we should only pay for VMs that we have selected from the set of available VMs, so we only consider the VMs that have executed at least one task (the MIN function in (9)).

$$\sum_{k \in SVM} UT \times VMC_{\mathrm{k}} \times Min(1, \sum_{i \in M} \sum_{j \in SM} X_{\mathrm{ij}} S_{\mathrm{jk}} + \sum_{i \in R} \sum_{j \in SR} Y_{\mathrm{ij}} S_{\mathrm{jk}}) \leq MC \tag{9}$$

Finally, we indicate the allowed values that the variables can get. These expressions imply that each task can be assigned to only one slot and it is not possible to distribute a task among several slots. In the relaxed version of our Integer Linear Program (ILP) formulation, we remove these two constraints so that such distribution is allowed and the problem is converted to a Linear Programming (LP) problem to more rapidly solve and get a lower bound of objective function. This relaxation converts the original optimization problem, which is NP-hard, to a problem that can be solved in a polynomial time.

$$X_{\text{ij}} = \{0,1\}, \forall i \in M, \forall j \in SM \tag{10}$$

$$Y_{\text{ij}} = \{0,1\}, \forall i \in R, \forall j \in SR \tag{11}$$

5 Proposed Heuristic Algorithms

We designed two heuristic algorithms for resource allocation and task assignment of MapReduce applications on cloud. The inputs of algorithms are the number of Map and Reduce tasks of the job, execution time and energy consumption of each task on every type of VM, cost limit, deadline, and available resources (VM types and number) in cloud. As the output, the algorithms choose VMs and assign tasks to slots of VMs in order to reduce the energy consumption while satisfying the cost limit and the deadline set by the customers. Following subsections describe the algorithms in more details.

5.1 Double Stage Provisioning and Scheduling

Double Stage Provisioning and Scheduling (DSPS) is our first proposed algorithm which tries to reduce the energy consumption for executing the job while satisfying the desired constraints i.e., cost limit and deadline. Since satisfying both constraints simultaneously increases the complexity of placement algorithm, int the first step, we devised DSPS to address each constraint in one of its stages. It first tries to reduce the energy as much as possible, while only satisfying deadline and ignoring cost limit. In the next stage, it eventually satisfies the cost limit by modifying task placement obtained by previous stage. .Before describing the functionality of algorithm, some parameters are introduced. Parameter em_{ik} indicates the energy consumption of Map task *i* on slots of VM k and er_{ik} does the same for Reduce task *i*. Execution time of tasks on slots of different machines is shown by tm_{ik} and tr_{ik} for Map and Reduce tasks respectively. There are three deadlines in this algorithm. The first one, $D_{P\&S}$, is for data placement and data shuffle which are the first and third stages in Figure 1. Deadline of Map and Reduce phases (second and forth stages of system model) is indicated by D_{exe}. Finally, we have total deadline which is submitted along with job and the sum of two deadlines must be equal to it: $D_{total} = D_{P\&S} + D_{exe}$. With total deadline being constant, we assign various values to $D_{P\&S}$ and schedule the tasks. Finally we choose the schedule that yields minimum energy consumption. In the following, flow of algorithm is presented and its Pseudo code is demonstrated in Algorithm 1.

The first line of algorithm gives different values to $D_{P\&S}$ and after that, the first stage of algorithm starts. The first stage of algorithm assigns the tasks to machines in order to meet the deadline while minimizing the energy consumption. Lines 2 to 14 correspond to this stage.

Two priority queues, Q_m and Q_r, keep the order of VMs for Map tasks and Reduce tasks respectively. Max_m stores the maximum time that a Map slot is busy executing Map tasks, among all the available Map slots. Max_r does the same for Reduce slots (lines 2 to 4). First step in this algorithm is to sort the VMs in Q_m and Q_r in ascending order in terms of energy consumption (line 5). The energy of each machine is calculated using (12). After this step, the first VM in queue Q_m uses the least amount of energy to execute all Map tasks. The same is true for Q_r.

$$Q_{\mathrm{m}}[k].energy() = \sum_{i \in 1}^{M} em_{\mathrm{ik}}, Q_{\mathrm{r}}[k].energy() = \sum_{i \in 1}^{R} er_{\mathrm{ik}}, \forall k \in SVM \quad (12)$$

In each iteration of lines 6 to 14 a Map or Reduce task is assigned to a machine. This process continues until all the tasks are assigned or there is no machine available for a task. After identifying the type of task in line 7 (in the rest of this subsection, we assume that the task is Map), lines 8 to 11 try to find an appropriate slot in one of the machines for the task. Appropriate slot is one that if it processes the task, the deadline will not be missed. Furthermore, the slot must consume the least possible energy for executing the task.

In the interest of not missing the deadline, the algorithm tries to assign the task to the slot that is the least occupied (Q_m[i].MinTime() indicates the current minimum execution time of all the machine slots - see Fig.3). Note that we assume the execution time and energy consumption of a task on any slot of a machine is the same. Line 8 checks if the sum of the current execution time of that slot, the execution time of the new task and maximum execution time of Reduce slots (if the task is a Map, otherwise, the Map slots) is less than D_{exe}. The reason for considering the maximum execution time of Reduce slots is that the Reduce phase starts after Map phase and their execution time directly affects the total execution time. The process is depicted in Fig. 3. If the condition is true, then it assigns the task to the specific slot of machine and updates the Max_m if necessary (line 9). Otherwise, if the assignment of task to any slot of the current machine cannot meet the deadline, then it tries another machine. If there is no machine that assignment of task to it can meet the deadline, then the scheduling is impossible and algorithm terminates.

On account of energy consumption, the algorithm tries to select the machines from the head of queue Q_m which is sorted in ascending order of energy. As it is stated in line 14, the same procedure is followed if the task is a Reduce. This process is repeated until all the tasks are assigned. Then first stage is finished and algorithm proceeds to second stage.

Second stage of algorithm starts from line 15 to line 22. This stage is put into action if the cost limit is not met. The first step is to change the order of machines in Q_m and Q_r. This time, we sort the queues in descending order regarding per-slot cost of machines (line 15). Per-slot cost of machines (for Map tasks) is obtained by (13):

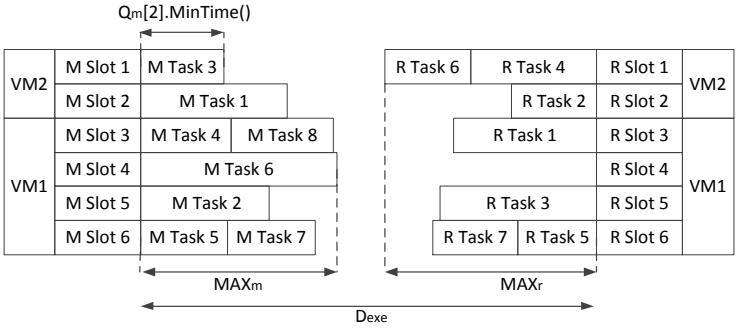


Fig. 3. First stage of DSPS algorithm.

Algorithm 1 *Double Stage Provisioning and Scheduling*

1: **for** $D_{P\&S} = D_{total}/10; D_{P\&S} \leq D_{total}; D_{P\&S} += D_{total}/10$
2: Q_m: priority queue of VMs for Map tasks
3: Q_r: priority queue of VMs for Reduce tasks
4: $Max_r = 0$, $Max_m = 0$, i = 1, $D_{exe} = D_{total} - D_{P\&S}$
5: Sort VMs in Qm and Qr in ascending order of energy consumption to run all Map and Reduce tasks respectively (em_{ik} , er_{ik})
6: **while** $\exists t \in M \cup R|$ t is not scheduled
7: **if** $t \in M$
8: **if** $Q_m[i].MinTime() + tm_{ti} + Max_r < D_{exe}$
9: assign t to Q_m[i] and Update Max_m
10: **else**
11: i++
12: **if** ($i > |SVM|$)
13: SCHEDULING IS IMPOSSIBLE
14: **else**■Do the same for Reduce tasks (if ($t \in R$))
15: Sort VMs in Q_m and Q_r in descending order based on their cost (Equation (13))
16: **while** Our scheduling cost + Placement & shuffle cost > Cost limit
17: **for** $i = 1$ to $|SVM|$
18: S = {t | t is assigned to Q_m[i] & t.StartTime() > P× Q_m[i].MaxTime()}
19: Move tasks of S to Qm[i+1] if the movement does not violate D_{exe}
20: ■ Do the same for Reduce tasks
21: **if** $\forall i \in SVM$: S is empty & cost limit is not met
22: NO SCHEDULING FOR CURRENT $D_{P\&S}$
23: **if** Our scheduling time + Placement & shuffle time < D_{total}
24: **if** Our scheduling energy + Placement & shuffle energy < Min_{Energy}
25: Min_{Energy}= Energy of new scheduling

$$Q_m[k].cost() = \frac{\sum_{i\in 1}^{M} tm_{ik} \times VMC[k]}{number\_of\_slots[k]}, \forall k \in SVM \tag{13}$$

The algorithm repeats lines 16 to 20 until it can modify the scheduling of previous stage to meet the cost limit. In line 18, it generates a set, S, for each machine in Q_m. S is the set of tasks which execute in the latest P percent of machine running time. Note that t.StartTime() indicates the start time of task and when this value is less than P*Q_m [i].MaxTime(), it means that the task lies in the latest P percent. Each time we move the tasks in set S from the machine to a cheaper one, provided that deadline is not violated, until we can find a schedule that meets the cost limit as well. The process is repeated for Reduce tasks in the same way. In this stage, if the S is empty for all the machines and we could not meet the cost limit yet, then the algorithm is not able to find a solution by current value of $D_{P\&S}$ so it returns to line 1 (lines 21and 22). In this algorithm, P is a tunable parameter that we can manipulate to find the best choice. In our experiments, we change the P from 10% to 100% by 10% steps. Since algorithm tests different values of $D_{P\&S}$, it compares the energy consumption of scheduling yield by each value and selects the scheduling with minimum energy consumption (lines 23 to 25).

The first *for* loop, line 1, repeats for constant number of times (10 times), and hence, can be ignored in time complexity analysis. Consequently, the two *while* loops, lines 6 to 14 and lines 16 to 20, determine the time complexity of algorithm. We use M_n as the number of Map tasks, R_n as the number of Reduce tasks, V_n as the number of VMs, and finally S_{max} as the maximum number of slots that a VM has. The time complexity of first while loop would be $O((M_n + R_n) \times V_n \times S_{max})$ and the time complexity of the second one would be $O(V_n^2 \times S_{max})$. Since these two loops are sequential, the maximum of them determines the time complexity of

whole algorithm: $Max(O((M_n + R_n) \times V_n \times S_{max}), O(V_n^2 \times S_{max}))$. Since in most cases the number of tasks is more than number of VMs, the first part of previous expression mostly determines the time complexity of algorithm i.e., $O((M_n + R_n) \times V_n \times S_{max})$

5.2 Machine Valuation Provisioning and Scheduling

The second proposed algorithm, Machine Valuation Provisioning and Scheduling (MVPS), takes a step forward and considers the both constraints simultaneously in one stage. It rates the machines according to their energy consumption and cost. Then, based on the rating, it assigns the tasks to slots of machines. In this algorithm, the three deadlines mechanism is again leveraged. The pseudo code is given in Algorithm 2. In each iteration of the main loop, it first calculates the energy and cost of each machine for Map tasks and Reduce tasks separately similar to (12) and (13). After that, it stores the results in four arrays A_{em}, A_{er}, A_{cm} and A_{cr} respectively for energy and cost of machines for Map and Reduce tasks. Then, it normalizes the values in arrays (lines 2 to 3). Then the algorithm calculates the value of each machine using previously created arrays. To rate the machines, we use parameter α to determine the share of energy and cost in the value of machine. To find the best answer, in our implementation, we increase the α from 0 to 1 by 0.01 steps. In this step, we calculate the value of machines for Map tasks and Reduce tasks separately and store the results in Q_m and Q_r (lines 5 and 6). The machines with lower value are better candidates for assigning tasks to. Hence, we sort the Q_m and Q_r in ascending order (line 7). Lines 8 to 17 are exactly as the first stage of DSPS algorithm i.e.,lines 6 to 14 of it. In this part, the focus of algorithm is on assigning the tasks to slots of VMs in order to meet the deadline. When the scheduling is done, it is evaluated against the cost limit. If it can satisfy the cost limit and use less energy than current minimum energy consumption, then we accept it as the new schedule and update the minimum energy consumption (lines 20 to 23). Otherwise, we ignore it and repeat lines 4 to 17 by one step increased. If the α is one and algorithm still could not find a solution, it means scheduling is not possible for current $D_{P\&S}$ so it starts from line 1 again (lines 18 and 19).

The first *while* loop of DSPS repeats in MVPS. So, the time complexity of *while* loop of MVPS, lines 9 to 17, is again $O((M_n + R_n) \times V_n \times S_{max})$. The *for* loop in line 4, which the *while* loop is nested within, repeats for 100 times. Hence, we can ignore it in time complexity since it repeats for a constant number of times. the other expression that might contribute in time complexity of MVPS, is the sort operation of line 7. the time complexity of this sort can be considered as $O(V_n \times logV_n)$. Consequently, the time complexity of MVPS is the maximum of time complexity of while loop and sort operation: $Max(O((M_n + R_n) \times V_n \times S_{max}), O(V_n \times logV_n))$. Note that here, we again ignore the first *for* loop. It is very exceptional that the logarithm of the number of VMs would be greater than the number of tasks multiplied by the maximum number of slots. Hence, the first part of Max expression certainly determines the time complexity of algorithm.

6 Evaluation of Proposed Approach

For evaluating the proposed algorithms, we used real world MapReduce workloads from MapReduce applications and datasets from PUMA benchmark suit [11]. We executed the MapReduce applications on VMs and measured information about the execution time and power consumption of their Map and Reduce tasks. Then,

Algorithm 2 *Machine Valuation Provisioning and Scheduling*

for $D_{\text{P\&S}} = D_{\text{total}}/10; D_{\text{P\&S}} \leq D_{\text{total}}; D_{\text{P\&S}} += D_{\text{total}}/10$
  Creating arrays A_{em}, A_{cm}, A_{er}, A_{er}; for energy and cost
  Normalize the arrays, D_{exe} = D_{total} - $D_{P\&S}$
  for $\alpha = 0, \alpha \leq 1, \alpha += 0.01$
    $Q_m[k] = A_{em}[k] \times \alpha + A_{cm}[k] \times (1 - \alpha), \forall k \in SVM$
    $Q_r[k] = A_{er}[k] \times \alpha + A_{cr}[k] \times (1 - \alpha), \forall k \in SVM$
    Sort Q_m and Q_r in ascending order
    Max_r = 0, Max_m = 0, i = 1
    while $\exists t \in M \cup R|$ t is not scheduled
      if $t \in M$
        if $Q_m[i].MinTime() + tm_{ti} + Max_r < D_{exe}$
          assign t to Q_m[i] and Update Max_m
        else
          i++
          if ($i > |SVM|$)
            SCHEDULING IS IMPOSSIBLE
      else■Do same for Reduce tasks (if ($t \in R$))
  if $\alpha = 1$ & a valid solution is not found yet
    NO SCHEDULING FOR CURRENT $D_{P\&S}$
  if Our scheduling time + Placment & shuffle time $< D_{total}$
    if cost of new schedule $<$ cost limit
      if energy of new schedule $< Min_{Energy}$
        Min_{Energy}= Energy of new scheduling

we used simulations to assess the proposed algorithms. The flow of our evaluation is the same as previous works such as [9]. We compare the proposed algorithms against several rivals as below:

- EM: Baseline algorithm we call Energy Minimization (EM) focuses on energy consumption and solely tries to decrease it as much as possible without considering cost limit and deadline. So, it may violate them.
- Makespan: The other rival is Makespan Minimization (MSPAN). It tries to minimize the makespan of MapReduce jobs while satisfying the cost limit and is employed in a large body of MapReduce scheduling researches [20], [21], [22]. Since the emphasis of MSPAN is on minimizing makespan, it does not pay attention to energy consumption.
- EMRSA: Energy-aware MapReduce Scheduling Algorithm [9], is a recently introduced algorithm whose aim is to minimize the energy consumption while meeting the deadline of jobs. This algorithm does not take into account the cost of job in scheduling. The original algorithm assumes that the machines and their specifications are given and fixed, so it does not propose any resource allocation or VM selection mechanism. But for fairness in comparisons, we have extended this algorithm and added resource allocation mechanism to it the same as our algorithms. Furthermore note that EMRSA algorithm only considers the Map and Reduce phases of MapReduce applications in its system model but does not take into account the data placement and data shuffle stages.
- EMRSAPS: To address the issue of not considering data placement and shuffle stages in EMRSA algorithm, we introduce EMRSAPS (EMRSA plus), enhanced version of EMRSA, which considers those stages as well.

- ILP and LP: Finally, the results of ILP and LP are presented to demonstrate how much the results of our algorithms are close to the optimal solution (ILP solution) and the lower bound (LP solution) of the objective function. It is noteworthy that since to the best of our knowledge there is no previous algorithm (even the above ones) addressing exactly the same problem as ours, the ILP/LP solutions is the golden solver to which we compare.

6.1 Experiments Setup

To obtain workloads energy and timing data, we deployed nine different types of VM. Hadoop version 1.2.1 is used for executing MapReduce applications. We have employed Hadoop 1 in our work due to its compatibility with PUMA benchmark suit. However, the system model we have used in our work for MapReduce (see Fig. 1) is applicable to both Hadoop 1 and Hadoop 2. In other words, we have data placement, Map phase, data shuffle, and Reduce phase in all the MapReduce applications irrespective of the Hadoop version. YARN in Hadoop 2 is responsible for managing the resources and uses containers instead of Map/Reduce slots of earlier versions of Hadoop. However, it has no effect on the flow of MapReduce applications, and hence, our approach works for newer versions of Hadoop that employ YARN as well.

We ran MapReduce applications with different number of tasks on the nodes and gathered the required information (execution time and energy consumption of Map and Reduce tasks) to obtain required information of the workloads. Hadoop generates logs for executed jobs and we obtain the execution time of Map and Reduce tasks from the logs. While execution time of Map tasks is predictable since they process equal sized data blocks, it seems more complex for Reduce tasks because their input data size is not usually known in advance. The default hash partitioner of Hadoop addresses this concern by trying to uniformly distribute intermediate data between Reduce tasks. We have used this partitioner and it could successfully balance the data between Reduce tasks. Moreover, many methods such as [23] are presented to mitigate the data skew in Reduce phase and make the execution time of Reduce tasks more predictable, if hash partitioner is not applicable.

HDFS replicates each block three times by default for improving the reliability of the system in the presence of failures and mitigating the impact of stragglers using speculative execution. However, Hadoop lets the user decide on the number of replications because it wants to help users with different objective functions (e.g., reliability, energy consumption, and cost) to achieve their goals, and hence, three replications is not a firm requirement in Hadoop. In our work, we consider one instance for each data block to reduce the time, and consequently, energy consumption and cost of data placement stage. But, the time of extra replications can be easily added to current formulation by slightly modifying (5) and add the time of extra replications to the data placement time of each VM. How to distribute the extra replications among VMs is an interesting direction for future work.

Since failures and speculative execution happen during runtime, we do not consider them in our work. Our approach is static and is applied before Hadoop starts copying the data and launching the tasks, and hence, failure and speculative execution are out of scope of our work. Other static approaches such as [7], [9] do not consider failures and speculative execution too. But, dynamic approaches such as [24] take them into account.

For measuring the power consumption, and consequently the energy consumption of each task, we use the built in power meter of server (HP DL380 G6) and subtract the idle power of server from its power when running the VMs in order to obtain the energy consumption of VMs. In our measurements, all the underlying physical servers are considered homogeneous. Hence, energy consumption of VMs, when executing a task, is the same on different physical servers. However, in a heterogeneous environment, the energy consumption of VMs might be different on each server type. Consequently, the energy consumption profiling should be conducted for different physical server types, or alternatively, a relationship between energy consumption of a VM on different server types should be investigated. We implemented the proposed algorithms as well as the rivals in C++. The ILP and LP solutions of problems are obtained using GAMS 23.5.2 [25]. Prior to presenting the results of experiments, we present the VM types and the workloads in more details.

VM Instances. The VM specifications are obtained from Google Cloud [12] and are presented in Table 2. The numbers of Map slots and Reduce slots are considered equal to the number of VMs cores. We chose these VM types since the resources available in our physical servers could only cover these cases. We have used VM specifications from Google Cloud because we wanted to have valid reference for hardware specifications (e.g., RAM, CPU) and monetary cost of VMs. This information is only available for public clouds and we could not find any information about VM configurations of private or hybrid clouds.

Workloads. We have chosen eight MapReduce applications from PUMA benchmark suit as workloads in our experiments (the abbreviated form of application name is followed in parentheses which are used later in the plots):

- Adjacency-List (Adjlist): for each vertex of a graph, produces the list of its neighbors. The results can be used in algorithms such as PageRank.
- Classification (Classif): uses the data of movie rating and classifies the movies into clusters.
- Histogram-Movies (HisMov): gets the movie rating data as input and generates the histogram of movies.
- Inverted-Index (InvInd): generates the mapping of word to document for a set of documents and their constituent words.

TABLE 2
Specifications of the VM Types We Used from Google Cloud [12]

VM Type	CPU (#)	Mem (GB)	Map Slot (#)	Reduce Slot (#)	Cost ($/h)
n1-standard-1	1	3.75	1	1	0.038
n1-standard-2	2	7.5	2	2	0.076
n1-standard-4	4	15	4	4	0.152
n1-highmem-2	2	13	2	2	0.096
n1-highCPU-2	2	1.8	2	2	0.058
n1-highCPU-4	4	3.6	4	4	0.116
n1-highCPU-8	8	7.2	8	8	0.232
n1-highCPU-16	16	14.4	16	16	0.464
g1-small	1	1.7	1	1	0.021

TABLE 3
Workloads Characteristics

Application	Block Size (MB)	Small-scale Workloads					Medium-scale Workloads					Large-scale Workloads				
		Map (#)	Reduce (#)	CostLimit (cent)	Deadline (s)	InputSize (GB)	Map (#)	Reduce (#)	CostLimit (cent)	Deadline (s)	InputSize (GB)	Map (#)	Reduce (#)	CostLimit (cent)	Deadline (s)	InputSize (GB)
Adjlist	170	7	8	2.5	300	1.16	70	80	11.5	2500	11.6	140	160	130	12500	23.2
Classif	256	24	0	0.5	150	6	120	0	6	600	30	240	0	58.33	2000	60
HisMov	128	16	0	0.1666	100	2	240	0	3.5	350	30	480	0	33.33	2000	60
InvInd	256	17	2	0.9	350	4.25	100	16	10	1800	25	200	32	100	7500	50
RanInvInd	215	13	10	2.5	400	2.73	130	100	16.66	2000	27.3	260	200	200	9000	54.6
Selfjoin	205	9	6	1.5	250	1.8	135	90	10	1800	27	270	180	135	13500	54
SeqCou	256	6	8	1	180	1.5	60	80	8.33	600	15	120	160	75	5000	30
WorCou	256	17	2	1.5	350	4.25	100	16	9.5	1400	25	200	32	90	7500	50

- Ranked-Inverted-Index (RanInvInd): for a list of words and number of their repetitions in documents, generates the decreasing list of documents that given words appears in them.
- Self-Join (Selfjoin): gets the association of k fields and generates it for k+1 fields.
- Sequence-Count (SeqCou): finds the number of all individual sets of three successive words
- Word-Count (WorCou): calculates the number of appearance of each word in a set of documents

We have generated three sets of workloads for each application above: small, medium and large scale. For input data of workloads, we have used the same data sets provided by PUMA benchmark suit for each application. The characteristics of each workload such as number of Map and Reduce tasks, volume of input data, the size of data block that each Map task must process, and cost limit and deadline determined for each job by user of jobs are presented in Table 3. As can be seen, some workloads do not have Reduce tasks. That means in these workloads, only one Reduce task with a very short execution time exists which is discarded. For each workload, the execution time and energy consumption of tasks on every type of VM is obtained using the mechanism described at the beginning of this subsection. We have implemented Tree topology for network. In these experiments, each VM has its own virtual network port with specific bandwidth. Same as some previous works [6] we do not consider interference between VMs that are co-located on the same physical machine. However, for future works, we intend to consider it too.

TABLE 4
Number of Availabe VMs from each VM Type Considered in Our Experiments for each Workload

VM Type	Workloads		
	Small	Medium	Large
n1-standard-1	2	3	6
n1-standard-2	2	1	2
n1-standard-4	1	1	2
n1-highmem-2	1	1	2
n1-highCPU-2	1	1	2
n1-highCPU-4	1	1	2
n1-highCPU-8	1	1	2
n1-highCPU-16	1	1	2
g1-small	1	2	4

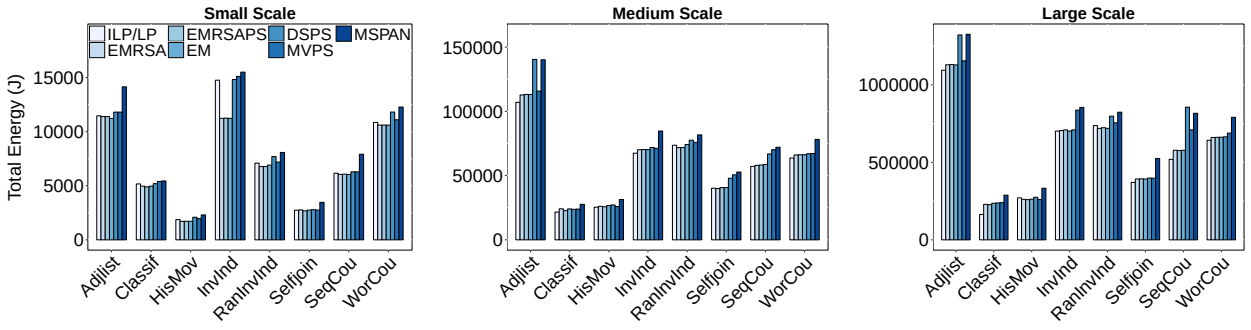


Fig. 4. Total energy consumption of algorithms for different workload sizes.

6.2 Experimental Results

In this subsection, three categories of result are depicted: Energy, Cost and Deadline. Each algorithm is given as input the number of Map and Reduce tasks, cost limit, and deadline of workload as Table 3, as well as the execution time and energy consumption of each task on every type of VM. Since the proposed algorithms are designed for the cloud, they need to be aware of accessible resources in the form of maximum available number of instances for each VM type. This could be relaxed to very high numbers (virtually infinity) in case of large public cloud providers such as Amazon and Google, but for private and hybrid clouds this is another constraint which is determined by the cloud provider computing capacity at the time of job submission. In experiments, for each workload we have considered the number of available VMs from each type as Table 4.

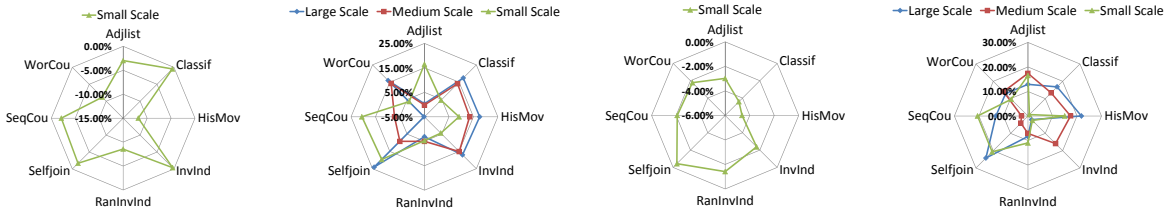
6.2.1 Energy

In this subsection we compare the algorithms regarding total energy consumption. The results are illustrated in Fig. 4 for all three categories of workloads (small, medium, large).

As can be seen in Fig.4, in small scale workloads, the total energy consumption of EMRSA, EMRSAPS, and EM in a number of applications such as InvInd is less than even the solution obtained by ILP. Since EMRSA and EMRSAPS do not consider cost limit and EM ignores both cost limit and deadline, their provided solution violates those limits but instead their energy consumption is less than ILP which satisfies both constraints; EMRSAPS violates cost limit by 75.44% and EM violates cost and deadline limits by 17.9% and 8.92% respectively on average in this case; more details of these violations are provided in Sections 6.2.2 and 6.2.3. However, the results are different in medium and large scale workloads. As mentioned before, since the ILP cannot find the solution for medium and large scale workloads in a reasonable time, we use LP instead of ILP. LP finds the lower bound solution. Regarding our proposed algorithms, DSPS and MVPS, results show that they perform better than MSPAN. However, they cannot conquer EMRSA and EM because DSPS and MVPS consider both cost limit and deadline, whereas EMRSA and EM ignore one or both of them. MSPAN is the most inferior algorithm in terms of energy consumption because it only focuses on reducing makespan and satisfying cost limit, but does not consider energy. In the following paragraphs, we analyze the results in more details. Details of energy improvement comparison are presented in Fig. 5

In small scale workloads, DSPS can reduce energy compared with MSPAN up to 20.56% for SeqCou and 10.42% on average. Comparing MVPS with MSPAN, the maximum energy saving is 20.64% for Selfjoin and the average is 12.08%. To evaluate how close the results of proposed algorithms are to the optimal solution, we compare them with ILP. In applications InvInd, Classif and SeqCou, the DSPS algorithm shows remarkable performance and increases the energy consumption only by 0.39%, 0.46% and 2.08% respectively compared with ILP. The average energy consumption of DSPS is 4.6% higher than ILP and the MVPS algorithm increases the energy consumption about 2.56% on average compared with ILP. The significance of this algorithm can be seen in Selfjoin, RanInvInd, and SeqCou where it exceeds ILP by only 0.43%, 1.41%, and 2.08% respectively. These results confirm our algorithms are performing very well, and little improvement opportunity is left to further extract.

In medium scale workloads, DSPS demonstrates the best performance against MSPAN in InvInd with reducing energy by 15.1%. The average energy reduction is 9.77%. The average energy reduction of MVPS compared with MSPAN is 11.48% and it shows the highest reduction in Adjlist by 17.4% reduction. Finally, in large scale workloads, the MVPS outperforms MSPAN up to 24.22% (Selfjoin) and 14.02% on average. The DSPS algorithm obtains maximum energy reduction in Selfjoin by 24.1%. The average energy reduction compared with MSPAN is 11.33%.



(a) DSPS vs ILP (b) DSPS vs MSPAN (c) MVPS vs ILP (d) MVPS vs MSPAN

Fig. 5. Comparing the energy improvement of DSPS and MVPS against ILP and MSPAN

6.2.2 Cost

While energy is definitely an important factor in cloud, as discussed in Section 1, the cost limit and the deadline submitted for the job are also two critical concerns to adhere to as determined by the contract or SLA signed between the provider and the user. Thus, for the problem statement addressed in this paper, scheduling algorithms such as EMRSA and EM are not practical. Below, the experimental results to evaluate the algorithms in terms of cost limit are presented to quantify the cost overshoots of these algorithms.

In this subsection, we present monetary cost of VMs for each algorithms schedule and compare them with the cost limit specified for each workload given in Table 3. Total cost for each algorithm in different workloads is depicted in Fig. 6. The first obvious observation regarding Fig.6 is that the EMRSA and EM algorithms usually violate the cost limit and this is the reason behind their better energy improvement shown in previous subsection. The average and maximum cost violation of EM, which is cost- and deadline-oblivious, in different workloads is (respectively): 17.9% and 90.1% in small scale, 13.78% and 83.75% in medium scale, and 6.1% and 60.91% in large scale workloads.

Similarly, the average cost violation of EMRSA in small, medium and large scale workloads is 36.33%, 52.21%, and 29.2% respectively. In large scale workloads, the EMRSA violates the cost limit in seven out of eight applications and maximum violation occurs in Classif by 79.14%. In medium scale workloads, the highest cost violation occurs in Classif by 122.06%. Finally, for the small scale workloads, the worst case violation occurs for RanInvInd by 119.53% cost violation. As expected, our proposed algorithms, DSPS and MVPS, as well as MSPAN always satisfy the cost limit in all the workloads.

6.2.3 Deadline

Finally, we analyze the performance of algorithms in terms of meeting the given deadline. User, who submits the job, determines the deadline too. We compare the performance of algorithms in terms of execution time of scheduled job with each other as well as with deadline. The results related to total execution time of each workload under different algorithms scheduling are presented in Fig. 7.

As the Fig.7 indicates, algorithms that cannot always meet the deadlines are EMRSA and EM. Note that since the EMRSA does not consider placement and shuffle time, it cannot meet the deadline. However, our improved version of it, EMRSAPS, is able to always meet the deadline. The average deadline violation of the EM for small scale, medium scale, and large scale workloads is 8.92%, 91.53%, and 78.91% respectively. The maximum deadline violation occurs in RanInvInd by 41.77% for small scale workloads, in SeqCou By 178.5% for medium scale workloads and again in RanInvInd by 190.43% for large scale ones. Unlike the EM, the MSPAN algorithm demonstrates significant performance in terms of execution time. As can be seen, the MSPAN algorithm not only meets the deadline in all situations, but also it decrease the execution time way below the deadline. Our two proposed algorithms DSPS and MVPS as well as EMRSAPS, always meet the deadline. However, they do not decrease the execution time in the way MSPAN does. The reason is that since their aim is decreasing the energy consumption instead of execution time, they prefer energy efficient, yet slow, machines rather than fast but energy-hungry ones.

6.2.4 VM Selection and Task Assignment Details

In the following we show which VMs are selected by each algorithm from the set of available VMs. Then, we demonstrate how the tasks are assigned to selected VMs. The execution time of each phase (placement,

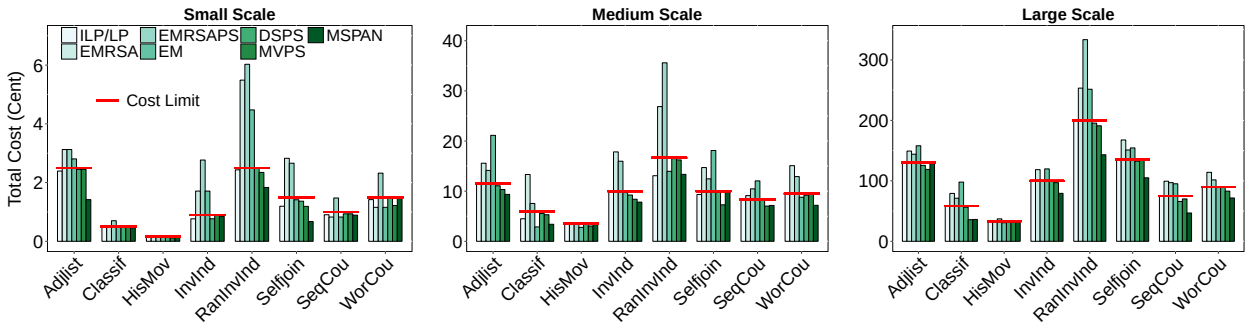


Fig. 6. Total cost of each algorithm as well as cost limit violation.

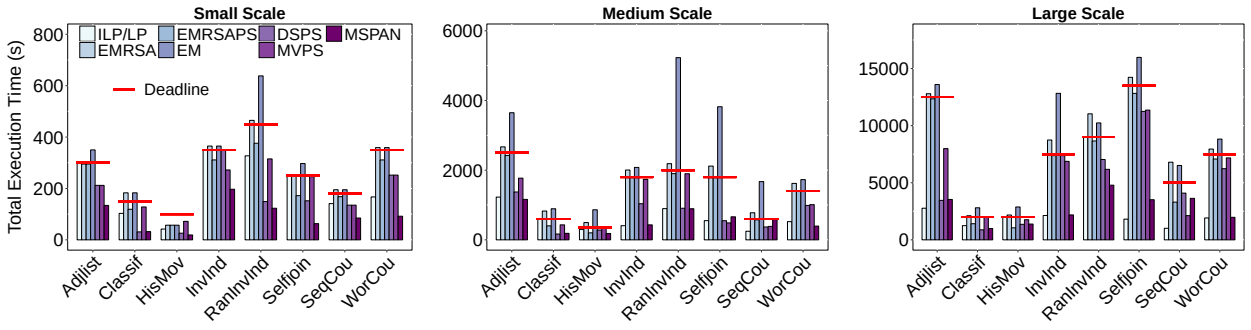


Fig. 7. Deadline violation of each algorithm.

Map, shuffle, and Reduce) is also illustrated. Fig. 8 presents the aforementioned information for small scale workloads. We compare our proposed algorithms against makespan and ILP. The vertical axis indicates the selected VMs. Horizontal axis illustrates the time. Within an algorithm, duration of placement (red) and shuffle (blue) phases are the same for all the VMs; however the Map phase and Reduce phase have different lengths depending on the number of tasks on each VM. The number inside the Map and Reduce rectangles shows the number of tasks that the VM executes. For example in WorCou application, DSPS algorithm has selected two VMs from N1-standard-1 type, one VM from N1-highCPU-4 and one from G1-small (see Table 2 for VM types). WorCou application has 17 Map tasks and two Reduce tasks in small scale (see Table 3). N1-highCPU-4 has executed 12 Map tasks and 2 Reduce tasks. N1-standard-1 VMs have executed two and one Map tasks and finally the G1-small has executed two Map tasks.

Investigating the behavior of ILP, which finds the optimal solution, indicates that 1) while ILP tries to reduce the number of selected VMs (on the contrary of MSPAN), it tries to distribute the tasks uniformly on VMs so they finish their job at the same time. 2) ILP chooses N1-highCPU-4 and N1-highCPU-2 VM types more than others. We can conclude that these VM types are more energy efficient than other ones. Generally, tracing the VM selection pattern of ILP can help identify the energy efficient VM types. Exploring the results of DSPS and MVPS and comparing them with ILP, reveals that there is still room for improvement in these algorithms regarding VM selection and task assignment.

6.3 Sensitivity Analysis

We first evaluate the effect of network bandwidth on energy consumption. Since the bandwidth can affect the duration of data placement and data shuffle stages, we change it from default value of 1GB to 100MB and then 10GB to see how their share in total energy consumption changes. We present the results of ILP for small scale workloads in Fig. 9. Increasing the bandwidth, decreases the data placement and data shuffle times and consequently their share in total energy consumption. The results show that the data movements on the network are responsible for consumption of an important share of total energy in most networks, and hence, they must be taken into account to obtain a practical solution for VM provisioning and scheduling.

Regarding deadline, we have changed it from 120 seconds to 743 seconds , and the cost limit starts from 0.033 cents to 3.263 cents. The normalized results for the two proposed algorithms, DSPS and MVPS, with

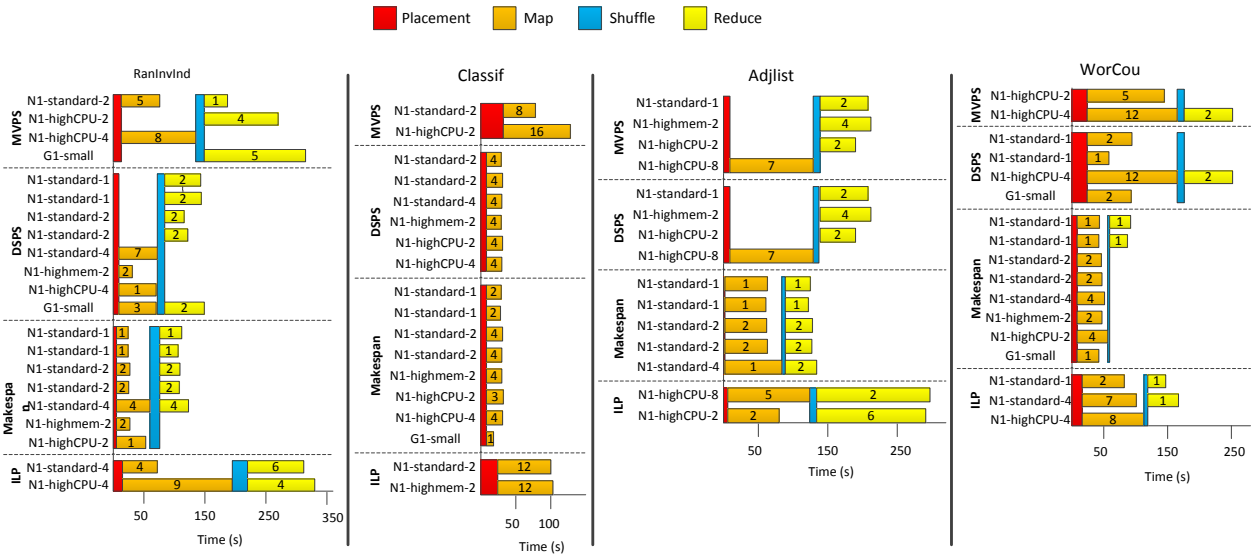


Fig. 8. VM selection and task assignment details of 4 applications.

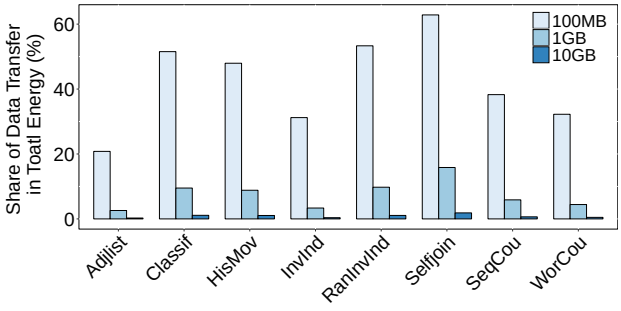


Fig. 9. Network sensitivity analysis.

respect to ILP are presented in Fig. 10. As can be seen, for some deadlines and cost limits, there is no solution and consequently no point is plotted in the figure. The plots show that beyond a certain amount of relaxing the deadline or cost limit, our algorithms converge to ILP in most cases. The other information these figures reveal is that in some cases, at very tight deadline or cost limits, our algorithms fail to find a feasible solution despite its existence. This warrants further work and analysis for further improvement.

7 Conclusion and Future Work

We proposed two algorithms, DSPS and MVPS, to tackle the problem of MapReduce batch jobs scheduling in cloud. These two algorithms reduce energy consumption while satisfying the cost limit and deadline determined by the customer. Extensive set of simulations using real world workloads demonstrate that the proposed algorithms can find near optimal solutions for the problem. As workloads, we used eight applications from PUMA benchmark suit in three different scale classes of small, medium and large. To find the optimal

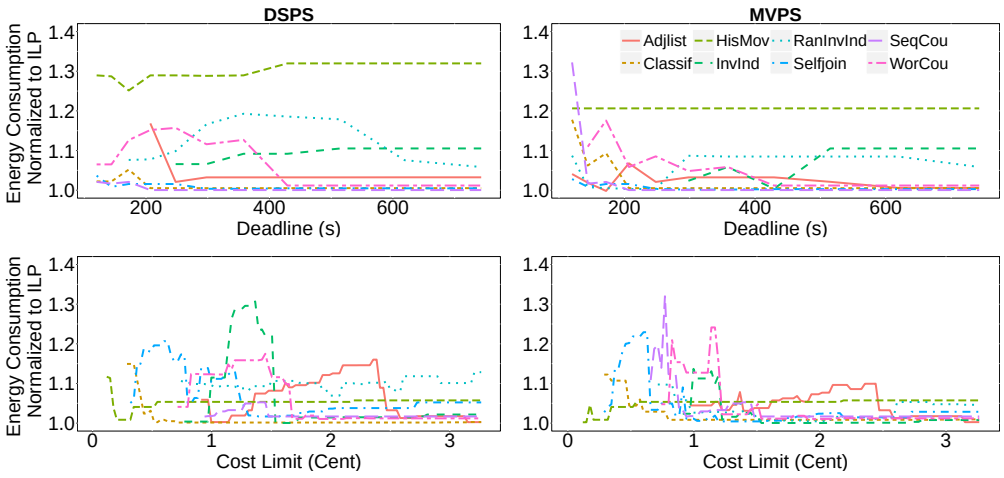


Fig. 10. Cost and Deadline Analysis.

and lower bound solutions, we use ILP and LP respectively. Comparing DSPS and MVPS with several rivals indicates that our algorithms can find valid and energy efficient schedules.

As directions for further research, other requirements such as dependability can also be considered as a constraint for the problem and added in terms of new set of constraints. Collocation of VMs on a server and the resulting resource contention among them is another impact of using clouds for Big Data computing. This interference can affect the overall performance of the cluster responsible for executing Big Data application. Addressing this issue is part of our future work in extending the techniques presented in this paper. Considering soft deadline instead of hard deadline for provisioning and scheduling is another interesting direction for future work.

REFERENCES

[1] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the future*, vol. 2007, pp. 1–16, 2012.

[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun ACM*, vol. 51, pp. 107–113, 2008.

[3] "Apache hadoop," https://hadoop.apache.org, acs: 2018-08-06.

[4] "Amazon emr," http://aws.amazon.com/elasticmapreduce, acs: 2018-08-06.

[5] C. Chen, J. Lin, and S. Kuo, "Mapreduce scheduling for deadline-constrained jobs in heterogeneous cloud computing systems," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 127–140, 2018.

[6] B. Palanisamy, A. Singh, and L. Liu, "Cost-effective resource provisioning for mapreduce in a cloud," *IEEE Transactions on parallel and distributed systems*, vol. 26, pp. 1265–1279, 2015.

[7] X. Xu and M. Tang, "A new approach to the cloud-based heterogeneous mapreduce placement problem," *IEEE Transactions on Services Computing*, vol. 9, no. 6, pp. 862–871, 2016.

[8] D. Cheng, P. Lama, C. Jiang, and X. Zhou, "Towards energy efficiency in heterogeneous hadoop clusters by adaptive task assignment," in *IEEE ICDCS'15*, 2015, pp. 359–368.

[9] L. Mashayekhy, M. Nejad, D. Grosu, Q. Zhang, and W. Shi, "Energy-aware scheduling of mapreduce jobs for big data applications," *IEEE Transactions on parallel and distributed systems*, vol. 26, pp. 2720–2733, 2015.

[10] S. M. Nabavinejad and M. Goudarzi, "Data locality and vm interference aware mitigation of data skew in hadoop leveraging modern portfolio theory," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, 2018, pp. 175–182.

[11] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "Puma: Purdue mapreduce benchmarks suite," 2012.

[12] "Google cloud," https://cloud.google.com, acs: 2018-08-06.

[13] Y. Yang, J. Xu, F. Wang, Z. Ma, J. Wang, and L. Li, "A mapreduce task scheduling algorithm for deadline-constraint in homogeneous environment," in *Second International Conference on Advanced Cloud and Big Data (CBD)*. IEEE, 2014, pp. 208–212.

[14] N. Maheshwari, R. Nanduri, and V. Varma, "Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework," *Future Generation Computer Systems*, vol. 28, pp. 119–127, 2012.

[15] X. Ma, X. Fan, J. Liu, H. Jiang, and K. Peng, "vlocality: Revisiting data locality for mapreduce in virtualized clouds," *IEEE Network*, vol. 31, no. 1, pp. 28–35, 2017.

[16] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 407–420, 2015.

[17] "Design a pluggable interface to place replicas of blocks in hdfs," https://issues.apache.org/jira/browse/HDFS-385, accessed: 2018-08-06.

[18] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "Mapreduce with communication overlap (marco)," *Journal of Parallel and Distributed Computing*, vol. 73, pp. 608–620, 2013.

[19] B. Krishnan, H. Amur, A. Gavrilovska, and K. Schwan, "Vm power metering: feasibility and challenges," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 3, pp. 56–60, 2011.

[20] Y. Zhu, Y. Jiang, W. Wu, L. Ding, A. Teredesai, D. Li, and W. Lee, "Minimizing makespan and total completion time in mapreduce-like systems," in *IEEE INFOCOM*, 2014, pp. 2166–2174.

[21] S. Tang, B.-S. Lee, and B. He, "Dynamic job ordering and slot configurations for mapreduce workloads," *IEEE Transactions on Services Computing*, vol. 9, no. 1, pp. 4–17, 2016.

[22] S. M. Nabavinejad and M. Goudarzi, "Faster mapreduce computation on clouds through better performance estimation," *IEEE Transactions on Cloud Computing*, 2017, in press.

[23] Q. Chen, J. Yao, and Z. Xiao, "Libra: Lightweight data skew mitigation in mapreduce," *IEEE Transactions on parallel and distributed systems*, vol. 26, no. 9, pp. 2520–2533, 2015.

[24] H. Fu, H. Chen, Y. Zhu, and W. Yu, "Farms: Efficient mapreduce speculation for failure recovery in short jobs," *Parallel Computing*, vol. 61, pp. 68–82, 2017.

[25] "General algebraic modeling system (gams)," https://www.gams.com/, accessed: 2018-08-06.

Seyed Morteza Nabavinejad is a Postdoc at School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. He received the B.Sc. degree from Ferdowsi University of Mashhad and the M.Sc., and Ph.D. degrees from Sharif University of Technology, in 2011, 2013, and 2018, respectively. His research interests include big data processing, cloud computing, and energy aware datacenters.

Maziar Goudarzi joined Sharif University of Technology in 2009 where he is now an associate professor of computer engineering. Before then, he was a research associate professor at Kyushu University, Japan, and then a member of research staff at University College Cork, Ireland. His current research interests include architectures for warehouse-scale computers, green computing, and hardware-software co-design.

Saeed Abedi is currently a Ph.D. student at the Department of Computer and Information Science, University of Pennsylvania. His research mainly focuses on the timing guarantee of virtual network functions in data centers. He is also interested in distributed systems, fault tolerance, and cloud computing. He received his BSc in 2016 from the Sharif University of Technology, Iran.