The Journal of Supercomputing (2021) 77:6788–6816
https://doi.org/10.1007/s11227-020-03516-3

Constraint programming versus heuristic approach to MapReduce scheduling problem in Hadoop YARN for energy minimization

Vaibhav Pandey[1] · **Poonam Saini**[1]

Accepted: 16 November 2020 / Published online: 4 January 2021

Abstract
In this paper, we consider a deadline-constrained MR scheduling problem of minimizing energy consumption in Hadoop's generic resource manager known as yet another resource negotiator. The problem has been modeled as an integer programming (IP) problem using the time-indexed decision variables. We propose two solution approaches to the problem. First, we give a heuristic algorithm that generates sub-optimal schedules in polynomial time. Second, we propose a novel constraint programming (CP) model (as an alternative to the IP model) which always generates optimal schedules when solved by a CP solver. The CP technique is a relatively new and an alternative approach to IP-based branch-and-cut algorithm to exactly solve NP-hard optimization problems. We performed several experiments to compare both proposed solution approaches over real data traces of a wide variety of MR jobs from the HiBench and PUMA benchmark suite. It is noticed that for large-scale big data jobs, the heuristic algorithm provides sub-optimal results in a very small amount of time. On the other hand, the CP approach not only gives optimal results but also takes a small amount of time when compared to IP-based approaches. Therefore, it can be used in non-time-critical situations for getting an optimal schedule. Besides this, a few experiments were also performed to compare the tightest satisfiable deadline under both approaches with the conclusion that the CP technique is able to produce optimal schedules in tighter deadline constraints than the heuristic approach. Moreover, we investigate the sensitivity of total energy consumption of tasks and the execution time of both approaches separately on the number of tasks and deadlines.

Keywords Constraint Programming · Energy efficient · MapReduce Scheduling · Integer Linear Programming · Hadoop YARN

✉ Vaibhav Pandey
pandeyvaibhav51@gmail.com

Poonam Saini
poonamsaini@pec.edu.in

[1] Department of Computer Science and Engineering, Punjab Engineering College (Deemed to be University), Chandigarh 160012, India

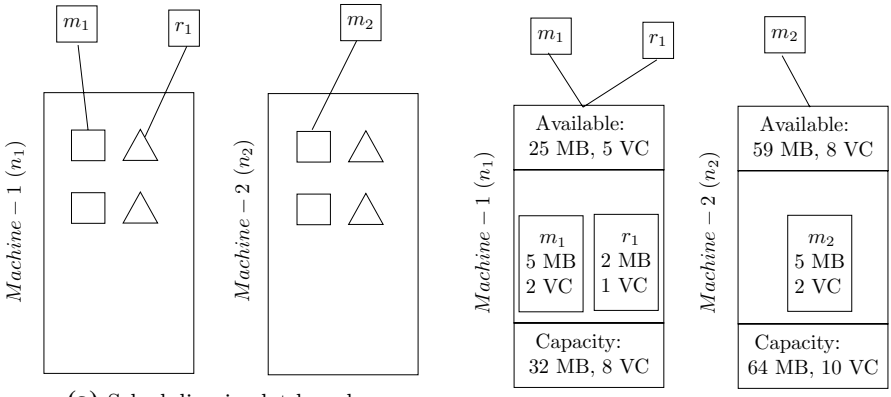
1 Introduction

Hadoop is an open-source implementation of the MapReduce (MR) computing paradigm [1, 2]. The performance of Hadoop is greatly influenced by its scheduler which has to cater to various quality of service (QoS) requirements of two stakeholders, namely Hadoop *user* and *administrator*. Based on the scheduling entity, the Hadoop scheduler can be classified as a job-level or a task-level scheduler [3, 4]. At the job-level, the scheduler has many jobs at hand to decide which job is going to be executed at what time. And in case of offline scheduling (at job-level), it simply arranges the jobs in a particular order [5–7]. Whereas at task-level, various map and reduce tasks of the job(s) are scheduled/assigned on different cluster machines to achieve some objective [8–10]. Mathematically, the problem of MR scheduling (at either level) for optimizing one or more QoS parameters simultaneously is strongly NP-hard (Non-deterministic Polynomial time-hard) under different system models and constraints [5, 10–12]. In practice, two variants of the Hadoop framework are available: slot-based and container-based [13]. The container-based Hadoop uses a generic resource manager known as Yet Another Resource Negotiator (YARN) which overcomes the limitations of slot-based Hadoop in terms of scalability, reliability, and resource utilization by introducing the concept of containers [13].

In the past few years, MapReduce (MR) scheduling has been a challenging area of research to minimize energy consumption. Mashayekhy et al. in [8] and Yousefi et al. in [9] have considered an energy-efficient MR scheduling problem at the task level to minimize the energy consumption of a single MR job. The authors considered a slot-based Hadoop environment where the various map and reduce tasks have to be scheduled on suitable energy-efficient computing slots so that overall energy consumption is minimized and the job may complete its execution before a user-specified deadline. In this paper, we consider the same problem, however, exclusively for the YARN environment and call it as the problem of deadline-constrained MR scheduling in YARN for energy minimization (DMRSYE). The problem has been formally formulated as an integer program in Sect. 3. Next, we discuss how scheduling in a YARN environment is different from slot-based Hadoop. This difference makes the considered problem very challenging and also servers as our motivation.

1.1 MR scheduling in slot-based versus container-based Hadoop

Scheduling under both variants of Hadoop (whether at job level or task level) differs in the way various computing resources of a node are allocated to map and reduce tasks. This happens due to different mechanisms of slots and containers, that further influences the (mathematical) scheduling model of both versions. In slot-based Hadoop, resources are allocated to tasks at the level of a fixed-size partition of nodes, called as slot. At each node, a static and pre-defined number of slots are



(a) Scheduling in slot-based Hadoop

(b) Scheduling in container-based Hadoop

Fig. 1 MR scheduling in Hadoop

created separately for map and reduce tasks,[1] which are specifically called as map and reduce slots, respectively. During the execution, map and reduce tasks are to be scheduled on their respective computing slots with a restriction that a reduce task may start only after all map task has completed its execution. In this scenario, slots act as independent parallel machines.

Figure 1a shows the scheduling process in a slot-based Hadoop cluster comprising two machines n_1 and n_2 where two map tasks m_1 and m_2, and a single reduce task r_1 are being scheduled. Two map slots (represented by squares) and two reduce slots (represented by triangles) are created per machine. And any slot (either map or reduce) represents one-fourth portion of computing resources in a node. Task m_1 and r_1 are scheduled on one of the respective slots in machine n_1, whereas task m_2 is scheduled on one of the map slots in machine n_2. In this particular instance of the Hadoop cluster, a maximum of two map tasks (equals to the number of map slots) can be scheduled on any machines parallelly at any time. Given this, the slot-based MR scheduling problem simply reduces to a traditional parallel machine scheduling problem (PMSP). The slot-based MR scheduling problem has been modeled as a PMSP using mathematical programming (MP) techniques in [5, 12, 14–17] for different objectives.[2] Particularly in [5, 17], the problem has been modeled as flexible flow shop machine scheduling.

The map and reduce tasks belonging to different jobs require a diverse amount of resources which cannot be fulfilled exactly by the slot-based mechanism. However, YARN facilitates this by allowing the tasks to place their request for required resources in the form of a vector $\overrightarrow{REQ} = \langle a_1, a_2, \ldots, a_l, \ldots, a_{|\mathcal{A}|} \rangle$, where a_l represents the amount of lth resource type requested by any particular task and $|\mathcal{A}|$ represents

[1] Equals to the number of computing cores in the node.

[2] Linear programming (LP), integer programming (IP), quadratic programming (QP), quadratic constraint programming (QCP), etc. are collectively known as mathematical programming (MP).

total resource types. Currently, YARN supports only two resource types: memory and CPU cores, and hence, $|\mathcal{A}| = 2$. If the amount of requested resources can be fulfilled by any cluster node, then the exact amount of resources are allocated to the task in the form of a container which can be thought of as a logical abstraction of all resources available at the machine. And, due to this container concept, scheduling in YARN needs special attention.

Figure 1b shows the resource allocation and scheduling of two map tasks m_1 and m_2, and a reduce task r_1 in a YARN cluster consisting of two machines n_1 and n_2. The machine n_1 has the resource capacity of 32 MB memory and 8 virtual cores (VC), whereas n_2 has 64 MB memory and 10 VCs. The tasks m_1 and m_2 have resource request vector as $\langle$5MB, 2VC$\rangle$ each and task r_1 has resource request vector as $\langle$2MB, 1VC$\rangle$. The tasks m_1 and r_1 are allocated to machine n_1 where two containers of exact resource demand are created for both tasks. Similarly, task r_1 is executed within a container of exact capacity at machine n_2. After task allocation, available resources at both nodes are shown in the diagram itself. At any particular time, simultaneously maximum four map tasks on machine n_1, and maximum five map tasks on n_2 can be scheduled. In other words, the total amount of resources consumed by all containers must not exceed the total capacity of a node.

Due to this kind of additional resource constraint, the task scheduling in YARN like environment is similar to a resource-constrained parallel machine scheduling problem (RCPMSP) [18, 19]. However, in the case of YARN, each machine has its own set of non-shared local resources, whereas, in standard RCPMSP problems, machines have a set of shared global resources. The problem investigated in [18] is a type of RCPMSP problem and proved to be an NP-hard problem. Therefore, the DMRSYE problem which is similar to [18] is also an NP-hard problem in a very strong sense. To solve any computationally intensive NP-hard problems, there are three main approaches:

- **Heuristics**: It solves the problems usually in polynomial time with a sub-optimal result without any guarantee on the sub-optimality.
- **Exact algorithms**: It solves an NP-hard problem optimally, however, takes exponential time, e.g., efficient graph search techniques, branch-and-bound (B&B), branch-and-cut (B&C), etc.
- **Approximation algorithms**: It provides a sub-optimal result with an assurance on the quality of the sub-optimal result.

1.2 Solution approaches to DMRSYE problem

In this paper, we have adopted the heuristic and exact solution approaches to the DMRSYE problem and compared them on the basis of various performance parameters. As expected, the proposed heuristic method provides sub-optimal results in polynomial time without any guarantee on the sub-optimality. The algorithm works in multiple rounds and performs a greedy search to find the best node for a task to be scheduled upon for minimizing the energy consumption. And, to the best of our

knowledge, no such static scheduling algorithm has been developed for the considered problem.

On the other hand for an exact solution, One of the options is to model it as an integer program and use either IP-based branch-and-cut/branch-and-bound (B&C/B&B) algorithms,[3] or cutting plane methods. All of them can optimally solve any NP-hard IP problem. Although it is possible to find optimal solutions using such methods, it may require an enormous amount of computation time with the increasing problem size. We used the in-built IP-based B&C algorithm of IBM ILOG CPLEX optimization studio v12.8 (CPLEX studio for short) [20, 21] to solve a small instance of considered problem and get an optimal schedule after several minutes. For large problem instances, we could not get the results even after several hours, which is absolutely not desirable for deadline critical applications. Hence, our main objective of this paper apart from designing a heuristic algorithm is to provide an efficient methodology that generates exact solutions for the DMRSYE problem and may compete with heuristic schemes in terms of solution time.

In recent years, researchers have been using the constraint programming (CP) technique as an alternative to IP-based solution approaches for NP-hard optimization problems. And, it has been observed that CP techniques are suitable for sequencing and scheduling applications besides highly constrained and strict feasibility problems [18]. The CP optimizer embedded with CPLEX studio uses the graph search algorithms to find the optimal solutions. The constraint propagation and problem-specific searching techniques may speed-up the solution process [21]. We briefly introduce CP technique in Sect. 4.2 and its applications in scheduling problem in Sect. 2. Although the CP technique has been applied in a different kind of scheduling problem, it has never been used for MR scheduling in YARN. We propose a novel CP model for the DMRSYE problem to be solved by default search algorithm of CP optimizer and compare it with our heuristic method for solution quality and computational time (i.e., schedule generation time).

1.3 Organization of the paper

The rest of the paper is organized as follows. Section 2 presents an overview of related work. Section 3 formulates the DMRSYE problem mathematically as an integer linear programming problem. Section 4 presents a heuristic algorithm and a novel CP-based scheduling model for the considered problem. Section 5 presents the experimental setup, selected workloads, and profiling techniques. Section 6 discusses the results of various experiments performed to compare both proposed approaches. Lastly, Sect. 7 concludes the paper.

[3] IP modeling is a common way to represent machine scheduling problems in general.

2 Related work

The closest works to ours are [8] and [9]. However, both of them assumed a slot-based scheduling environment and adopted heuristic approaches to solve the considered problem sub-optimally in a quick time. Particularly, Mashayekhy et al. in [8] formulated the problem of energy-aware scheduling of a single MapReduce job within a deadline as an IP problem. The authors proposed two heuristic methods called energy-aware MapReduce scheduling algorithms (EMRSAs) which take the energy efficiency differences of different machines into account. The algorithms use a metric called energy consumption rate (ECR) which specifies the rate of energy consumption of map and reduce slots and induces an order relation among the cluster machines. To further improve the performance of EMRSA, Yousefi et al. [9] proposed a better task-based greedy heuristic to establish the mapping between slots and tasks. Their heuristic minimizes the energy consumption of an MR job without significant loss in performance while satisfying user-specific service level agreements (SLAs).

Yigitbasi et al. [22] proposed an energy-efficient algorithm to schedule heterogeneous workloads to the heterogeneous cluster comprising high and low power machines. This provides an opportunity to save energy by intelligently placing jobs on its corresponding energy-efficient machine. The authors have not mathematically modeled the problem and did not consider any deadline constraint like [8, 9]. Bampis et al. [23] proposed an energy-efficient scheduling algorithm to minimize the weighted completion time of a set of n MapReduce job with a constraint of the energy budget. The authors formulated the problem as an integer program assuming that the order of job execution is not fixed. Thereafter, they derived a polynomial-time constant-factor approximation algorithm using linear programming (LP) relaxation to solve the formulated problem. The authors also developed a convex programming formulation and combined that with list scheduling algorithms.

Shao et al. [24] formulated the energy-efficient MR scheduling problem for YARN architecture as a m-dimensional knapsack problem (MKP) resulting in an integer program. Besides energy minimization, it also considered the fairness metric and proposed a heuristic approach to produce a sub-optimal solution. Cai et al. [25] proposed a YARN scheduler to minimize energy consumption with a deadline as a constraint, like [8]. The proposed scheduler works at both, job level and task level. At the job level, the scheduler is highly inspired by the automatic resource inference and allocation (ARIA) scheduler [26] and competes for jobs within its deadline. At the task level, the authors optimized energy consumption through the user-space dynamic voltage and frequency scaling (DVFS) governor.

Now, we will discuss some pointers where the CP technique has been used to solve RCPMSP problems or its variants. Edis et al. in [18, 27] and [28] presented a study comparing the application of IP and CP techniques in the parallel machine scheduling problem with additional resources for minimizing the makespan. Besides this, Ham and Andy [29] proposed a CP technique with a problem specific variable ordering heuristic for a dual resource-constrained

scheduling problem in the field of semiconductor manufacturing. Gökgür et al. [30] developed three CP models for parallel machine scheduling with tool loading constraints and showed that their proposed models are better than MP and tabu search techniques proposed in [31]. Arbaoui and Yalaoui [32] proposed a CP model for unrelated parallel machines with additional resources and compared it with two MP models proposed in [33].

3 Problem formulation

In this section, we mathematically formulate the DMRSYE problem as an integer programming (IP) problem. Some of the references given in [19, 28] have proposed different IP or mixed-integer programming (MIP) formulations for a variety of RCPMSP problems. We take motivation from those IP/MIP models and incorporate appropriate changes in our model. The following system model assumptions have been taken regarding the set of tasks, machines, resources, energy consumption, processing time, etc.:

- We consider a MR job comprising two task sets, $\mathcal{M} = \{m_1, \ldots, m_{|\mathcal{M}|}\}$ and $\mathcal{R} = \{r_1, \ldots, r_{|\mathcal{R}|}\}$, which have $|\mathcal{M}|$ number of map tasks and $|\mathcal{R}|$ number of reduce tasks, respectively. The job is to be processed by a YARN cluster comprising n heterogeneous machines (or nodes in Hadoop terminology) represented by a set $\mathcal{N} = \{n_1, \ldots, n_{|\mathcal{N}|}\}$.
- The energy consumption and processing time of a map task $m_j \in \mathcal{M}$ on machine $n_i \in \mathcal{N}$ is e^m_{ij} and p^m_{ij}, respectively. Similarly, the energy consumption and processing time of a reduce task $r_k \in \mathcal{R}$ on machine $n_i \in \mathcal{N}$ is e^r_{ik} and p^r_{ik}, respectively.
- Each machine has $|\mathcal{A}|$ types of resources represented by set $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$. The total amount of resource type $a_l \in \mathcal{A}$ available at machine $n_i \in \mathcal{N}$ is indicated by cap_{il}. Moreover, the amount of resource type $a_l \in \mathcal{A}$ requested by any map tasks m_j is req^m_{jl}, while by any reduce task r_k is req^r_{kl}.

The problem here is to schedule all MR tasks over the cluster machines in such a way so that the total energy consumption of machines is minimized. Further, it is also required that all tasks complete their execution within a user-specified deadline D while maintaining the temporal dependency between map and reduce tasks and fulfilling the resource constraint, i.e., at any particular time, no more tasks can be scheduled at any machine beyond its total capacity.

To formulate the problem as an integer program, we use two types of *time-indexed* (TI) binary decision variables X_{ijt} and Y_{ikt} which are based on discretization of time horizon in which time t ($t = 0, 1, \ldots, T$) is divided into discrete time intervals. The assignment of value 1 to a decision variable X_{ijt} (Y_{ijt}) signals that map task m_j (reduce task r_k) is assigned to machine n_i at time t. The time-indexed formulation has been previously introduced in [34, 35] for a single machine scheduling problem. The formulation **DMRSYE-IP** is given as:

Table 1 Indexes used in the DMRSYE-IP formulation

Index	Meaning
i	Index of cluster machines ($i = 1, \ldots, \lvert\mathcal{N}\rvert$)
j	Index of map task to be scheduled ($j = 1, \ldots, \lvert\mathcal{M}\rvert$)
k	Index of reduce task to be scheduled ($k = 1, \ldots, \lvert\mathcal{R}\rvert$)
l	Index of resource type ($l = 1, \ldots, \lvert\mathcal{A}\rvert$)
t	Index of time ($t = 0, 1, \ldots, T$)

$$\min \sum_{i=1}^{|\mathcal{N}|} \sum_{j=1}^{|\mathcal{M}|} \sum_{t=0}^{T-p_{ij}^m} e_{ij}^m X_{ijt} + \sum_{i=1}^{|\mathcal{N}|} \sum_{k=1}^{|\mathcal{R}|} \sum_{t=0}^{T-p_{ik}^r} e_{ik}^r Y_{ikt}$$

subject to

$$\sum_{i=1}^{|\mathcal{N}|} \sum_{t=0}^{T-p_{ij}^m} X_{ijt} = 1, \quad (\forall j = 1, \ldots, |\mathcal{M}|) \tag{1}$$

$$\sum_{i=1}^{|\mathcal{N}|} \sum_{t=0}^{T-p_{ik}^r} Y_{ikt} = 1, \quad (\forall k = 1, \ldots, |\mathcal{R}|) \tag{2}$$

$$\sum_{i=1}^{|\mathcal{N}|} \sum_{t=0}^{T-p_{ik}^r} (t + p_{ik}^r) Y_{ikt} \leq D, \quad (\forall k = 1, \ldots, |\mathcal{R}|) \tag{3}$$

$$\sum_{j=1}^{|\mathcal{M}|} \sum_{s=max(0,t-p_{ij}^m)}^{t-1} (req_{jl}^m) X_{ijs} + \sum_{k=1}^{|\mathcal{R}|} \sum_{s=max(0,t-p_{ik}^r)}^{t-1} (req_{kl}^r) Y_{iks} \leq cap_{il}, \quad (\forall i = 1, \ldots, |\mathcal{N}|; \forall l = 1, \ldots, |\mathcal{A}|; \forall t = 0, 1, \ldots, T-1) \tag{4}$$

$$\sum_{i=1}^{|\mathcal{N}|} \sum_{s=max(0,t-p_{ij}^m)}^{T-p_{ij}^m} X_{ijs} + \sum_{i=1}^{|\mathcal{N}|} \sum_{s=0}^{t-1} Y_{iks} \leq 1, \quad (\forall j = 1, \ldots, |\mathcal{M}|; \forall k = 1, \ldots, |\mathcal{R}|; \forall t = 0, 1, \ldots, T-1) \tag{5}$$

$$X_{ijt} \in \{0, 1\} \quad (\forall i = 1, \ldots, |\mathcal{N}|; \forall j = 1, \ldots, |\mathcal{M}|; \forall t = 0, 1, \ldots, T-1) \tag{6}$$

$$Y_{ikt} \in \{0, 1\} \quad (\forall i = 1, \ldots, |\mathcal{N}|; \forall k = 1, \ldots, |\mathcal{R}|; \forall t = 0, 1, \ldots, T-1). \tag{7}$$

Indices used in the formulation have meanings as shown in Table 1.

In this **DMRSYE-IP** formulation, the objective is to minimize total energy consumption. Constraints 1 and 2 are assignment constraints which ensure that each map

and reduce task to be assigned to one machine only once. Constraint 3 is a deadline constraint that requires all map tasks must finish their computation before the user-specified deadline D. Constraint 4 is a resource constraint that ensures that at any instance of time, the total amount of computing resources allocated to map and reduce tasks in the form of containers, should not exceed the total capacity of any machine. The constraint 5 establishes the temporal dependency between map and reduce tasks, i.e., all map tasks must finish their execution before the start of any reduce task. The last constraints 6 and 7 ensure that decision variables X_{ijt} and Y_{ikt} may take values either 0 or 1.

4 Proposed solution approaches

We use the in-built IP-based B&C algorithm CPLEX studio [20, 21] to the considered problem to get the optimal schedule. However, for large problem instances, we could not get the results even after several hours. Therefore, we choose to design a heuristic algorithm that can generate a sub-optimal schedule in a quick time. Moreover, we also propose a novel CP model as an alternative to the IP approach to produce an optimal schedule in a reasonable amount of time.

We explain the working of both approaches with a simple problem instance comprising 5 map and 2 reduce tasks to be scheduled on two machines. Afterward, we propose a novel CP model as an alternative to the IP model to be exactly solved by the CP optimizer of CPLEX studio.

4.1 The heuristic solution

Our heuristic technique employs a greedy approach to schedule map and reduce tasks on suitable energy-efficient machines. The main algorithm is shown in Algorithm 1 where it simply calls two subroutines, namely *map_sched*() and *reduce_sched*() which handle the scheduling decisions of map and reduce tasks, respectively. After completion of these subroutines, if some map or reduce tasks are still unallocated then the main algorithm reports no feasible schedule is possible and returns, otherwise it produces the schedule in the form of decision variables X and Y.

Table 2 The meaning of vectors used in subroutines *map_sched*() and *reduce_sched*()

Vector	Meaning
$\overrightarrow{REQ_j^m}$	jth row of matrix REQ^m which represents the resource request vector of map task m_j
$\overrightarrow{REQ_k^r}$	kth row of matrix REQ^r which represents the resource request vector of reduce task r_k
$\overrightarrow{CAP_i}$	ith row of matrix CAP which represents the total amount of each resource type at machine n_i
$\overrightarrow{RA_i}$	ith row of matrix RA which represents the total amount of each resource type currently allocated at machine n_i

Algorithm 1: Main Algorithm

Input: $\mathcal{M}, \mathcal{R}, \mathcal{N}, \mathcal{A}, E^m, P^m, E^r, P^r, REQ^m, REQ^r, CAP, RA$
Output: X, Y
1 $t = 0$
2 $map\_sched()$;
3 $reduce\_sched()$;
4 **if** $\mathcal{M} \neq \phi \| \mathcal{R} \neq \phi$ **then**
5 **Print** "No feasible schedule"
6 return
7 **else**
8 **Output:** X, Y

The main algorithm requires some input parameters to fix the scheduling decisions. Among these parameters, $\mathcal{M}$, $\mathcal{R}$, $\mathcal{N}$, and $\mathcal{A}$ represent the set of map tasks, reduce tasks, machines, and resources, respectively. The parameters E^m and P^m are two dimensional matrices of size $|\mathcal{N}| \times |\mathcal{M}|$ each and are collection of all energy consumption values e_{ij}^m $(i = 1, \ldots, |\mathcal{N}|; j = 1, \ldots, |\mathcal{M}|)$ and all processing time values p_{ij}^m $(i = 1, \ldots, |\mathcal{N}|; j = 1, \ldots, |\mathcal{M}|)$ of all map tasks, respectively. Particularly, $E^m[i][j] = e_{ij}^m$ and $P^m[i][j] = p_{ij}^m$. Similarly, E^r and P^r are two dimensional matrices of size $|\mathcal{N}| \times |\mathcal{R}|$ each and are collection of all values of e_{ik}^r $(i = 1, \ldots, |\mathcal{N}|; k = 1, \ldots, |\mathcal{R}|)$ and p_{ik}^r $(i = 1, \ldots, |\mathcal{N}|; k = 1, \ldots, |\mathcal{R}|)$, respectively. Particularly, $E^r[i][k] = e_{ik}^r$ and $P^r[i][k] = p_{ik}^r$. Moreover, the parameters REQ^m and REQ^r are 2D matrices of size $|\mathcal{M}| \times |\mathcal{A}|$ and $|\mathcal{R}| \times |\mathcal{A}|$, respectively, where REQ^m is collection of all resource request values req_{jl}^m $(j = 1, \ldots, |\mathcal{M}|; l = 1, \ldots, |\mathcal{A}|)$of map tasks and REQ^r is collection of all resource request values req_{kl}^r $(k = 1, \ldots, |\mathcal{R}|; l = 1, \ldots, |\mathcal{A}|)$ of reduce tasks. Furthermore, the parameters CAP is 2D matrix of size $|\mathcal{N}| \times |\mathcal{A}|$ which stores the total capacity of each resource type at each machine, and particularly $CAP[i][l] = cap_{il}$. Lastly, parameter RA is 2D matrix of size $|\mathcal{N}| \times |\mathcal{A}|$ which stores the total amount of particular resource type currently allocated to various tasks at any machine, and initially at time $t = 0$, $RA[i][l] = 0, (\forall i = 1, \ldots, |\mathcal{N}|; \forall l = 1, \ldots, |\mathcal{A}|)$. Few vectors also have been used in subroutines *map_sched*() and *reduce_sched*() which are mentioned in Table 2 with corresponding meanings.

Algorithm 2: *map_sched*()

```
1  r = 1
2  while M ≠ φ do
3      round_time_r = 0
4      for each unallocated map task m_j ∈ M do
5          Create a priority Q_j of all machines n_i ∈ N
             for map task m_j on the basis of E^m[i][j]
6          while task m_j is unallocated ∧ Q_j is not
             empty do
7              n_i = Q_j.extractMin()
8              if RE⃗Q_j^m + RA⃗_i ≤
                 CA⃗P_i ∧ t + P^m[i][j] ≤ D then
9                  X_ijt = 1
10                 RA⃗_i = RA⃗_i + RE⃗Q_j^m
11                 M = M − {m_j}
12                 if P^m[i][j] ≥ round_time_r then
13                     round_time_r = P^m[i][j]
14     t = t + round_time_r
15     r = r + 1
```

The subroutine *map_sched*() is shown in Algorithm 2 that takes decision regarding which map task is to be assigned to which machine and at what time instance. This assignment takes place in multiple rounds until all map tasks are assigned. During any *r*th round (lines 2–15), first of all, the variable $round\_time_r$, which represents the total duration of *r*th round, is initialized to zero. Afterward, each unallocated map task is picked one at a time in line 4, and for its assignment, a node is greedily searched in while loop of lines 6–13 on the basis of minimum energy consumption. Particularly, for a map task m_j picked in line 4, a priority queue Q_j of all machines is created on the basis of energy consumption e_{ij}^m. After that, a node n_i with minimum energy consumption is extracted in line 7. If the extracted node has the required amount of resources to accommodate the map task and its assignment will not violate the deadline constraint, the task m_j is finally assigned to it at time *t* in line 9 ($X_{ijt} = 1$). Next, few data structures are updated in lines 9 and 10. If the processing time of task m_j is greater than the current value of $round\_time_r$, we update its value as p_{ij}^m. Eventually, $round\_time_r$ is set to the maximum processing time of any map task assigned to any machine in that round. When all unallocated map tasks are attempted to get scheduled in *r*th round, time variable *t* is updated in line 14 and the algorithm enters into the next round by incrementing the (round counter) variable *r* in line 15.

Algorithm 3: *reduce_sched*()

```
1  r = 1
2  while R ≠ φ do
3      round_time_r = 0
4      for each unallocated map task r_k ∈ R do
5          Create a priority Q_k of all machines n_i ∈ N
             for map task r_k on the basis of E^r[i][k]
6          while task r_k is unallocated ∧ Q_k is not
             empty do
7              n_i = Q_k.extractMin()
8              if REQ_k^r + RA_i ≤ CAP_i ∧
                 t + P^r[i][k] ≤ D then
9                  Y_ikt = 1
10                 RA_i = RA_i + REQ_k^r
11                 R = R − {r_k}
12                 if P^r[i][k] ≥ round_time_r then
13                     round_time_r = P^r[i][k]
14     t = t + round_time_r
15     r = r + 1
```

After the subroutine *map_sched*() finishes the scheduling process of map tasks, Algorithm 1 calls the *reduce_sched*() procedure which schedules reduce tasks also in multiple rounds. The procedure *reduce_sched*() is shown in Algorithm 3 with necessary modification as required. The detailed working of this procedure is similar to *map_sched*() procedure and we skip its discussion.

The time complexity of proposed heuristic algorithm can be calculated by combining the complexities of both subroutines called in line 2 and 3 of Algorithm 1. Line 1 and "if" block of lines 4–8 have time complexities of $\mathcal{O}(1)$ each, and do not asymptotically contribute in running time. The complexity of *map_sched*() in worst case is as follows. The priority queue within the subroutine is implemented as a binary heap, and hence line 5 (creating the priority queue), and line 7 (extracting an element) takes $\mathcal{O}(|\mathcal{N}|)$ and $\mathcal{O}(lg|\mathcal{N}|)$ time, respectively. There are 2 while loops: inner (lines 6–13), outer (lines 2–15), and a for loop from lines 4–13. The inner while loop is executed $|\mathcal{N}|$ times at most so its total running time is $\mathcal{O}(|\mathcal{N}|lg|\mathcal{N}|)$. The for loop is executed at most $|\mathcal{N}|$ times, hence its running time is $\mathcal{O}(|\mathcal{M}|$(running time of line 5 and inner while loop)) $= \mathcal{O}(|\mathcal{M}|(|\mathcal{N}| + |\mathcal{N}|lg|\mathcal{N}|)) = \mathcal{O}|\mathcal{M}|(|\mathcal{N}|lg|\mathcal{N}|)$. And finally, the outer while loop is also executed at most $|\mathcal{M}|$ times, hence its complexity, and eventually the *map_sched*() subroutine's complexity is $\mathcal{O}(|\mathcal{M}|^2(|\mathcal{N}|lg|\mathcal{N}|))$. Similarly, the time complexity of *reduce_sched*() is $\mathcal{O}(|\mathcal{R}|^2(|\mathcal{N}|lg|\mathcal{N}|))$, and as a result the main algorithm has $\mathcal{O}(|\mathcal{M}|^2(|\mathcal{N}|lg|\mathcal{N}|) + |\mathcal{R}|^2(|\mathcal{N}|lg|\mathcal{N}|))$ running time in worst case.

Table 3 Characteristics of map and reduce tasks for heuristic algorithm example

Task	Resource requirement	Processing time (s)		Energy consumption (J)	
		n_1	n_2	n_1	n_2
m_1	⟨5MB, 2VC⟩	2	3	4	5
m_2	⟨5MB, 2VC⟩	5	5	3	5
m_3	⟨5MB, 2VC⟩	4	7	4	4
m_4	⟨5MB, 2VC⟩	6	5	2	3
m_5	⟨5MB, 2VC⟩	4	4	3	4
r_1	⟨2MB, 1VC⟩	4	3	2	8
r_2	⟨2MB, 1VC⟩	3	2	6	5

round 1: $\mathcal{M} = \{m_1, m_2, m_3, m_4, m_5\}$

Task	Priority queue	Assignment	Remaining capacity
m_1	Q_1={n_1, n_2}	$m_1 \rightarrow n_1$	⟨5, 1⟩, ⟨15, 4⟩
m_2	Q_2={n_1, n_2}	$m_2 \rightarrow n_2$	⟨5, 1⟩, ⟨10, 2⟩
m_3	Q_3={n_1, n_2}	$m_3 \rightarrow n_2$	⟨5, 1⟩, ⟨5, 0⟩
m_4	Q_4={n_1, n_2}		⟨5, 1⟩, ⟨5, 0⟩
m_5	Q_5={n_1, n_2}		⟨5, 1⟩, ⟨5, 0⟩
$round\_time_1 = 7s$			
$t = 7s$			

round 2: $\mathcal{M} = \{m_4, m_5\}$

Task	Priority queue	Assignment	Remaining capacity
m_4	Q_4={n_1, n_2}	$m_4 \rightarrow n_1$	⟨5, 1⟩, ⟨15, 4⟩
m_5	Q_5={n_1, n_2}	$m_5 \rightarrow n_2$	⟨5, 1⟩, ⟨10, 2⟩
$round\_time_2 = 6s$			
$t = 13s$			

Fig. 2 Working of *map_sched*() subroutine

4.1.1 A numerical example of heuristic algorithm

We take an example to show the working of the proposed heuristic algorithm. In the example, an MR job, comprising 5 map tasks and 2 reduce tasks, is to be scheduled on two machines with characteristics shown in Table 3. The job is to be completed within a deadline $D = 17s$. The machine n_1 has a total 10 MB of RAM and 3 VCs denoted as ⟨10MB, 3VC⟩, whereas machine n_2 has 15 MB of RAM and 4 VCs, i.e., ⟨10MB, 3VC⟩. The algorithm starts by initializing time variable $t = 0s$. It then simply calls *map_sched*() and *reduce_sched*() subroutines one after another. The working of subroutine *map_sched*() is shown in Fig. 2 where it assigns map tasks to machines in two rounds. The first column in Fig. 2 lists the map tasks attempted for allocation in that order during each round. The second column shows the priority queue of machines created on the basis of energy consumption of a particular selected map task in line 4. The third column shows the final assignment of map task to a machine (if any), and the last column shows the remaining resource capacity of machines after the assignment.

At the start of first round of *map_sched*() when time $t = 0s$, the set $\mathcal{M}$ comprises all map tasks, i.e., all map tasks are unallocated. First, the map task m_1 is picked from set $\mathcal{M}$ and a priority queue $Q_1 = \{n_1, n_2\}$ is created based on the energy consumption of map task m_1 on both machines n_1 and n_2. Next,

round 1: $\mathcal{R} = \{r_1, r_2\}$

Task	Priority queue	Assignment	Remaining capacity
r_1	Q_1=$\{n_1, n_2\}$	$r_1 \rightarrow n_1$	$\langle 8, 2\rangle$, $\langle 15, 4\rangle$
r_2	Q_2=$\{n_2, n_1\}$	$r_2 \rightarrow n_2$	$\langle 8, 2\rangle$, $\langle 13, 3\rangle$
$round\_time_1 = 4s$			
$t = 17s$			

Fig. 3 working of *reduce_sched*() subroutine

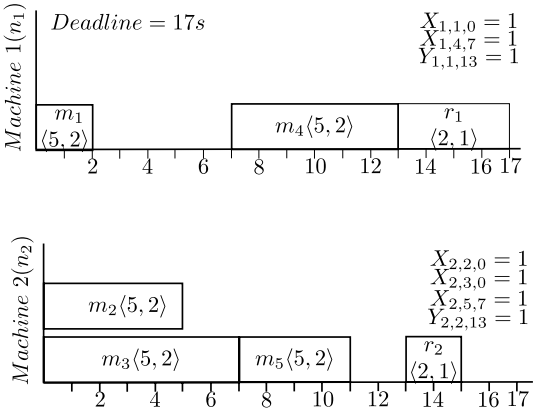


Fig. 4 An example schedule for deadline $D = 17s$ generated by heuristic algorithm

machine n_1 is extracted from Q_1 for allocation and found to be able to accommodate task m_1. Moreover, its processing will exceed the deadline $D = 17s$ once assignment to machine n_1 at $t = 0s$. Hence, it is finally allocated to machine n_1 ($X_{1,1,0} = 1$). after this assignment remaining capacity of n_1 and n_2 are $\langle$5MB, 1VC$\rangle$ and $\langle$15MB, 4VC$\rangle$, respectively. All this working has been shown in first row of the table. Next, task m_2 is picked for allocation and again a priority queue $Q_2 = \{n_1, n_2\}$ is created. However, the first node n_1 of queue Q_2 is unable to accommodate task m_2. Hence, second node n_2 is extracted and found to have sufficient resources required by task m_2 which is then scheduled on it ($X_{2,2,0} = 1$). The remaining capacity of n_1 and n_2 are $\langle$5MB, 1VC$\rangle$ and $\langle$10MB, 2VC$\rangle$, respectively, after this assignment. Similarly, task m_3 is scheduled on node n_2($X_{2,3,0} = 1$). Further, tasks m_4 and m_5 are picked for allocation but do not get scheduled on any node due to resource unavailability. After the end of first round, $round\_time_1$ is calculated as $7s$ and time t also is set as $7s$.

In the starting of second round of *map_sched*() subroutine, the set $\mathcal{M}$ has two map tasks m_4 and m_5 which are picked one by one for allocation in that order. It is clear from Fig. 2 that task m_4 and m_5 are scheduled on node n_1 ($X_{1,4,7} = 1$) and n_2 ($X_{2,5,7} = 1$), respectively. After the end of second round, $round\_time_2$ is calculated as $6s$ and time t is set as $7 + 6 = 13s$. After the completion of *map_sched*() subroutine, *reduce_sched*() starts and completes within one round as shown in Fig. 3 which shows that reduce tasks r_1 and r_2 are schedules on

nodes n_1 ($Y_{1,1,13} = 1$) and n_2 ($Y_{2,2,13} = 1$), respectively. At the end of this round, $round\_time_2$ is calculated as $4s$ and time t is set as $13 + 4 = 17s$. Figure 4 shows the generated schedule.

4.2 Constraint programming approach

The constraint programming (CP) technology is principally based on computer science fundamentals and takes its origin in graph theory (GT), artificial intelligence and logic programming endeavors of the 1980s. The recent improvements in the development of a tunable and robust black-box search for constraint programming engines have turned this into a powerful and easy-to-use optimization technology. A CP problem is usually represented as a constraint satisfaction problem (CSP) which is defined as a triple (X, D, C) where $X = \{x_1, x_2, ...x_n\}$ is a set of n variables, $D = \{D_1, D_2, ...D_n\}$ is a set of n domains of respective variables, and $C = \{c_1, c_2, ...c_m\}$ is a set of m constraints defined over different variables.

The domain $D_i \in D$ of a variable x_i is a set of all possible values that can be assigned to the variable. Every constraint $c_j \in C$ is represented as pair (T_j, R_j) where T_j is subset X and has k variables, and R_j is a k-ary relation on the corresponding subset of k domains. A solution to a CSP is a complete instantiation of the variables in X satisfying all the constraints in C. The instantiation of a variable refers to the assignment of a value from its domain, and it is considered complete if all variables are assigned a value only once.

The CP technique considers any formulated combinatorial problem as a graph search problem and extracts its power from the techniques of artificial intelligence, algorithms, graph theory, operations research, etc. The search space of a CP problem comprises all combinations of the values in the domains of the decision variables. The CP optimizer engine explores the search space to find a solution. One way to find a solution would be to explicitly try each combination of values until a solution is found. This approach is sometimes called the exhaustive search technique which is obviously time-consuming and inefficient even for a simple problem. However, problem-specific searching techniques and constraint propagation can dramatically speed-up the solution process [21].

The CP problem has the same conceptual elements in its definition as MP problems, e.g., both have a set of decision variables, one or more objective functions, and a set of constraints, however, there are some basic and important differences between them. For example, MP models support both discrete and continuous decision variables, whereas CP models have only discrete decision variables (integer or Boolean). Another important difference is the solution approach to both models. The MP optimizer relies on numerical linear algebra for finding the solutions while the CP optimizer uses logic programming and graph search mechanisms. The differences and similarities of both approaches are summarized in Table 4.

In the field of scheduling, constraint programming has proven very efficient. Smith et al. [36], Darbi-Dowman et al. [37], and Lustig and Puget [38] compare IP and CP approaches in various scheduling problems. These studies imply that IP techniques seem to be better for problems in which LP relaxations provide strong bounds

Table 4 Characteristics of mathematical and constraint programming

Feature	CP	MP
Modeling limitations	Discrete problems	Discrete as well as continuous problems
Theoretical grounds	GT and algorithmic	Algebra
Specialized constraints	Yes	No
Logical constraints	Yes	Yes
Optimality proof	Yes	Yes
Relaxation	No	Yes
GAP measure	No	Yes

for the objective function, whereas CP is better than IP techniques in sequencing, scheduling applications, and strict feasibility problems.

The CP model for any combinatorial optimization problem, in contrast to the IP model, is highly dependent on the CP package used to model the problem because of the differences in constructs available in various modeling languages. In this paper, the optimization programming language (OPL) framework bundled with CPLEX Studio v12.8 has been used as the modeling language to prepare the CP model. The details of this modeling language are beyond the scope of this paper, however, the structures that have been used to model the considered scheduling problem has been described next.

4.2.1 The optimization programming language

The OPL framework [39] has special constructs to model scheduling problems that often involve a set of activities (e.g., tasks) that need to be completed using a set of resources, e.g., machines, operators, tools, etc. In any OPL scheduling model, The tasks are represented by interval variables, which are defined by `interval` keyword. The variable represents an interval of time during which a task is executed or any activity happens. It is characterized by an unknown *start_time* and an *end_time*, a known *size* (or duration). The position (i.e., *start_time*) of an interval variable has to be fixed during the solution of the scheduling problem. The difference of its *end_time* minus and *start_time* must be equal to its *size*.

An important additional feature of interval variables is the fact that they can be optional (declared by the `optional` keyword); that is, those variables may not be included in the feasible schedule and can be left unperformed. An OPL Boolean function `presenceOf` is used to represent the presence of an optional interval. If the (interval) variable is included in the generated schedule, the function `presenceOf` returns the Boolean value 1, otherwise 0.

The `alternative` function creates an alternative constraint between interval variable *i* and the set of interval variables *i_array* and has the following syntax: `alternative(interval i, interval i_array, int cardinality = 1)`. The default value of cardinality is always considered as 1 if it is not present. In such cases, only one of the intervals from *i_array* will be selected by the

alternative constraint and the *start_time* and *end_time* of interval *i* will be same as ones of the selected interval.

The `pulse` function has the following syntax: `pulse(interval i, int h)` which returns an elementary cumulative function expression that is equal to a value `h` everywhere between the *start_time* and *end_time* of an interval variable `i`. The function is equal to zero outside of the interval. This function is used in our proposed formulation to express resource constraint.

Two more OPL functions `endOf` and `endBeforeStart` are also used in our CP formulation where `endOf(interval i)` function returns the end_time of the interval variable `i` if it is present. The function `endBeforeStart` maintains a minimum delay between two interval variables and has the following syntax: `endBeforeStart(interval predecessor, interval successor, int minDelay = 0)`, where the default value for minDelay is zero.. If both interval variables predecessor and successor are present, the successor cannot start before `endOf(predecessor)` + minDelay. If the predecessor or successor is absent, then the constraint is automatically satisfied. The functions `endOf` and `endBeforeStart` are used in proposed model for deadline and temporal dependency constraints, respectively.

4.2.2 The proposed CP model

In this section, we present the proposed CP model for the DMRSYE problem. The model uses four different arrays of interval variables, namely **Map**, **Red**, **Machine_Map**, and **Machine_Red**. Among these, **Map** and **Red** are 1D arrays of $|\mathcal{M}|$ OPL activities $map_j (j = 1, \ldots, |\mathcal{M}|)$, and $|\mathcal{R}|$ OPL activities $red_k (k = 1, \ldots, |\mathcal{R}|)$, respectively. The OPL activity $map_j \in$ **Map** represents jth map tasks, whereas $red_k \in$ **Red** represents and kth reduce tasks.

Further, **Machine_Map** (size $|\mathcal{N}| \times |\mathcal{M}|$) and **Machine_Red** (size $|\mathcal{N}| \times |\mathcal{R}|$) are 2D arrays of *optional* interval variables $machine\_map_{ij} (i = 1, \ldots, |\mathcal{N}|; j = 1, \ldots, |\mathcal{M}|)$ and $machine\_red_{ik} (i = 1, \ldots, |\mathcal{N}|; k = 1, \ldots, |\mathcal{R}|)$, respectively. The ith row of both **Machine_Map** and **Machine_Red** matrices represents the collection of map and reduce tasks, respectively, which are assigned to ith machine.

And when OPL Boolean function `presenceOf` is applied on these variables, the return values (1 or 0) signify the assignment of tasks on machines. Particularly, if `presenceOf`($machine\_map_{ij}$) returns the value 1, it signifies that map task m_j is assigned to machine n_i and 0 otherwise. Similarly, return value 1 of `presenceOf`($machine\_red_{ik}$) signifies that reduce task r_k is assigned to machine n_i and 0 otherwise.

Using the discussed OPL constructs, the proposed CP model is for the DMRSYE problem is as follows.

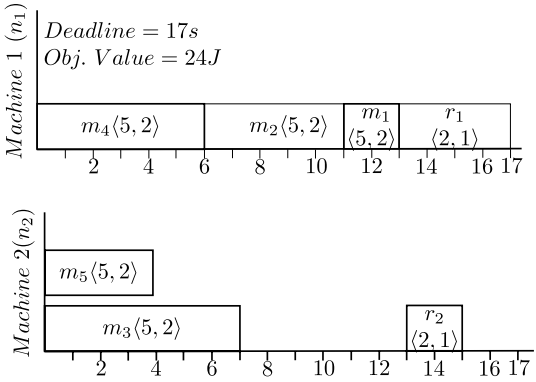


Fig. 5 An example CP schedule for deadline $D = 17s$

$$\min \sum_{j=1}^{|\mathcal{M}|} \sum_{i=1}^{|\mathcal{N}|} e_{ij}^{m} * \text{presenceOf}(machine\_map_{ij}) + \sum_{k=1}^{|\mathcal{R}|} \sum_{i=1}^{|\mathcal{N}|} e_{ik}^{r} * \text{presenceOf}(machine\_red_{ik})$$

subject to:

$$\texttt{alternative}(map_j, \textbf{\textit{all}}(i \in \{1, \ldots, |\mathcal{N}|\}) machine\_map_{ij}), \quad (j = 1, \ldots, |\mathcal{M}|) \tag{8}$$

$$\texttt{alternative}(red_k, \textbf{\textit{all}}(i \in \{1, \ldots, |\mathcal{N}|\}) machine\_red_{ik}), \quad (k = 1, \ldots, |\mathcal{R}|) \tag{9}$$

$$\texttt{endBeforeStart}(map_j, red_k), \quad (j = 1, \ldots, |\mathcal{M}|; k = 1, \ldots, |\mathcal{R}|) \tag{10}$$

$$\texttt{endOf}(red_k) \leq D, \quad (k = 1, \ldots, |\mathcal{R}|) \tag{11}$$

$$\sum_{j=1}^{|\mathcal{M}|} \texttt{pulse}(machine\_map_{ij}, RR_{jl}^{m}) + \sum_{k=1}^{|\mathcal{R}|} \texttt{pulse}(machine\_red_{ik}, RR_{kl}^{r}) \leq RC_{il}, \quad (i = 1, \ldots, |\mathcal{N}|; l = 1, \ldots, |\mathcal{A}|). \tag{12}$$

In the above CP model, the objective function minimizes the total energy consumption. Constraints 8 and 9 collectively ensure that all map and reduce tasks are scheduled to only one machine once. Constraint 10 establishes the temporal dependency between map tasks and reduce tasks. Constraint 11 ensures that all tasks end before the user-specified deadline. Lastly, constraint 12 (pulse constraint) ensures that total resources consumed by all tasks scheduled on a machine at a particular time do not exceed its resource capacity.

If this CP model is used to solve the problem instance of Sect. 4.1.1 with the same characteristics as shown in Table 3 for a deadline $D = 17s$, the CP optimizer of CPLEX studio generates an optimal schedule as shown in Fig. 5 with objective value $24J$. The value of each decision variable (in this case interval variable) are represented as a tuple (*start_time*, *end_time*, *duration*). The optimizer outputs the following result:

Map	$[(11,\ 13,\ 2), (6,\ 11,\ 5), (0,\ 7,\ 7,)\ (0,\ 6,\ 6), (0,\ 4,\ 4)]$
Red	$[(13,\ 17,\ 4), (13,\ 15,\ 2)]$
Machine_Map	$[[map_1(11,\ 13,\ 2),\ map_2(6,\ 11,\ 5),\ ABSENT,\ map_4(0,\ 6,\ 6),\ ABSENT]\ [ABSENT,\ ABSENT,\ map_3(0,\ 7,\ 7),\ ABSENT,\ map_5(0,\ 4,\ 4)]]$
Machine_Red	$[[reduce_1(13,\ 17,\ 4), ABSENT]\ ABSENT, reduce_2(13,\ 15,\ 2)]]$.

If the deadline parameter is set as $D = 12s$ for the same problem instance, the CP optimizer still produces the optimal schedule with an objective value of $27J$. However, the heuristic algorithm fails to produce even a feasible schedule for $D = 12s$. We notice that both approaches have a lower bound on the deadline parameter which they can meet. Moreover, the CP approach can satisfy tighter deadline constraints.

5 Experiments: workloads, cluster setup and profiling

We perform two sets of experiments to compare the proposed heuristic and default search algorithm of CP optimizer which solves the proposed CP model. From now onward, we will call this default search algorithm as CP-SA. In the first set of experiments (called as performance analysis), we take 8 different MR jobs from HiBench and PUMA benchmark suite and compare heuristic and CP-SA techniques on the basis of following performance metrics:

- Total energy consumption (TEC) in joule (J): This metric is defined as the total energy consumed within the YARN cluster during the job execution while following the schedule generated by respective algorithm.
- Execution time (ET) or schedule generation time (SGT) in seconds (s): It is defined as the time taken by algorithms to generate the (static) schedule. It is simply the execution time of algorithms.
- Tightest satisfiable deadline (D_{TS}) in seconds (s): It is defined as the lowest value of user-specified deadline which can be achieved by both approaches.

In the second set of experiments, we perform the sensitivity analysis of these performance metrics on the workload size (total number of map and reduce tasks), and again on deadline, separately. For that purpose, we choose the WordCount benchmark job.

The selected MR jobs from HiBench [40] and PUMA [41] benchmark suite for the first set of experiments are listed in Table 5 along with its type and category. HiBench

Table 5 Selected HiBench and PUMA workload

Workload	Type	Category
Self-Join	IO bound	PUMA
Adjacency-List	Mix bound	PUMA
WordCount	CPU bound	HiBench (Micro)
TeraSort	IO bound	HiBench (Micro)
Histogram-Movies	IO bound	PUMA
K-Means clustering	Mix bound	HiBench (ML)
PageRank	CPU bound	HiBench (Web search)
Inverted-Index	IO bound	PUMA

is a big data benchmark suite that helps evaluate different big data frameworks in terms of speed, throughput, and system resource utilization. There are a total of 19 workloads in HiBench. The workloads are divided into 6 categories: micro, machine learning (ML), SQL, graph, web search, and streaming. PUMA is a newer benchmark suite comprising 13 workloads among which few are common to HiBench as well.

All experiments have been performed through simulations on a single machine using the estimated value of processing time (p^m_{ij}/p^r_{ik}) and energy consumption (e^m_{ij}/e^r_{ik}) of tasks. This estimation is performed with the help of profiled data. As both approaches are static and non-distributed in nature, simulative experiments on a single machine are sufficient for a comparative study. The profiling of processing time and the energy consumption is done by executing a single benchmark job several times on a real YARN cluster which is described next.

5.1 YARN cluster setup

A five node Hadoop YARN cluster has been set up to profile processing time and energy consumption of tasks. The cluster is composed of five nodes where one node acts as a master and the remaining four nodes as slaves. The master node has a 10 core Intel Xeon W-2155 processor, 64 GB RAM, and 2TB hard disk. One of the slave nodes has the same configuration as the master node, two slave nodes have 6 cores Intel Xeon E5645 processor, 8 GB RAM, and 1TB hard disk each, and lastly, one slave node has 2 cores Intel Core i5-7200U processor, 12 GB RAM and 1TB hard disk. We use Hadoop 2.7.2 framework with default HDFS block size of 128 MB, inbuilt FAIR scheduler, and file replication factor as 3. All nodes are connected through a 1Gbps network switch. The cluster configuration is summarized in Table 6.

5.2 Profiling and estimation of energy consumption and processing time of tasks

During the profiling, we execute each selected benchmark MR job several times with random file sizes that generate a different number of map and reduce tasks each

Table 6 Cluster configuration

Machine	Processor	#Physical cores	RAM (GB)	Disk (TB)
Master	Intel Xeon W-2155	10	64	2
Slave-1	Intel Xeon W-2155	10	64	2
Slave-2	Intel Xeon E5645	6	8	1
Slave-3	Intel Xeon E5645	6	8	1
Slave-4	Intel Core i5-7200U	2	12	1

time. At a particular run, if map task m_j is scheduled on the machine n_i, then its energy consumption e_{ij}^m is calculated using the following equation:

$$e_{ij}^m = P_i^{cpu} \times p_{ij}^m + P_i^{mem} \times p_{ij}^m + P_i^{disk} \times D_{ij}^m + P_i^{nic} \times N_{ij}^m$$

where P_i^{cpu} and P_i^{mem} are CPU and memory power consumption rate, respectively, of machine n_i in watt. P_i^{disk} and P_i^{nic} are disk power consumption per byte read/written and NIC power consumption per byte sent/received, respectively, of machine n_i in joule. Moreover, p_{ij}^m has usual meaning, i.e., processing time of map task m_j on machine n_i and can easily be noted down from the Hadoop system logs. Furthermore, The values D_{ij}^m and N_{ij}^m are total disk IO in bytes and data shuffled in bytes, respectively, for map task m_j on machine n_i. The energy consumption e_{ik}^r of reduce task r_k is also evaluated using the similar equation if scheduled on machine n_i.

We take the average energy consumption of all map tasks scheduled on machine n_i during a particular run and denote it as $\bar{e}_i^m$. Similarly, the values of $\bar{e}_i^r$, $\bar{p}_i^m$, and $\bar{p}_i^r$ are also calculated where the expressions $\bar{e}_i^r$, $\bar{p}_i^m$, and $\bar{p}_i^r$ represents average energy consumption of all reduce tasks, average processing time of all map tasks, and average processing time of all reduce tasks scheduled on machine n_i at a particular run, respectively. As a single job is executed several times, we have multiple values of $\bar{e}_i^m$, $\bar{e}_i^r$, $\bar{p}_i^m$, and $\bar{p}_i^r$, each for a particular run.

During the actual performance evaluation on a single machine, the value required parameter is estimated as follows. The processing time and energy consumption of any map task at a particular machine n_i are taken randomly between the minimum and maximum of all $\bar{e}_i^m$ and $\bar{p}_i^m$ values, respectively. Similarly, processing time and energy consumption of reduce tasks are taken between the minimum and maximum of all $\bar{e}_i^r$ and $\bar{p}_i^r$ values, respectively. Further, the value of the user-specified deadline (D) has been set according to Eq. 13 and denoted as D_S so that every time we get a feasible schedule.

$$D = D_S = \frac{\sum_{j=1}^{|\mathcal{M}|} \overline{T}_j^m + \sum_{k=1}^{|\mathcal{R}|} \overline{T}_k^r}{n}, \tag{13}$$

where $\overline{T}_j^m = \dfrac{\sum_{i=1}^{n} p_{ij}^m}{n}$ and $\overline{T}_k^r = \dfrac{\sum_{i=1}^{n} p_{ik}^r}{n}$ are average processing time of any map task m_j and reduce task r_k.

6 Experiments: results and discussion

In this section, we discuss the results obtained from both sets of experiments. First of all, Sect. 6.1 discusses the results of the first set of experiments (i.e., performance analysis) where we compare both algorithms on the basis of three performance metrics for each selected benchmarks as explained in Sect. 5. After this, Sect. 6.2 discusses the results of sensitivity analysis experiments.

6.1 Performance analysis

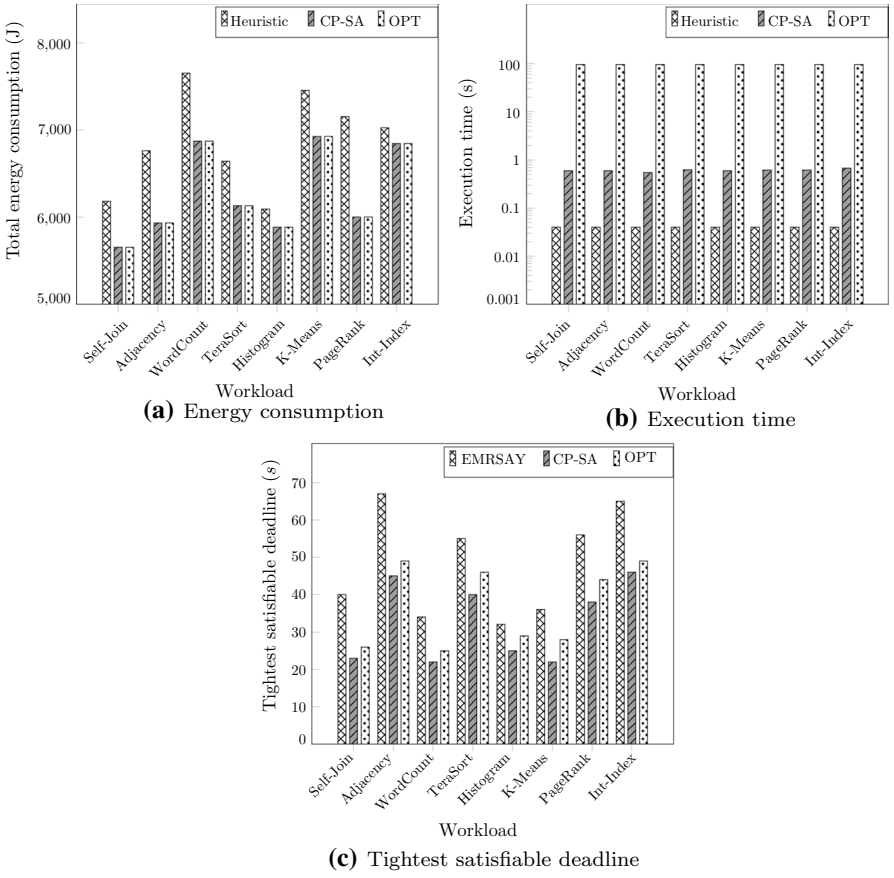
The performance analysis experiment is repeated twice, first for small-scale workload denoted as (64M, 32R), comprising 64 map and 32 reduce tasks, and then for large-scale workload denoted as (256M, 256R) which has 256 map and reduce tasks each.

6.1.1 Small workloads

In small-scale experiments, besides heuristic and CP-SA algorithms, the results of CPLEX solver's inbuilt IP-based branch-and-cut (B&C) technique are also included which optimally solves the initial IP formulation of the DMRSYE problem. We refer to the inbuilt B&C algorithm as OPT. The objective is here to compare our proposed heuristic and novel CP models with that of IP formulation.

Figure 6a shows the total energy consumption of heuristic, CP-SA, and OPT algorithms for small workloads. CP-SA and OPT always achieve optimal results as they are intended to do so. On the other hand, the heuristic algorithm consumes at most 12% and at least 6% with an average of 7% more energy than optimal solutions. Figure 6b shows the execution time (ET) of all algorithms for each selected benchmarks. It is observed that the execution time of heuristic, CP-SA, and OPT algorithm is $\approx 0.04s$, $\approx 0.6s$, and $\approx 96.45s$, respectively. Therefore, we conclude that CP-SA takes 7.14% more time than the heuristic algorithm but always produces an optimal schedule. when compared to the OPT algorithm, CP-SA takes far less time. Moreover, it can also be observed that individually, ET of the heuristic, CP-SA, and OPT algorithms are approximately the same for each benchmark job. It means, for a fixed number of tasks and machines (in this case, 96 tasks and 5 machines), the execution time of algorithms does not depend on energy consumption, processing time, and resource request variations of tasks among different benchmarks.

Both heuristic and CP-SA have a lower bound for any user-specified deadline to be met which we define as the tightest satisfiable deadline (D_{TS}). Obviously, it is better to have a smaller lower bound, which means the algorithm can schedule a job under tighter deadline conditions. The heuristic, CP-SA, and OPT can be run several times in a binary search manner as in [9] in order to find the tightest deadline that each algorithm can meet. We perform experiments to evaluate the D_{TS} for all three approaches under each selected benchmarks. The result in Fig. 6c shows



(a) Energy consumption

(b) Execution time

(c) Tightest satisfiable deadline

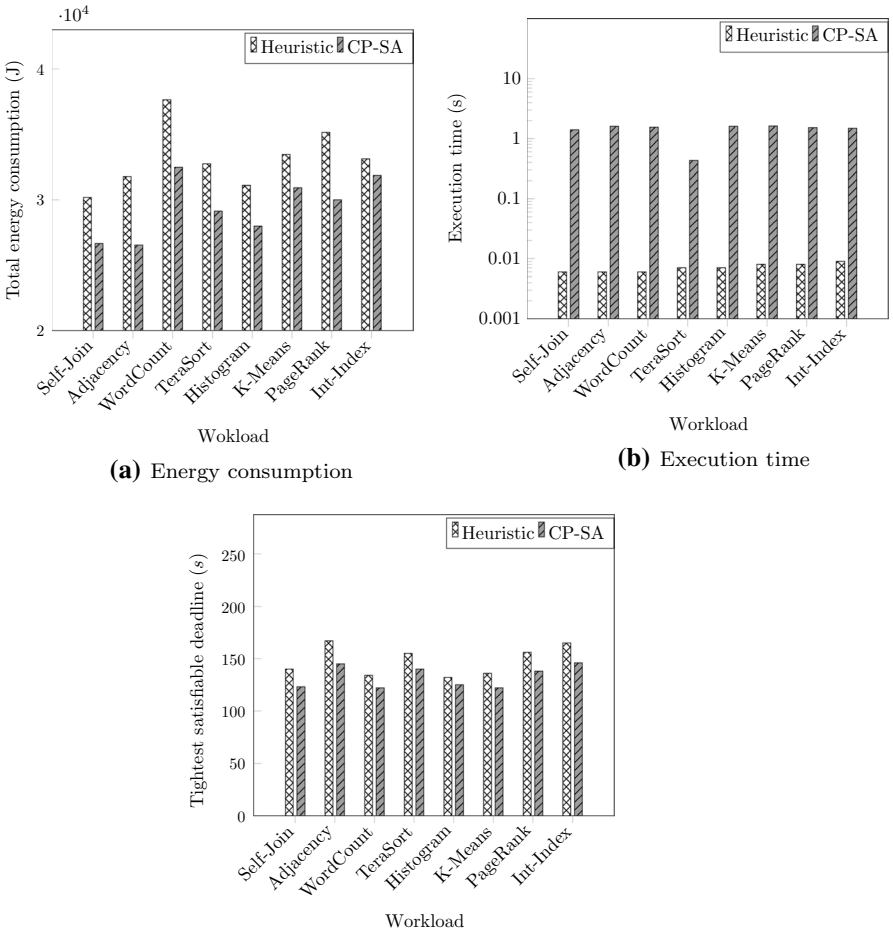
Fig. 6 Performance on small workloads

that CP-SA achieves tighter deadlines than heuristic and OPT algorithm for every benchmark. Particularly, it achieves 30% tighter deadlines than the heuristic, and 5% tighter deadlines than the OPT algorithm on average.

6.1.2 Large workloads

In large-scale experiments, OPT takes an enormous amount of time to solve the IP formulation. Even for some benchmarks, we could not get the result after several hours of an execution, hence we exclude it from our results.

Figure 7 shows the results for large-scale workloads. We observe that the optimality gap in the case of total energy consumption increases between heuristic and CP-SA. Now, the heuristic algorithm consumes at most 25% and at least 15% with an average of 23% more energy than optimal solutions as shown in Fig. 7a. Execution times for large workloads are shown in Fig. 7b which clearly indicates that the heuristic approach takes $\approx 0.4s$ and CP-SA takes $\approx 1.14s$. It means ET of



(a) Energy consumption

(b) Execution time

(c) Tightest satisfiable deadline

Fig. 7 Performance on large workloads

both algorithms increases as we increase the workload size. However, heuristic takes slightly less time than CS-PA at the cost of producing sub-optimal schedules. As far as the tightest satisfiable deadline D_{TS} is concerned for both approaches, Fig. 7c shows that CP-SA achieves tighter deadlines for every benchmarks.

6.2 Sensitivity analysis

In this section, first we analyze the sensitivity of TEC, ET, and D_{TS} metrics on workload size (total number of tasks) and then the sensitivity of TEC, ET on deadline under both approaches. When analyzing the sensitivity on workload size, we take workloads as shown in Table 7. Whereas, when the sensitivity analysis on user deadline is performed, the number of map and reduce tasks are fixed as (256M, 256R),

Table 7 WordCount workloads for sensitivity analysis experiment

Workload	#Total tasks	#Map tasks	#Reduce tasks
(64M, 32R)	64	32	96
(64M, 64R)	64	64	128
(128M, 64R)	128	64	192
(128M, 128R)	128	128	256
(256M, 128R)	256	128	384
(256M, 256R)	256	256	512
(512M, 256R)	512	256	768
(512M, 512R)	512	512	1024

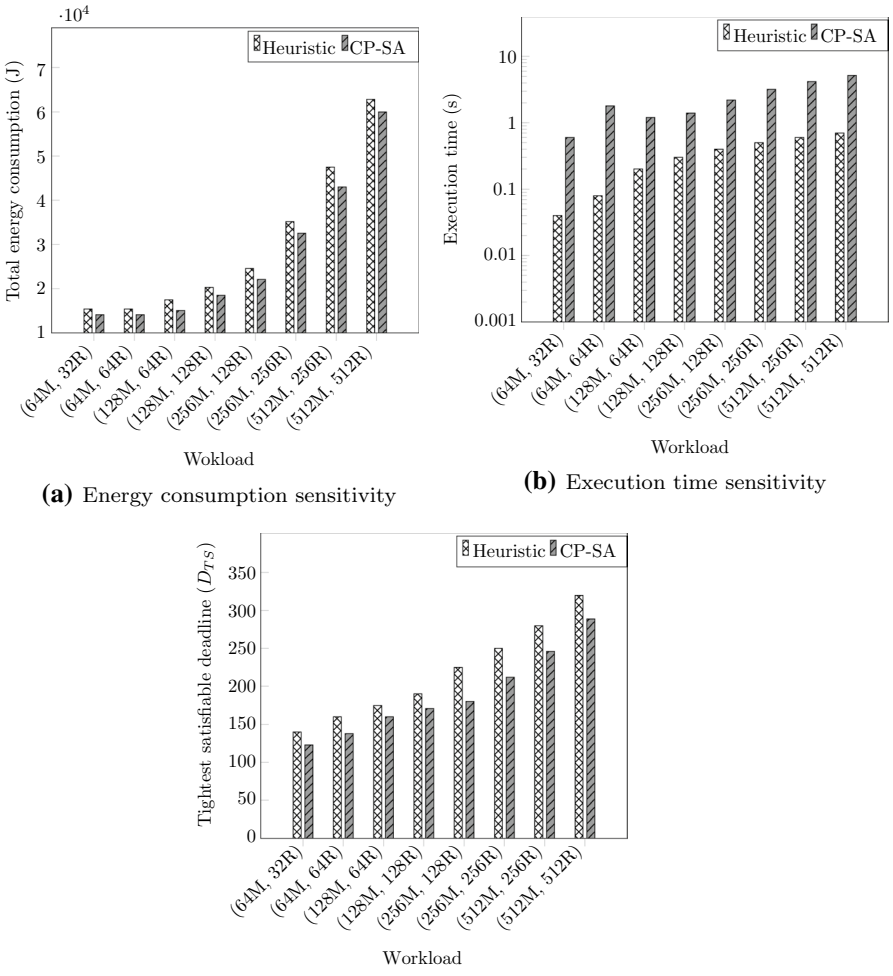
and the deadlines are varied from $D_S - 40$ to $D_S + 60$ where D_S represent the satisfiable deadline calculated according to Eq. 13.

6.2.1 Sensitivity on workload size

Figure 8a and b shows the sensitivity of TEC and ET, respectively, on the total number of tasks. Particularly, Fig. 8a shows that energy consumption increases as the number of map and reduce tasks are increased. For example, the total energy consumption of Heuristic and CP-SA algorithms for the workload (128M, 128R) is 22896 and 20319J, respectively, while the total energy consumption for the workload (256M, 128R) are 31358 and 52552J. It is to be noted that for all workloads, energy consumption in the heuristic approach is always greater than but very close to CP-SA. Figure 8b shows that execution time also increases as the number of map and reduce tasks are increased for both proposed approaches. Finally, Fig. 8c shows the sensitive analysis of D_{TS} on the total number of tasks. It has been found that as we increase the number of total tasks, D_{TS} for both approaches also increases. For example, at workload size of (128M, 128R), D_{TS} for heuristic and CP-SA are 240 and 200s, respectively, whereas at (256M, 128R) D_{TS} are 280 and 230 s.

6.2.2 Sensitivity on deadlines

In modern data centers, the deadline parameter, before which the job is to be completed, is supplied by the user when he submits the job. To investigate the sensitivity of TEC and ET on different deadlines under both algorithms, two experiments were performed with the results shown in Fig. 9a and b. And particularly, as shown in Fig. 9a, the energy consumption of tasks in the heuristic algorithm does not depend on the deadline parameter and remains constant. On the other hand, CP-SA produces optimal schedule with large objective value at tighter deadlines and optimizes it more as the deadlines are relaxed. For example, when deadline is $D_S - 20$, the energy consumption is 34743J, whereas it is 33517J when deadline is set to D_S. Further, the sensitivity of ET on deadlines is shown in Fig. 9b. It shows that ET of heuristic and CS-PA algorithms remain constant for all deadline parameters and do not depend on it.



(a) Energy consumption sensitivity

(b) Execution time sensitivity

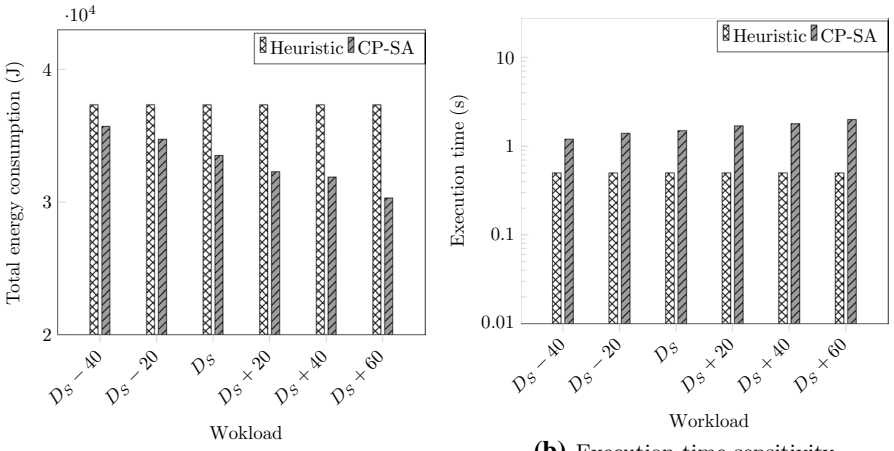
(c) Tightest satisfiable deadline sensitivity

Fig. 8 Sensitivity analysis on workload size

7 Conclusion

In the era of green computing, the reduction of energy consumption in modern big data processing frameworks like Hadoop YARN is a challenging problem. This can be addressed through efficient scheduling of map and reduce tasks. The YARN has a different resource management technique than slot-based Hadoop and has introduced the concept of containers, which consequently influences the MR scheduling model. In this paper, we consider the deadline-constraint MR scheduling problem in Hadoop YARN to minimize energy consumption.

Two different solution approaches: a heuristic algorithm and a novel CP model to be solved by IBM CPLEX studio, have been proposed to solve the considered



(a) Energy consumption sensitivity

(b) Execution time sensitivity

Fig. 9 Sensitivity analysis on deadline

problem. Both solution approaches have been compared on the basis of various performance parameters including TEC, ET, and the tightest satisfiable deadline. We take a wide variety of MR jobs from HiBench and PUMA benchmark suite in our experiments. Although the CP approach takes some more time than the heuristic technique to produce a schedule, it always gives optimal results. Hence, in situations where we can tolerate some delay of a few seconds, the CP technique turns out to be a better option. Besides this, the CP solution produces optimal schedules under tighter deadline constraints. In the future, we aim to design some custom search procedures in place of the default search of the CPLEX CP engine which can further reduce the schedule generation time.

Acknowledgements This work is financially supported by Ministry of Electronics and Information Technology, Government of India, under the Visvesvaraya PhD scheme, award no. VISPHD-MEITY-2689.

References

1. Apache hadoop. https://hadoop.apache.org/. Accessed: 2019-06-10
2. White T (2009) Hadoop: The definitive guide. O'Reilly Media, Inc., USA
3. Tiwari N, Sarkar S, Bellur U, Indrawan M (2015) Classification framework of mapreduce scheduling algorithms. ACM Comput Surv (CSUR) 47(3):49
4. Pandey V, Saini P (2018) How heterogeneity affects the design of hadoop mapreduce schedulers: A state-of-the-art survey and challenges. Big Data 6(2):72–95
5. Moseley B, Dasgupta A, Kumar R, Sarlós T (2011) On scheduling in map-reduce and flow-shops. In: Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 289–298. ACM
6. Verma A, Cherkasova L, Campbell RH (2013) Orchestrating an ensemble of MapReduce jobs for minimizing their makespan. IEEE Trans Dependable Secure Comput 10(5):314–327
7. Tian W, Li G, Yang W, Buyya R (2016) Hscheduler: an optimal approach to minimize the makespan of multiple mapreduce jobs. J Supercomput 72(6):2376–2393

8. Mashayekhy L, Nejad MM, Grosu D, Zhang Q, Shi W (2015) Energy-aware scheduling of mapreduce jobs for big data applications. IEEE Trans Parallel Distrib Syst 26(10):2720–2733
9. Yousefi MHN, Goudarzi M (2018) A task-based greedy scheduling algorithm for minimizing energy of mapreduce jobs. J Grid Comput 16(4):535–551
10. Fischer MJ, Su X, Yin Y (2010) Assigning tasks for efficiency in hadoop. In: Proceedings of the Twenty-Second Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 30–39. ACM
11. Aggarwal V, Xu M, Lan T, Subramaniam S (2017) On the optimality of scheduling dependent mapreduce tasks on heterogeneous machines. arXiv preprint arXiv:1711.09964
12. Zhu Y, Jiang Y, Wu W, Ding L, Teredesai A, Li D, Lee W (2014) Minimizing makespan and total completion time in mapreduce-like systems. In: IEEE INFOCOM 2014-IEEE Conference on Computer Communications, pp. 2166–2174. IEEE
13. Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, et al (2013) Apache hadoop yarn: Yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing, p. 5. ACM
14. Fotakis D, Milis I, Papadigenopoulos O, Vassalos V, Zois G (2016) Scheduling mapreduce jobs under multi-round precedences. In: European Conference on Parallel Processing, pp. 209–222. Springer
15. Chen F, Kodialam M, Lakshman T (2012) Joint scheduling of processing and shuffle phases in mapreduce systems. In: 2012 Proceedings IEEE INFOCOM, pp. 1143–1151. IEEE
16. Zheng Y, Shroff NB, Sinha P (2013) A new analytical technique for designing provably efficient mapreduce schedulers. In: 2013 Proceedings IEEE INFOCOM, pp. 1600–1608. IEEE
17. Fotakis D, Milis I, Papadigenopoulos O, Zampetakis E, Zois G (2015) Scheduling mapreduce jobs and data shuffle on unrelated processors. In: International Symposium on Experimental Algorithms, pp. 137–150. Springer
18. Edis EB, Ozkarahan I (2011) A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. Eng Optim 43(2):135–157
19. Edis EB, Oguz C, Ozkarahan I (2013) Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. Eur J Oper Res 230(3):449–463
20. Ibm ilog cplex optimization studio opl language reference manual. White Paper (2018)
21. Ibm ilog cplex optimization studio opl user reference manual. White Paper (2018)
22. Yigitbasi N, Datta K, Jain N, Willke T (2011) Energy efficient scheduling of mapreduce workloads on heterogeneous clusters. In: Green Computing Middleware on Proceedings of the 2nd International Workshop, p. 1. ACM
23. Bampis E, Chau V, Letsios D, Lucarelli G, Milis I, Zois G (2014) Energy efficient scheduling of mapreduce jobs. In: European Conference on Parallel Processing, pp. 198–209. Springer, Berlin
24. Shao Y, Li C, Gu J, Zhang J, Luo Y (2018) Efficient jobs scheduling approach for big data applications. Comput Ind Eng 117:249–261
25. Cai X, Li F, Li P, Ju L, Jia Z (2017) Sla-aware energy-efficient scheduling scheme for hadoop yarn. J Supercomput 73(8):3526–3546
26. Verma A, Cherkasova L, Campbell RH (2011) Aria: automatic resource inference and allocation for mapreduce environments. In: Proceedings of the 8th ACM International Conference on Autonomic Computing, pp. 235–244. ACM
27. Edis EB, Oguz C (2011) Parallel machine scheduling with additional resources: a lagrangian-based constraint programming approach. In: International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, pp. 92–98. Springer
28. Edis EB, Oguz C (2012) Parallel machine scheduling with flexible resources. Comput Ind Eng 63(2):433–447
29. Ham A (2017) Scheduling of dual resource constrained lithography production: using cp and mip/cp. IEEE Trans Semicond Manuf 31(1):52–61. https://doi.org/10.1109/TSM.2017.2768899
30. Gökgür B, Hnich B, Özpeynirci S (2018) Parallel machine scheduling with tool loading: a constraint programming approach. Int J Prod Res 56(16):5541–5557
31. Özpeynirci S, Gökgür B, Hnich B (2016) Parallel machine scheduling with tool loading. Appl Math Modell 40(9–10):5660–5671
32. Arbaoui T, Yalaoui F (2018) Solving the unrelated parallel machine scheduling problem with additional resources using constraint programming. In: Asian Conference on Intelligent Information and Database Systems, pp. 716–725. Springer

33. Fanjul-Peyro L, Perea F, Ruiz R (2017) Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. Eur J Oper Res 260(2):482–493
34. Van den Akker J, Hurkens CA, Savelsbergh MW (2000) Time-indexed formulations for machine scheduling problems: Column generation. INFORMS J Comput 12(2):111–124
35. Queyranne M, Schulz AS (1994) Polyhedral approaches to machine scheduling. TU, Fachbereich 3, Berlin
36. Smith BM, Brailsford SC, Hubbard PM, Williams HP (1996) The progressive party problem: Integer linear programming and constraint programming compared. Constraints 1(1–2):119–138
37. Darby-Dowman K, Little J, Mitra G, Zaffalon M (1997) Constraint logic programming and integer programming approaches and their collaboration in solving an assignment scheduling problem. Constraints 1(3):245–264
38. Darby-Dowman K, Little J (1998) Properties of some combinatorial optimization problems and their effect on the performance of integer programming and constraint logic programming. INFORMS J Comput 10(3):276–286
39. Ibm ilog cplex optimization studiogetting started with scheduling in cplexstudio. White Paper (2018)
40. Huang S, Huang J, Dai J, Xie T, Huang B (2010) The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010), pp. 41–51. IEEE
41. Ahmad F, Lee S, Thottethodi M, Vijaykumar T (2012) Puma: Purdue mapreduce benchmarks suite