# Energy-aware Scheduling of Embarrassingly Parallel Jobs and Resource Allocation in Cloud

Li Shi, *Student Member, IEEE,* Zhemin Zhang,and Thomas Robertazzi, *Fellow, IEEE*

**Abstract**—In cloud computing, with full control of the underlying infrastructures, cloud providers can flexibly place user jobs on suitable physical servers and dynamically allocate computing resources to user jobs in the form of virtual machines. As a cloud provider, scheduling user jobs in a way that minimizes their completion time is important, as this can increase the utilization, productivity, or profit of a cloud. In this paper, we focus on the problem of scheduling embarrassingly parallel jobs composed of a set of independent tasks and consider energy consumption during scheduling. Our goal is to determine task placement plan and resource allocation plan for such jobs in a way that minimizes the Job Completion Time (JCT). We begin with proposing an analytical solution to the problem of optimal resource allocation with pre-determined task placement. In the following, we formulate the problem of scheduling a single job as a Non-linear Mixed Integer Programming problem and present a relaxation with an equivalent Linear Programming problem. We further propose an algorithm named TaPRA and its simplified version TaPRA-fast that solve the single job scheduling problem. Lastly, to address multiple jobs in online scheduling, we propose an online scheduler named OnTaPRA. By comparing with the start-of-the-art algorithms and schedulers via simulations, we demonstrate that TaPRA and TaPRA-fast reduce the JCT by 40%-430% and the OnTaPRA scheduler reduces the average JCT by 60%-280%. In addition, TaPRA-fast can be 10 times faster than TaPRA with around 5% performance degradation compared to TaPRA, which makes the use of TaPRA-fast very appropriate in practice.

## 1 Introduction

In recent years, we have witnessed a dramatic increasing use of cloud computing techniques as it enables on-demand provisioning of computing resources and platforms for users [1]. In a cloud system, users can easily access the required computing resources, while the underlying infrastructure (i.e., data centers composed of physical servers,) is hidden from them and user jobs are executed on Virtual Machines (VMs) whose location is unknown from these users. By deploying the applications or executing the jobs in a cloud, cloud users are able to avoid the cost and responsibility of purchasing, setting up, and maintaining the hardware and software infrastructures and thereby focus more on their missions [2].

In contrast to cloud users' unawareness of the infrastructures hidden behind the cloud, cloud providers have full control of the infrastructure. By widely using modern virtualization techniques, cloud providers can flexibly place jobs on suitable physical servers and dynamically allocate computing resources to user jobs in the form of VMs while keeping the provisioned VMs isolated and interference free from each other. As a large number of user jobs can be simultaneously executed in a cloud, one of the cloud provider's important responsibilities is to properly schedule these jobs and determine an appropriate sharing of resources among these user jobs.

As a cloud provider, scheduling user jobs in a way that minimizes their completion time is very important: With smaller job completion times, the cloud can execute more user jobs; For public clouds, this means generating more profit, while for private clouds, this means higher throughout and therefore usually higher productivity. However, scheduling jobs while minimizing their completion time can be a challenging problem in clouds.

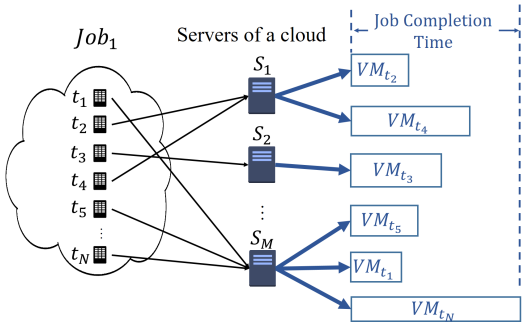



Fig. 1. Job Execution Model.

In this paper, we study the job scheduling problem and focus on scheduling embarrassingly parallel jobs which are composed of a set of independent tasks with very minimal or no data synchronization. A large number of applications belong to this type of jobs. Examples include distributed relational database queries, Monte Carlo simulations, BLAST searches, parametric studies, and image processing applications such as ray tracing [3]. To execute an embarrassingly parallel job, each of its tasks is placed on a physical server and executed in a VM created for that task. The completion time of this job is the completion time of the last finished task, i.e., the makespan of that set of tasks. Fig. 1 shows the execution model of an embarrassingly parallel job. Scheduling such a job includes determining a task placement plan that indicates the servers to execute each task in the job and a resource allocation plan that indicates the amount of computing resources allocated to each task.

To schedule embarrassingly parallel jobs with the goal of minimizing the Job Completion Time (JCT), we need to answer three questions:

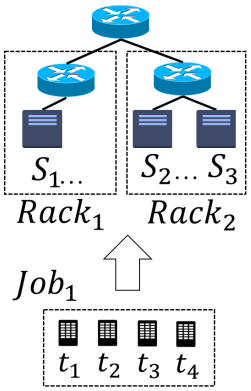
1) How to optimally allocate computing resources to a job with pre-determined task placement plan?

2) How to place tasks and allocate resources for one job?
3) How to address multiple jobs in online scheduling?
Naturally, these three questions are in a progressive relationship. By answering the previous question, we essentially reduce the complexity of the later question. While several approaches has been proposed to schedule independent tasks in data centers [4]–[11], none of them consider task placement and resource allocation together. Motivated by this, in this paper, we focus on the problem of scheduling embarrassingly parallel jobs and propose solutions to the three questions shown above.

Moreover, we consider the energy consumption for executing a job during the scheduling procedure. Along with the rapidly increasing number of cloud users, more and more large-scale data centers comprising tens of thousands of servers are built recently, which leads tremendous amount of energy consumption with huge cost [12]. High energy consumption also reduces system reliability and has negative impacts on the environment [13]. Consequently, reducing the total energy consumption of a cloud is highly desirable. While some approaches [14]–[16] with the objective of reducing the total energy consumption of a cloud have been proposed, in this paper, we focus on scheduling jobs with the goal of minimizing their completion time but constrain the total energy consumed for executing a job.

In summary, our main contributions include

- We formally define the problem of scheduling embarrassingly parallel jobs. We derive a job energy consumption model based on the existing VM power model proposed by other researchers and then formulate the energy consumption constraint. We also formulate the resource availability constraints which limit the amount of resources that can be used by a job. (Section 3)
- We formulate the problem of optimal resource allocation with pre-determined task placement (OptRA) as a convex optimization problem and present an analytical solution of this problem. (Section 4.1)
- We study the problem of scheduling a single embarrassingly parallel job (SJS) and formulate it as a Nonlinear Mixed Integer Programming (NLMIP) problem. We propose a relaxation in which the tasks are assumed to be divisible and transform it to a Linear Programming (LP) problem. (Section 4.2)
- We propose an algorithm named Task Placement and Resource Allocation (TaPRA) and its simplified version TaPRA-fast that solve the SJS problem based on the solution of the relaxed problem. (Section 4.3)
- We propose an online scheduler named OnTaPRA to address multiple jobs in online scheduling. The OnTaPRA scheduler periodically schedules all jobs in the waiting queue by using Shortest Job First (SJF) scheduling policy. For work conservation, it distributes residual capacity of servers to running tasks. (Section 5)
- We evaluate the performance of the proposed TaPRA algorithm and OnTaPRA scheduler through simulations. In offline simulations, we compare the TaPRA algorithm with some existing algorithms. The simulations results show that the TaPRA and TaPRA-fast algorithm can achieve 40%-430% smaller JCT than the existing algorithms. In online simulations, we compare the OnTaPRA scheduler with some existing schedulers. The simulations results show that the OnTaPRA scheduler can achieve 60%-280% smaller average JCT that the existing schedulers. In addition, the results also show that TaPRA-fast can be 10 times faster than TaPRA with around 5% performance degradation compared to TaPRA, which makes the use of TaPRA-fast very applicable in practice (Section 6)




Schedule A with JCT of 33.33

| ID | Load | Placed host | Allocated resource | Execution time | JCT |
|---|---|---|---|---|---|
| $t_1$ | 100 | $S_2$ | 3 | 30 | 33.33 |
| $t_2$ | 200 | $S_1$ | 30 | 6.67 | |
| $t_3$ | 300 | $S_3$ | 12 | 33.33 | |
| $t_3$ | 400 | $S_3$ | 15 | 33.33 | |

Schedule B with JCT of 20

| ID | Load | Placed host | Allocated resource | Execution time | JCT |
|---|---|---|---|---|---|
| $t_1$ | 100 | $S_1$ | 5 | 20 | 20 |
| $t_2$ | 200 | $S_2$ | 10 | 20 | |
| $t_3$ | 300 | $S_3$ | 20 | 20 | |
| $t_3$ | 400 | $S_1$ | 25 | 20 | |

Fig. 2. Example of scheduling a embarrassingly parallel job in cloud.

### 1.1 An Example

To illustrate the job scheduling problem, consider the example shown in Fig. 2, in which we need to schedule an embarrassingly parallel job composed of four tasks ($t_1$, $t_2$, $t_3$, $t_4$) onto three servers ($s_1$, $s_2$, $s_3$) in a data center. Let the available computing resources (for example, the number of CPU cores) on three servers be (35, 10, 30) and load of the four tasks be (100, 200, 400, 500). [1] We further consider a resource availability constraint: the total computing resources allocated to this task set in each rack cannot exceed 30. Fig. 2 also shows two schedules: Schedule A with JCT of 33.33 and Schedule B with JCT of 20. Schedule B has smaller JCT because it proportionally allocates computing resources to tasks. In this way, the maximum task execution time, i.e., the JCT, is efficiently reduced. From this example, we can see that task placement plan and resource allocation plan together determine the JCT. We can achieve the minimum JCT only if we find out the optimal solution on both of them.

However, when the problem scale is large in practice, there exists a vast amount of possible task placement plans and for each placement plan there are very many ways to allocate resources. Searching for the optimal solution in such a huge solution space is not an easy problem to solve. This problem becomes even more complex, when we consider the energy consumption limitation.

## 2 Related Work

A significant amount of research has focused on task/job scheduling and resource allocation in clouds. In this section, we discuss some of the research works that we consider most relevant to our problem from the following aspects: (1) Approaches focusing on scheduling performance, like

1. We temporarily omit the meaning of quantified computing power and task load and simply assume that the execution time of a task is inversely proportional to the amount of resources allocated to the task.

response time, makespan, and completion time; (2) Energy-aware scheduling approaches which set the scheduling goal as minimizing energy consumption or consider the energy consumption during scheduling; (3) Online scheduling approaches which focus on proposing an online scheduler or scheduling policy.

**Approaches Focusing on Scheduling Performance.** This type of approaches mainly focus on optimizing the time-related performance, like response time, makespan or completion time [4]–[11]. Comprehensive surveys about task scheduling and resources scheduling in this category can be found in [4], [5]. Zuo *et al.* [6] propose a multi-objective Ant Colony Algorithm to solve the task scheduling problem. This algorithm considers the makespan and the user's budget costs as constraints of the optimization problem. Tang *et al.* [8] propose a self-adaptive scheduling algorithm for MapReduce jobs. The algorithm decides the start time point of each reduce task dynamically according to each job context, including the task completion time and the size of map output. Tsai *et al.* [9] propose a hyper-heuristic scheduling algorithm which dynamically determines which low-level heuristic is to be used in find better candidate solutions for scheduling tasks in cloud. Verma *et al.* [10] propose an improved Genetic Algorithm which uses the outputs of Max-Min and Min-Min as initial solutions to scheduling independent tasks. Gan *et al.* [11] propose a Genetic Simulated Annealing algorithm to optimize the makespan of a set of tasks, in which Simulated Annealing is used to optimize each offspring generated by the Genetic algorithm. However, all these proposed approaches consider the computing resources allocated to each task as static. Without benefiting from dynamic resource allocation, the efficiency of these approaches in solving the task scheduling problem may be diminished.

**Energy-aware Task Scheduling.** Energy-aware task scheduling has also been given great attention [16]–[21]. Shen *et al.* [17] propose a genetic algorithm to achieve adaptive regulations for different requirements of energy and performance in cloud tasks. In this algorithm, two fitness functions for energy and task completion time are designed for optimizations. Zhao *et al.* [18] propose an energy and deadline aware task scheduling method which models the data-intensive tasks as binary trees.The proposed method aims to schedule Directed Acyclic Graph (DAG)-like workflows. In contrast, in this paper, we focus on embarrassingly parallel jobs composed of independent tasks. Hosseinimotlagh *et al.* [19] propose a VM scheduling algorithm that allocates resources to VMs in a way that the optimal energy level of the host of those VMs is reached. The proposed algorithm assumes that the VMs are pre-mapped onto a host and focuses on allocating resources to the VMs. In contrast to this algorithm, our algorithms determine the task placement. Wu *et al.* [20] develop a scheduling algorithm for the cloud datacenter with a DVFS technique. The algorithm schedules one job at a time and does not consider about co-scheduling between jobs. In addition, this algorithm pre-defines several frequency ranges with corresponding voltage supply and determines a specific range for the job. Mhedheb *et al.* [16] propose a thermal-aware VM scheduling mechanism that achieves both load balance and temperature balance with the final goal of reducing energy consumption. Mhedhed *et al.* analyze the impact of VM migration on energy consumption and utilize the VM migration technique in the proposed mechanism to lower the host temperature. Xiao *et al.* [21] propose a system that dynamically combine VMs running different types of workloads together to improve the overall utilization of resources and reduce the number of running servers, which reduces the energy consumption. The goal of these introduced energy-aware approaches is to minimize the energy consumption in clouds. In contrast, in this paper, we consider the energy consumption as a constraint and set our goal as minimizing the job completion time.

**Online Scheduling.** How to address multiple tasks/jobs in online scheduling is also an important question which has attracted a lot attention [13], [22]–[25] Shin *et al.* [22] modify the conservative backfilling algorithm by utilizing the earliest deadline first and largest weight first policies to address the waiting jobs according to their deadline. The objective of this algorithm is to guarantee the job deadline while improving resource utilization, which is different from the our objective in this paper. Zhu *et al.* [13] design a rolling-horizon scheduling architecture for real-time task scheduling in clouds, which includes an energy consumption model and an energy-aware scheduling algorithm. However, in the proposed architecture, the tasks are scheduled separately. In contrast, in this paper, we addresses tasks belonging to one job together to minimize the job compltion time. Liu *et al.* [23] propose an online scheduler that allows VMs to obtain extra CPU shares when blocked by I/O interrupted recently and thereby reduces the energy-efficiency losses caused by I/O-intensive tasks. Ge *et al.* [25] propose an GA-based task scheduler which evaluates all the waiting tasks and uses a genetic algorithm to schedule these tasks with the goal of achieving better load balance.

## 3 Problem Definition

In this paper, we consider scheduling jobs composed of independent tasks in a data center comprising heterogeneous servers. To schedule such a job, we are required to place each of its tasks onto a server and launch a VM with certain amount of computing capacity to execute that task. We now introduce input, output, objective function and constraints considered in the problem.

### 3.1 Input

The input contains user jobs submitted by users at different times and a data center with a set of heterogeneous servers used to execute those jobs.

**Job.** A job $J$ comprise a set of independent tasks. It is defined as $J = \{T_1, T_2, \ldots, T_N\}$ in which $T_i$ is the $i$th independent task of that job. The load $Load_i$ of the task $T_i$ is defined as the execution time of $T_i$, when it is placed on a unit-efficiency server and executed on a VM with unit computing capacity.

**Data Center.** The data center used to execute user jobs is defined as $DC = \{S_1, S_2, \ldots, S_M\}$ in which $S_j$ is the $j$th server in the data center. The available computing resources of server $S_j$ is denoted by $C_{s_j}$.

Due to the heterogeneity of servers, different severs may have different efficiencies of executing the same task [26]. To model such heterogeneity, we denote the efficiency of executing task $T_i$ on server $S_j$ by $\lambda_{ij}$ ($\lambda_{ij} \in (0, 1]$). Correspondingly, when a task $T_i$ is placed on a server $S_j$ with efficiency $\lambda_{ij}$ and executed on a VM with computing power $c_{t_i}$, the execution time of $T_i$, denoted by $et_{t_i}$, is

$$et_{t_i} = \frac{Load_i}{\lambda_{ij} c_{t_i}}. \quad (1)$$

### 3.2 Output

For each input job, the output contains a task placement plan and a resource allocation plan.

**Task placement plan.** A task placement plan indicates the servers to execute the input job's tasks. We use binary variables $x_{ij}$ to present such a plan. Specifically, if task $T_i$ is placed on server $S_j$, the value of $x_{ij}$ is 1; otherwise its value is 0.

**Resource allocation plan.** A resource allocation plan indicates the amount of computing resources allocated to each task, i.e., the computing capacity allocated to the VM created to execute the tasks. We denote the set of VMs by $\mathcal{VM} = \{VM_{t_1}, \ldots, VM_{t_M}\}$, where $VM_{t_i}$ is the VM created to execute task $T_i$, and denote the resource allocation plan by $\vec{c} = \{c_{t_1}, \ldots, c_{t_N}\}$ in which $c_{t_i}$ is the amount of computing resources allocated to $VM_{t_i}$.

### 3.3 Objective

**Single job scheduling.** When scheduling a single job, our objective is to determine a task placement plan and a resource allocation plan while minimizing the job completion time (JCT). Because a job is not completed until all its tasks finish, the JCT essentially equals the completion time of the last finished task. Let the completion time of task $T_i$ be $et_{t_i}$, based on Equation (1), the objective function is

$$Minimize \quad \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{Load_i}{\sum_{j=1}^{M} x_{ij} \lambda_{ij} c_{t_i}} \right\}. \quad (2)$$

**Online scheduling.** In online scheduling, multiple jobs arrive in a time sequence and we are required to schedule all these jobs. In this situation, our objective is to minimize the average completion time of these jobs, i.e., the average JCT.

### 3.4 Constraints

#### 3.4.1 Task Placement Constraints

Because each task should be placed on only one server, we have the following constraint:

$$\sum_{j=1}^{M} x_{ij} = 1, \ i = 1, \ldots, N. \quad (3)$$

#### 3.4.2 Resource Availability Constraints

When allocating resources to the tasks of a job, there may be limitations on the total amount of computing resource that can be allocated to that job on a server or a subset of servers. Such constraints are usually enforced by system administrator to maintain proper sharing of resources among multiple jobs belong to different users. We can formulate these constraints as

$$\sum_{j=1}^{M} A_{jk} \sum_{i=1}^{N} x_{ij} c_{t_i} \leq B_k, \ k = 1, \ldots, R. \quad (4)$$

where $\sum_{i=1}^{N} x_{ij} c_{t_i}$ is the total computing capacity allocated to the input jobs on server $S_j$; $A_{jk}$ is the coefficient in the $k$th constraint; $B_k$ is the total resources allowed to be allocated in the $k$th constraint.

TABLE 1
List of constants and variables.

| Constants | |
|---|---|
| Name | Description |
| $J$ | the input embarrassingly parallel job |
| $T_i$ | the $i$th independent task of job $J$ |
| $Load_i$ | the load of the task $T_i$ |
| $S_j$ | the $j$th server in the data center |
| $C_{s_j}$ | the available computing resource on server $S_j$ |
| $VM_{t_i}$ | the VM executing task $T_i$ |
| $\alpha_{VM_{t_i}}$ | the power model constant of $VM_{t_i}$ |
| $\lambda_{ij}$ | the efficiency of executing task $T_i$ on server $S_j$ |
| $A_{jk}$ | the coefficient in the $k$th availability constraint for $S_j$ |
| $B_k$ | the total allowed resources in the $k$th availability constraint |
| $X_{ij}$ | the determined placement between task $T_i$ and server $S_j$ |
| $E_{MAX}$ | the maximum allowed energy consumption |
| Variables | |
| Name | Description |
| $x_{ij}$ | the placement relationship between task $T_i$ and server $S_j$ |
| $c_{t_i}$ | the amount of computing resources allocated to $T_i$ |
| $c_{s_j}$ | the total amount of resources allocated to the job on $S_j$ |
| $et_{t_i}$ | the completion time of task $T_i$ |
| $l_{ij}$ | the load of sub-task $T_{ij}$ |
| $c_{t_{ij}}$ | the amount of resources allocated to sub-task $T_{ij}$ |

#### 3.4.3 Energy Consumption Constraints

As the energy cost can contribute a significant part of operating cost of a data center [12], the system administrator may also limit the total amount of energy that can be consumed by a job. To formulate this energy consumption constraint, we first consider the VM power model and energy consumption model.

**VM Power Model.** Power modeling for VMs in data centers has attracted significant attention [27]–[30]. While both linear and non-linear power model are proposed, the linear model is the most widely used method in the estimation of power consumption [30], whose accuracy has been proved [27]–[29]. The linear VM power model have also been used in many existing task scheduling and resource allocation approaches [2], [13], [17], [31].

In the linear model, the total power consumption $P_s$ of a physical server is composed of the static power $P_{static}$ and the dynamic power $P_{dynamic}$. While $P_{static}$ is usually constant regardless of whether VMs are running or not, as long as the server is turned on, $P_{dynamic}$ is consumed by VMs running on the server. Suppose that $n$ VMs are running on a server, then the server power consumption $P_s$ is

$$P_s = P_{static} + \sum_{i=1}^{n} P_{VM_i}, \text{subject to} \sum_{i=1}^{n} c_{VM_i} \leq C_s, \quad (5)$$

in which $C_s$ is the total computing capacity of the server; $P_{VM_i}$ and $c_{VM_i}$ are the power consumption and the computing capacity of VM $i$. $P_{VM_i}$ can be further decomposed

into power of components such as CPU, memory, disk and IO devices [28], thus it can be calculated as

$$P_{VM_i} = P_{VM_i}^{CPU} + P_{VM_i}^{Memory} + P_{VM_i}^{Disk} + P_{VM_i}^{IO}. \quad (6)$$

In this paper, we mainly focus on the power consumption of CPU utilized by a VM, because the CPU utilization of a VM is directly related to the execution time of tasks running on that VM. Therefore, we approximate the VM power consumption by the CPU power consumption of a VM, following similar setting in existing work [2]. A utilization based VM power model then is

$$P_{VM_i} = P_{VM_i}^{CPU} = \alpha_{VM_i} \cdot c_{VM_i}, \quad (7)$$

where $P_{VM_i}$ is the power consumption of $VM_i$, $c_{VM_i}$ is the amount of computing resources allocated to the VM and $\alpha_{VM_i}$ is model specific constant, following similar model proposed in previous work [28], [30].

Based on the VM power model (7), the energy consumed by VM $VM_{t_i}$, denoted by $E_{VM_{t_i}}$, is

$$E_{VM_{t_i}} = P_{VM_{t_i}} \cdot et_{t_i} = \alpha_{VM_{t_i}} c_{t_i} \cdot et_{t_i} = \frac{\alpha_{VM_{t_i}} Load_i}{\sum_{j=1}^{M} x_{ij}\lambda_{ij}}. \quad (8)$$

Let $E_{total}$ denote the total energy consumption of the input job and $E_{MAX}$ denote the maximum energy consumption allowed, the energy consumption constraint is

$$E_{total} = \sum_{t=1}^{N} E_{VM_{t_i}} = \sum_{i=1}^{N} \frac{\alpha_{VM_{t_i}} Load_i}{\sum_{j=1}^{M} x_{ij}\lambda_{ij}} <= E_{MAX}. \quad (9)$$

## 4 Scheduling a Single Job with Independent Tasks

In this section, we focus on scheduling a single job. We start from its sub-problem: Optimal Resource Allocation with Pre-determined Task Placement (OptRA).

### 4.1 Optimal Resource Allocation with Pre-determined Task Placement (OptRA)

In the OptRA problem, the task placement plan has already been determined, i.e., the value of $x_{ij}$ is known. For convenience and clarity, we use $X_{ij}$ to indicate the determined task placement plan. Our goal is to allocate computing resources to these tasks while minimizing the JCT.

With the determined task placement, the objective (2) becomes

$$Minimize \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{\frac{Load_i}{\sum_{j=1}^{M} X_{ij}\lambda_{ij}}}{c_{t_i}} \right\}. \quad (10)$$

The resource availability constraints (4) become

$$\sum_{i=1}^{N}\sum_{j=1}^{M} X_{ij}A_{jk}c_{t_i} \le B_k,\ k = 1, \ldots, R. \quad (11)$$

We can then formulate the OptRA problem as

**OptRA**

$$Minimize \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{L_{t_i}}{c_{t_i}} \right\}, \quad (12)$$

**Subject to**

$$\sum_{i=1}^{N} P_{ik}c_{t_i} \le B_k, k = 1, \ldots, R, \quad (13)$$

$$c_{t_i} \ge 0,\ i = 1, \ldots, N, \quad (14)$$

$$L_{t_i} = \frac{Load_i}{\sum_{j=1}^{M} X_{ij}\lambda_{ij}}, P_{ik} = \sum_{j=1}^{M} X_{ij}A_{jk}. \quad (15)$$

**Remarks:**

- While $L_{t_i}$ and $P_{ik}$ are used to simplify the formulation of the problem, $L_{t_i}$ also stands for the equivalent load of $T_i$ considering the execution efficiency of the server on which $T_i$ is placed; $P_{ik}$ is the coefficient of $T_i$ in the $k$th availability constraint.
- Task placement constraints (3) and energy consumption constraints (9) are not included in OptRA, as they are only related with the variable $x_{ij}$ whose value is already determined in the above problem.

We observe that the constraints (13) and (14) are all affine on $c_{t_i}$. Meanwhile, the function $et_{t_i}$ is convex, because its second derivative is nondecreasing when $c_i$ is larger than 0. Therefore, according to [32], the objective function, i.e., the pointwise maximum function of $et_{t_i}$, is also a convex function. As a result, OptRA is a convex optimization problem.

#### 4.1.1 Analytical Solution

While existing convex optimization algorithms [32] can be used to solve the OptRA problem, we develop an analytical solution which is more efficient. Specifically, we define a vector $\vec{c}^* = \{c_1^*, c_2^*, \ldots, c_N^*\}$ with

$$c_i^* = \min_{k \in \mathcal{R}_i} \left\{ \frac{L_{t_i}}{\sum_{j=1}^{N} P_{jk}L_{t_j}} B_k \right\},\ i = 1, \ldots, N, \quad (16)$$

where $\mathcal{R}_i$ is the set of constraints in which the coefficient of variable $c_i$ is not zero. We now show that this vector $\vec{c}^*$ is an optimal solution of the OptRA problem. We have the following lemma and theorem.

**Lemma 1.** *Assume that for the vector $\vec{c}^*$ defined in Equation 16, task $p$ has the largest finish time, i.e., $et_p^* = L_{t_p}/c_p^* = \max_{i=1,\ldots,N}\{et_i^*\}$. Also assume that $c_p^*$ obtains the minimum value when constraint $u$ is considered,*

$$c_p^* = \min_{k \in \mathcal{R}_p} \left\{ \frac{L_{t_p}}{\sum_{j=1}^{N} P_{jk}L_{t_j}} B_k \right\} = \frac{L_{t_p}}{\sum_{j=1}^{N} P_{ju}L_{t_j}} B_u. \quad (17)$$

*then every $c_i$ subjected to constraint $u$, i.e., $P_{iu} \neq 0$, obtains the minimum value when constraint $u$ is considered, i.e.,*

$$c_i^* = \min_{k \in \mathcal{R}_i} \left\{ \frac{L_{t_i}}{\sum_{j=1}^{N} P_{jk}L_{t_j}} B_k \right\} = \frac{L_{t_i}}{\sum_{j=1}^{N} P_{ju}L_{t_j}} B_u. \quad (18)$$

*Proof.* To begin with, assume that there exists a variable $c_q^*$ with $P_{qu} \neq 0$, which gets the minimum value when constraint $v$ (other than constraint $u$) is considered, i.e.,

$$c_q^* = \min_{k \in \mathcal{R}_q} \left\{ \frac{L_{t_q}}{\sum_{j=1}^{N} P_{jk}L_{t_j}} B_k \right\} = \frac{L_{t_q}}{\sum_{j=1}^{N} P_{jv}L_{t_j}} B_v. \quad (19)$$

Next, we define $c'_q$ as

$$c'_q = \frac{L_{t_q}}{\sum_{j=1}^{N} P_{ju} L_{t_j}} B_u. \quad (20)$$

Based on Equations (19) and (20), we have

$$et'_q = \frac{L_{t_q}}{c'_q} < \frac{L_{t_q}}{c^*_q} = et^*_q. \quad (21)$$

On the other hand, based on Equation (17), we have

$$et^*_p = \frac{L_{t_p}}{c^*_p} = \frac{\sum_{j=1}^{N} P_{ju} L_j}{B_u} = \frac{L_{t_q}}{c'_q} = et'_q. \quad (22)$$

Now using the Inequality (21) and Equation (22), we get

$$et^*_p = et'_q < et^*_q, \quad (23)$$

which conflicts with the assumption that $et^*_p = \max_{i=1,\ldots,N}\{et^*_i\}$. Therefore, there does not exist a variable $c^*_q$ and a constraint $v$ that satisfies the Equation (19). As a result, we have proved the lemma. □

**Theorem 1.** *The vector $\vec{c}^*$ defined in Equation 16 is an optimal solution of the OptRA problem.*

*Proof.* Assume that $et^*_p = L_{t_p}/c^*_p = \max_{i=1,\ldots,N}\{et^*_i\}$ and $c^*_p$ obtains the minimum value when constraint $u$ is considered. Then, according to Lemma 1, there exists

$$c^*_i = \frac{L_{t_i}}{\sum_{j=1}^{N} P_{ju} L_j} B_u, \ \forall i \ that \ P_{ju} \neq 0. \quad (24)$$

Based on this equation, for every $i$ with $P_{iu} \neq 0$, we have

$$et^*_i = \frac{L_{t_i}}{c^*_i} = \frac{\sum_{j=1}^{N} P_{ju} L_{t_j}}{B_u} = \frac{L_{t_p}}{c^*_p} = et^*_p. \quad (25)$$

Now assume that instead of $\vec{c}^*$, a vector $\vec{c}'$ is the optimal solution of the problem. Also assume that $et'_q = L_{t_q}/c'_q = \max_{i=1,\ldots,N}\{et'_i\}$. Together with Equation (25), we have

$$et'_i \leq et'_q < et^*_p = et^*_i, \ \forall i \ that \ P_{iu} \neq 0. \quad (26)$$

Naturally, we have

$$c'_i > c^*_i, \ \forall i \ that \ P_{iu} \neq 0. \quad (27)$$

On the other hand, from Equation (24), we can get

$$\sum_{i=1}^{N} P_{iu} c^*_i = B_u. \quad (28)$$

Putting Inequity (27) and Equation (28) together, we have

$$\sum_{i=1}^{N} P_{iu} c'_i > B_u, \quad (29)$$

which conflicts with the assumption that $\vec{c}'$ is a feasible solution. As a result, there does not exist a feasible solution that is better than $\vec{c}^*$. Therefore, the vector $\vec{c}^*$ defined in Equation (16) is an optimal solution of the OptRA problem. □

Note that the analytical solution (16) has important usage when scheduling a single job. It essentially reduces the dimension of the problem: For any determined task placement plan, we can use Equation (16) to calculate the corresponding optimal resource allocation plan.

### 4.2 Formulation of the Single Job Scheduling Problem and Its Solvable Relaxation

We now study the Single Job Scheduling (SJS) problem in which we are required to determine the task placement plan and resource allocation plan for an input job. The task execution model of the input job has been shown in Fig. 1.

Based on the objective and constraints introduced in Section 3, the SJS problem can be formulated as

**SJS**

$$Minimize \ \max_{i=1,\ldots,N} \left\{ et_{t_i} \mid et_{t_i} = \frac{Load_i}{\sum_{j=1}^{M} x_{ij} \lambda_{ij} c_{t_i}} \right\}, \quad (30)$$

**Subject to**

$$\sum_{i=1}^{N} \frac{\alpha_{VM_{t_i}} Load_i}{\sum_{j=1}^{M} x_{ij} \lambda_{ij}} \leq E_{MAX}, \quad (31)$$

$$\sum_{j=1}^{M} A_{jk} \sum_{i=1}^{N} x_{ij} c_{t_i} \leq B_k, \ k = 1, \ldots, R, \quad (32)$$

$$\sum_{j=1}^{M} x_{ij} = 1, \ i = 1, \ldots, N, \quad (33)$$

$$x_{ij} = 0 \ or \ 1, \ i = 1 \ldots, N, \ j = 1, \ldots, M, \quad (34)$$

$$c_{t_i} \geq 0, \ i = 1, \ldots, N. \quad (35)$$

**Remarks:**

- The objective (30) and constraints (31)-(33) are formally defined in Section 3.
- Constraints (34) and (35) are domain constraints.

Naturally, the SJS problem is a NLMIP problem which is hard to solve directly. To solve this problem, we first propose a solvable relaxation and then determine a solution of the SJS problem based on the solution of the relaxation.

#### *4.2.1 A Relaxation of the SJS problem and An Equivalent Linear Programming Problem*

Because the SJS problem is a NLMIP problem, a straightforward relaxation is relaxing the binary variable $x_{ij}$ to a real variable. However, due to the term $x_{ij}c_{t_i}$, this relaxation is a NLP problem which is still hard to solve.

To obtain a solvable relaxation, we assume that the tasks of the input job are divisible [33]–[35] and each task $T_i$ is divided into $M$ sub-tasks and placed on $M$ servers respectively. Let $t_{ij}$ denote the sub-task of $T_i$ placed on server $S_j$ and let $l_{ij}$ denote the load of $t_{ij}$. A VM is then created for each sub-task placed on each server. Fig. 3 shows this execution model.

Furthermore, let $VM_{t_{ij}}$ denote the VM created by server $S_j$ to execute sub-task $t_{ij}$ and let $c_{t_{ij}}$ denote the amount of resources allocated to $VM_{t_{ij}}$. The SJS problem is now relaxed to a problem of determining the value of $l_{ij}$ and $c_{t_{ij}}$ with the goal of minimizing the JCT. We name this new problem as **SJS-Relax-Divisible**.

The execution time of sub-task $t_{ij}$, denoted by $et_{t_{ij}}$, is

$$et_{t_{ij}} = \frac{l_{ij}}{\lambda_{ij} c_{t_{ij}}}. \quad (36)$$

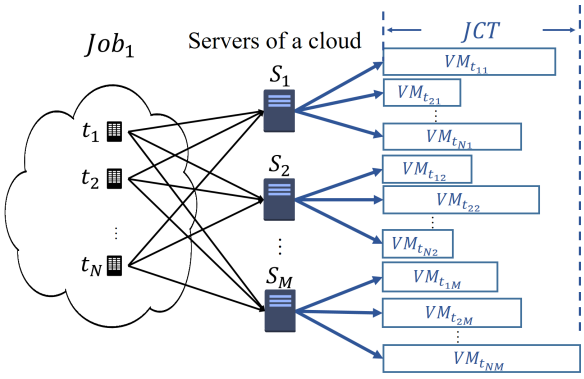



Fig. 3. Job Execution Model for SJS-Relax-Divisible.

The energy consumed for executing $t_{ij}$ is

$$E_{t_{ij}} = P_{VM_{t_{ij}}} \cdot et_{t_{ij}} = \alpha_{VM_{t_{ij}}} c_{t_{ij}} \cdot et_{t_{ij}} = \alpha_{VM_{t_{ij}}} \cdot \frac{l_{ij}}{\lambda_{ij}}. \quad (37)$$

The total resources allocated by server $S_j$, i.e., $c_{s_j}$, is

$$c_{s_j} = \sum_{i=1}^{N} c_{t_{ij}}. \quad (38)$$

Based on the above equations, the SJS-Relax-Divisible problem can be formulated as

**SJS-Relax-Divisible**

$$Minimize \quad \max_{\substack{i=1,\ldots,N \\ j=1,\ldots,M}} \left\{ et_{t_{ij}} \mid et_{t_{ij}} = \frac{l_{ij}}{\lambda_{ij} c_{t_{ij}}} \right\}, \quad (39)$$

**Subject to**

$$\sum_{i=1}^{N} \sum_{j=1}^{M} \alpha_{VM_{t_{ij}}} \cdot \frac{l_{ij}}{\lambda_{ij}} \le E_{MAX}, \quad (40)$$

$$\sum_{j=1}^{M} A_{jk} \sum_{i=1}^{N} c_{t_{ij}} \le B_k, \ k = 1, \ldots, R, \quad (41)$$

$$\sum_{j=1}^{M} l_{ij} = Load_i, \ i = 1, \ldots, N, \quad (42)$$

$$c_{t_{ij}} \ge 0, \ l_{ij} \ge 0, \ i = 1, \ldots, N, \ \ j = 1, \ldots, M. \quad (43)$$

**Remarks:**

- The objective (39) is to minimize the maximum execution time of all sub-tasks, which also minimizes the completion time of the input job.
- Constraints (40) and (41) are energy consumption constraints and resource availability constraints.
- Constraints (42) ensure that the total load of all sub-tasks of $T_i$ equals to the load of task $T_i$, and constraints (43) are domain constraints.

We observe that in the SJS-Relax-Divisible problem, all constraints are affine and the objective is the pointwise maximum of $NM$ ratios of affine functions. Therefore, this problem is a Generalized Linear Fractional Programming (GLFP) problem, which can be solved as a sequence of LP feasibility problems [32].

However, solving a GLFP problem can be time-consuming as it needs to solve a set of LP feasibility problems. To avoid this, we further transform the SJS-Relax-Divisible problem into an equivalent LP problem.

**An Equivalent LP Problem: SJS-Relax-LP.** The transformation starts from defining variable $T$ as the JCT, i.e.,

$$T = \max_{\substack{i=1,\ldots,N \\ j=1,\ldots,M}} \left\{ \frac{l_{ij}}{\lambda_{ij} c_{t_{ij}}} \right\}. \quad (44)$$

Substituting $T$ into the objective function, we have

$$Minimize \quad T$$

$$T \ge \frac{l_{ij}}{\lambda_{ij} c_{t_{ij}}}, \ i = 1, \ldots, N, \ j = 1, \ldots, M. \quad (45)$$

Further define variable $p_{ij}$ as

$$p_{ij} = c_{t_{ij}} \cdot T, \quad (46)$$

and then reformulate the constraints (45) as

$$\lambda_{ij} \cdot p_{ij} \ge l_{ij}, \ i = 1, \ldots, N, \ j = 1, \ldots, M. \quad (47)$$

On the other hand, by substituting $p_{ij}$ into constraints (41), we have

$$\sum_{j=1}^{M} A_{jk} \sum_{i=1}^{N} p_{ij} \le B_k \cdot T, \ k = 1, \ldots, R. \quad (48)$$

Integrating all transformations shown above together, we obtain an equivalent problem, named **SJS-Relax-LP**, as shown below.

**SJS-Relax-LP**

$$Minimize \ T \quad (49)$$

**Subject to**

$$\lambda_{ij} \cdot p_{ij} \ge l_{ij}, \ i = 1, \ldots, N, \ j = 1, \ldots, M, \quad (50)$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M} \alpha_{VM_{t_{ij}}} \cdot \frac{l_{ij}}{\lambda_{ij}} \le E_{MAX}, \quad (51)$$

$$\sum_{j=1}^{M} A_{jk} \sum_{i=1}^{N} p_{ij} \le B_k \cdot T, \ k = 1, \ldots, R, \quad (52)$$

$$\sum_{j=1}^{M} l_{ij} = Load_i, \ i = 1, \ldots, N, \quad (53)$$

$$c_{t_{ij}} \ge 0, \ l_{ij} \ge 0, \ i = 1, \ldots, N, \ \ j = 1, \ldots, M. \quad (54)$$

Naturally, the SJS-Relax-LP problem is a Linear Programming problem as its objective function and all constraints are linear. Therefore, it can be efficiently solved by linear programming algorithms.

### 4.3 The Task Placing and Resource Allocation (TaPRA) Algorithm and Its Simplified Version: TaPRA-fast

Based on the relaxation SJS-Relax-LP, we propose an algorithm, called Task Placing and Resource Allocation (TaPRA), to solve the SJS problem.

The TaPRA algorithm has three phases. In the first phase, TaPRA obtains a solution of the SJS-Relax-LP problem; in the second phase, it determines an initial solution of the SJS

**Algorithm 1** The TaPRA Algorithm

**function** TAPRA($A_{jk}$,$B_k$,$Load_i$,$E_{MAX}$)
*Phase I:*
  Solve SJS-Divisible and get $\{l_{ij}, p_{ij}\}$
*Phase II:*
  **for** each $T_i \in J$ **do**
    $u \leftarrow \text{argmax}_{k=1,\dots,M}\{l_{ik}\}$;
    $x_{ij} \leftarrow 1$ if $j == u$; otherwise, $x_{ij} \leftarrow 0$;
  **end for**
  Get $\{c_{t_i}, et_{t_i}\}$ using Equation (16); Update $E_{total}$;
  **if** $E_{total} > E_{MAX}$ **then** call REC;
*Phase III:*
  **while** true **do**
    $\mathcal{T}_{max} \leftarrow \{T_i \mid et_{t_i} == JCT\}$;
    **for** each $T_i \in \mathcal{T}_{max}$ **do**
      $\mathcal{TM}_{valid} \leftarrow \{\text{all valid } TM_{ij}\}$;
      **if** $\mathcal{TM}_{valid} == \emptyset$ **then continue;**
      $TM_{iu} \leftarrow \text{argmin}_{TM_{ij} \in \mathcal{TM}_{valid}}\{\text{new } et_{t_i}\}$;
      Perform $TM_{iu}$ and update $\{c_{t_i}, et_{t_i}\}$; **break;**
    **end for**
    **if** no task movement is performed **then break**;
  **end while**
  **return** $x_{ij}$ and $c_{t_i}$
**end function**

problem based on the solution of the SJS-Relax-LP problem; in the last phase, it utilizes a local search procedure to further optimize the obtained initial solution. Algorithm 1 shows the pseudocode of TaPRA.

We now introduce the details of each phase.

**Phase I: Solve the relaxed problem.** The TaPRA algorithm starts from solving the relaxed LP problem SJS-Relax-LP. Let the solution of SJS-Relax-LP be $T$, $l_{ij}$, and $p_{ij}$.

**Phase II: Obtain an initial solution of the SJS problem.** In this phase, TaPRA obtains an initial solution of the SJS problem from the solution of SJS-Relax-LP in two steps:

- First, it determines the value of variable $x_{ij}$, i.e., obtaining the task placement plan. Specifically, for each task $T_i$, TaPRA selects sub-task $t_{iu}$ that gets the largest portion of $T_i$ and assigns task $T_i$ to server $s_u$. Such assignment can be presented as

$$x_{ij} = \begin{cases} 1, & \text{if } j = \underset{k=1,\dots,M}{\text{argmax}} \{l_{ik}\} \\ 0, & \text{otherwise} \end{cases}, \ i = 1, \dots, N. \tag{55}$$

  The intuition is that if a server gets a larger portion of task $T_i$, the result may be "closer" to the optimal solution by assigning $T_i$ to that server.
- Second, the TaPRA algorithm determines the value of $c_{t_i}$, i.e., the resource allocation plan. Because the task placement has been determined in the first step, the SJS problem is naturally reduced to the OptRA problem. Therefore, the TaPRA algorithm simply utilizes Equation (16) to determine the value of $c_{t_i}$.

However, in some cases, the assignment (55) may lead to a violation of the energy consumption constraint (31), because the server that gets the largest percentage of a task may not be the one with the highest efficiency to execute that task, i.e., consumes the least energy to execute that task.

**Algorithm 2** Reduce Energy Consumption (REC)

**function** REC($x_{ij}$,$c_{t_i}$,$A_{jk}$,$B_k$,$Load_i$,$E_{MAX}$,$E_{total}$)
  **while** $E_{total} > E_{MAX}$ **do**
    List $\mathcal{H} \leftarrow \{\}$;
    **for** each $t_i \in J$ & each $S_j \in DC$ **do**
      $TM_{ij} \leftarrow \{T_i, s_{src_i}, s_j, \Delta E_{ij}, \Delta JCT_{ij}\}$;
      **if** $\Delta E_{ij} < 0$ **then** add $TM_{ij}$ into $\mathcal{H}$;
    **end for**
    $TM_{uv} \leftarrow \text{argmin}_{TM_{ij} \in \mathcal{H}}\{\Delta JCT_{ij}\}$;
    $x_{u,s_{src_u}} \leftarrow 0$ and $x_{uv} \leftarrow 1$;
    Get $\{c_{t_i}, et_{t_i}\}$ using Equation (16); Update $E_{total}$;
  **end while**
**end function**

To resolve the problem, the TaPRA algorithm utilizes a procedure called Reduce Energy Consumption (REC) to reduce total energy consumption by performing task movement. A task movement is moving a task from one server to another server and is defined by $TM_{ij} = \{T_i, s_{src_i}, s_j, \Delta E_{ij}, \Delta JCT_{ij}\}$, where task $T_i$ is moved from the original server $s_{src_i}$ to server $S_j$; $\Delta E_{ij}$ and $\Delta JCT_{ij}$ are the difference of total energy consumption and JCT respectively between the two task placements. The REC procedure runs in iterations. In each iteration, it finds out all task movements that can reduce the total energy consumption and performs the one with the smallest $\Delta JCT_{ij}$. If the new task placement satisfies the energy consumption constraint, the REC procedure stops; otherwise, it starts the next iteration. Algorithm 2 describes the REC procedure.

**Phase III: Local search.** In this phase, the TaPRA algorithm utilizes a local search procedure to further improve the initial solution obtained in phase II.

In this local search procedure, TaPRA runs in iterations. In each iteration, it starts with identifying all tasks with the largest execution time and putting them into a set named $\mathcal{T}_{max}$. Subsequently, TaPRA iteratively considers each task in the set $\mathcal{T}_{max}$. For each task $T_i \in \mathcal{T}_{max}$, the TaPRA algorithm calculates all valid task movements (i.e., the energy consumption constraint is not violated and the execution time of $T_i$ is reduced after performing the movement.) and put them into a set $\mathcal{TM}_{valid}$; it then selects the task movement which reduces the execution time of task $T_i$ most; in the following, TaPRA performs the selected task movement and update $x_{ij}$ and $c_{t_i}$. If a task movement is performed, the TaPRA algorithm starts a new iteration of phase III.

If the TaPRA algorithm cannot improve the execution time of any tasks in $\mathcal{T}_{max}$ in some iteration, it then finishes and returns the current solution $x_{ij}$ and $c_{t_i}$, as it cannot improve the JCT anymore.

#### 4.3.1 TaPRA-fast: A Simplified Version of TaPRA

The TaPRA algorithm begins with solving the SJS-Relax-LP problem which is a LP problem. When the problem's scale is large enough, solving this problem can be time-consuming. On the other hand, we observe that the local search procedure in the TaPRA algorithm can be used to optimize any feasible schedules. With such observations, we propose TaPRA-fast, a simplified version of TaPRA with less time complexity.

**Algorithm 3** Online scheduler: OnTaPRA

1: **procedure** ONTAPRA(Current time $curT$)
2: Add all jobs arriving at $curT$ to $Q_{wait}$;
3: $\mathcal{T}_{finish} \leftarrow$ all tasks finishing at $curT$;
4: **if** $\mathcal{T}_{finish} \neq \emptyset$ **then**
5: Release all computing resources allocated to $\mathcal{T}_{finish}$;
6: Call the DRC procedure;
7: **end if**
8: **if** $curT$ mod $\Delta T_{schedule} == 0$ **then**
9: Release resource allocated in last DRC call;
10: Use a scheduling policy to schedule jobs in $Q_{wait}$;
11: Call the DRC procedure;
12: **end if**
13: **end procedure**

TaPRA-fast has two phases: First, it obtains an initial solution; Second, it utilizes the local search procedure used in the TaPRA algorithm to optimize the initial solution. To obtain an initial solution, the TaPRA-fast algorithm places each task $T_i$ on the server with the highest efficiency on executing this task, i.e., $\lambda_{ij}$. Subsequently, TaPRA-fast calculates $c_{t_i}$ and JCT based on the determined task placement.

## 5 Online Scheduling

In the previous section, we have studied how to schedule a single job composed of a set of independent tasks with the goal of minimizing its completion time and have proposed algorithms to solve this problem.

However, in practice, jobs arrive the system in a time sequence and in a long term view, our goal is to minimize the average JCT of all arrived jobs. With this goal, it may be inefficient to address each of the arrived jobs individually. Motivated by this, we propose an online scheduler named OnTaPRA, which periodically schedules all arrived jobs together. Algorithm 3 shows the main logic of OnTaPRA.

**Online Scheduler: OnTaPRA.** The OnTaPRA scheduler puts each arrived job into a waiting queue $Q_{wait}$ and keeps track of the waiting time of each job in $Q_{wait}$.

The OnTaPRA scheduler periodically schedule all jobs in the waiting queue together. To schedule the jobs in $Q_{wait}$, the OnTaPRA scheduler uses a scheduling policy named Shortest Job First (SJF) which is introduced later. While all scheduled jobs are placed on corresponding servers according their task placement plan, those jobs that fail to be scheduled stay in $Q_{wait}$.

After the jobs in the waiting queue are scheduled and placed on the servers, there may be residual computing capacity on those servers. These residual resources are actually wasted, as no job can utilize these computing resources until the next call of scheduling algorithm. To follow the work conservation rule, the OnTaPRA scheduler further uses a procedure named Distribute Residual Capacity (DRC) to temporarily distribute the residual computing capacity of each server to the tasks running on that server. In this way, all computing resources of a server will be in use as long as there are tasks running on it. On the other hand, in the next round of scheduling, that residual capacity temporarily distributed will be recollected and treated as the available capacity of the servers. The details of the DRC procedure are introduced later.

**Algorithm 4** The SJF Scheduling Policy

1: **procedure** SJF(Current Waiting Queue $Q_{wait}$)
2: $Q_{wait}^{new} \leftarrow \emptyset$;
3: **while** $Q_{wait} \neq \emptyset$ **do**
4: Perform the MAR test;
5: **if** test fails **then** Add $Q_{wait}$ to $Q_{wait}^{new}$; **break**;
6: **for** each job $J_k \in Q_{wait}$ **do**
7: Calculate $\{x_{ij}^k, c_{T_i}^k, JCT_k\}$ using TaPRA;
8: **if** fails **then** Move $J_k$ from $Q_{wait}$ to $Q_{wait}^{new}$;
9: **end for**
10: $J_u \leftarrow \text{argmin}_{J_k \in Q_{wait}}\{JCT_k\}$;
11: Execute $J_u$; Remove $J_u$ from $Q_{wait}$;
12: **end while**
13: $Q_{wait} \leftarrow Q_{wait}^{new}$;
14: **end procedure**

**Algorithm 5** The DRC procedure

1: **procedure** DRC
2: **for** each server $S_j \in DC$ **do**
3: $C_{s_j}^{resi} \leftarrow$ residual capacity of $S_j$;
4: $\mathcal{T}_{s_j} \leftarrow$ tasks on $S_j$; $L_{s_j} \leftarrow$ total load of $\mathcal{T}_{s_j}$;
5: **for** each $T_i \in \mathcal{T}_{s_j}$ **do** $c_{t_i} \leftarrow c_{t_i} + C_{s_j}^{resi} \cdot \frac{L_{t_i}}{L_{s_j}}$;
6: **end for**
7: **end procedure**

Moreover, whenever a task finishes, the computing resource allocated to that task will be released and temporarily distributed to other tasks on the same server by using the DRC procedure.

**Scheduling Policy: Shortest Job First (SJF).** The OnTaPRA scheduler uses the Shortest Job First (SJF) scheduling policy to address all jobs in $Q_{wait}$. (Line 10 of Algorithm 3).

The SJF scheduling policy runs in iterations. In each iteration, the SJF policy begins with a test named **Minimum Available Resource (MAR)** which checks the total amount of available computing capacity in the data center (denoted by $C_{avai}^{total}$). If $C_{avai}^{total}$ is lower than a certain percentage (denoted by $Max\_Percent$) of the total computing capacity of all servers (denoted by $C_{DC}^{total}$), i.e., if the following condition is satisfied:

$$C_{avai}^{total} \leq Max\_Percent \cdot C_{DC}^{total}, \tag{56}$$

the test fails and the SJF policy stops scheduling all jobs current in $Q_{wait}$. The intuition here is that when the amount of available resource is small, a job may get little resource allocated and thereby have a extremely long completion time. In such cases, it may be a better choice to keep the job waiting until more resources become available.

If the SJF policy passes the MAR test, it sorts the jobs in $Q_{wait}$ in the decreasing order of their waiting time and calculates the JCT by using the TaPRA algorithm for each job $J_i$. If a job cannot be scheduled, that job is moved to a new waiting queue $Q_{wait}^{new}$. Subsequently, the job with the smallest JCT is executed according to its schedule and removed from $Q_{wait}$. In the following, SJF updates the available computing capacity of the servers and starts a new iteration. Once the waiting queue $Q_{wait}$ becomes empty, the SJF scheduling policy set the new waiting queue $Q_{wait}^{new}$ as current $Q_{wait}$ and finishes. Algorithm 4 shows the SJF scheduling policy.

**Work Conservation: Distribute Residual Capacity (DRC).** The DRC procedure iterates all servers. For each server $S_j$, the residual capacity of $S_j$ is proportionally distributed to all tasks running on $S_j$ according the load of those tasks.

Because the residual resource is not utilized by any job, by distributing these resources, we essentially avoid resource wastage, improve the system utilization, and further accelerate the job completion. On the other hand, such distribution of resource is temporary: Once new jobs arrive, the distributed resource is recollected and can be allocated to newly arrived jobs. Note that current virtualization techniques already support dynamic scaling of CPU and RAM for VMs [36]. Algorithm 5 shows the DRC procedure.

## 6 Performance Evaluation

### 6.1 Performance of the TaPRA Algorithm

In this section, we evaluate the TaPRA algorithm through offline simulations. In the following, we present our simulation setup, evaluation metrics, comparing algorithms and simulation results.

#### 6.1.1 Simulation Setup

In each single run of the simulation, we randomly generate an input job, a set of servers, and a set of scheduling constraints. The TaPRA algorithm is then called to schedule the job on the given servers.

**Job.** We randomly generate an input job with $N$ independent tasks. The load $Load_i$ of each task $T_i$ follows a uniform distribution in the range of $(0, 3600]$ seconds.

**Data Center.** For the data center, we use a FatTree [37] architecture. A $y$-array FatTree architecture contains $y$ pods. Each pod contains $y/2$ racks and each rack has $y/2$ servers. As a result, there are in a total of $y^3/4$ servers.

In the simulation, we modeled the available resource of each server $S_j$, i.e., $C_{s_j}$, by the number of virtual CPUs (vCPUs) that can be hosted by that server. Specifically, $C_{s_j}$ follows a uniform distribution between 0 and 10 vCPUs.

The efficiency matrices $\lambda_{ij}$ follows a uniform distribution in the range of $[0.1, 1]$. Because it is unlikely that the execution efficiency of a task is lower than 0.1 in a modern data center environment, we exclude the range $(0, 0.1)$ from the possible value of execution efficiency, following similar setups found in existing work [26].

**Resource Availability Constraints.** A $y$-array FatTree architecture contains $y$ pods. We generate one resource availability constraint for each pod.

Specifically, for pod $k$, we denote the set of servers in this pod by $\mathcal{S}_{Pod_k}$. For each server $S_j \in \mathcal{S}_{Pod_k}$, we set the coefficient $A_{jPod_k}$ as 1; for all other servers, we set $A_{jPod_k}$ as 0. Subsequently, we calculate the total available resources in pod $k$, denoted by $C_{Pod_k}$, using the following equation

$$C_{Pod_k} = \sum_{S_j \in \mathcal{S}_{Pod_k}} C_{s_j}. \tag{57}$$

We then generate the following constraint for pod $k$

$$\sum_{j=1}^{M} A_{jPod_k} \sum_{i=1}^{N} x_{ij} c_{t_i} \leq B_{Pod_k} = \beta_{pod} \cdot C_{Pod_k}, \tag{58}$$

where $\beta_{pod}$ is a constant belonging to $(0, 1]$. Using this way, we generate $y$ constraints corresponding to $y$ pods.

Following a similar approach, we generate $y^2/2 + 1$ constraints for the $y^2/2$ racks plus the whole data center in the $y$-array FatTree architecture. Therefore, we have $1 + y + y^2/2$ resources availability constraints for each single run of the simulation. In the simulations, we set $\beta_{DC}$, $\beta_{pod}$ and $\beta_{rack}$ as 0.2, 0.2 and 0.3 respectively.

**Energy Consumption Constraint.** To generate the maximum energy consumption $E_{MAX}$, we first calculate the total task load $Load_{total}$, which equals to $\sum_{i=1}^{N} Load_i$. Furthermore, we set the constant $\alpha_{VM_i}$ as 1 for each $VM_i$.

Because the execution efficiency is no larger than 1, according to Equation 9, the minimum possible energy consumption equals to $Load_{total}$. We then determine $E_{MAX}$ by using a uniform distribution in $[Load_{total}, 1.1 \cdot Load_{total}]$.

Each data point in our simulation results is an average of 50 simulations performed on an Intel 2.5 GHz processor.

#### 6.1.2 Evaluation Metrics

We use three metrics to evaluate our algorithms.

**JCT.** Because minimizing JCT for the input job is our objective, JCT is the most important metric.

**Total allocated resources (vCPUs).** This metric is the sum of the resources allocated to each task of the input job, which is also the total number of vCPUs allocated to the input job, according to our simulation setup. This metric shows how well an algorithm utilizes the available resources and is useful when analyzing the simulation results.

**Running time.** Running time of an algorithm is also important. It gives a sense of the scalability of that algorithm.

#### 6.1.3 Comparison Algorithms

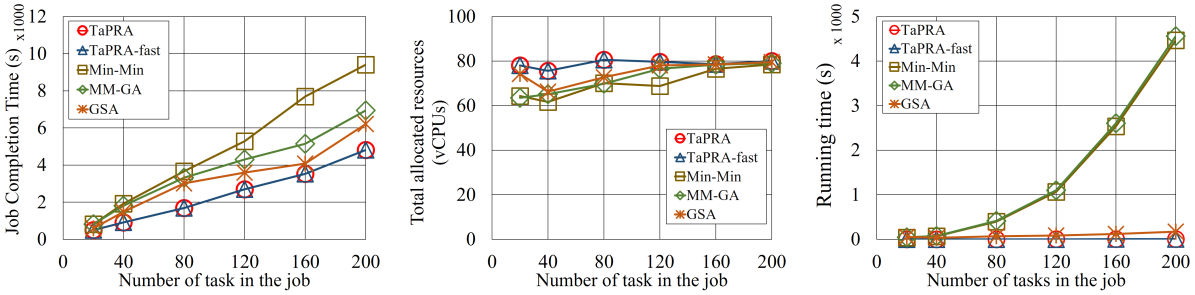
We compare our algorithms with three other algorithms.

**Min-Min.** The Min-Min algorithm is a classic scheduling algorithm. It runs in iterations. In each iteration, for each unplaced task, it calculates the expected JCT of placing that task on each server and adds the placement with the minimum expected JCT into a set $\mathcal{M}$. Subsequently, Min-Min selects and performs the placement with the smallest minimum expected JCT. It then starts a new iteration until all the tasks are placed.

**MM-GA.** Kumar *et al.* [10] proposed an improved genetic algorithm to schedule a set of independent tasks with the goal of minimizing the makespan. In that algorithm, the scheduling results of Max-Min and Min-Min are added into the initial population of the genetic algorithm. We use a modified version of this algorithm in which only the result of Min-Min is added into the initial population and we name this algorithm as MM-GA.

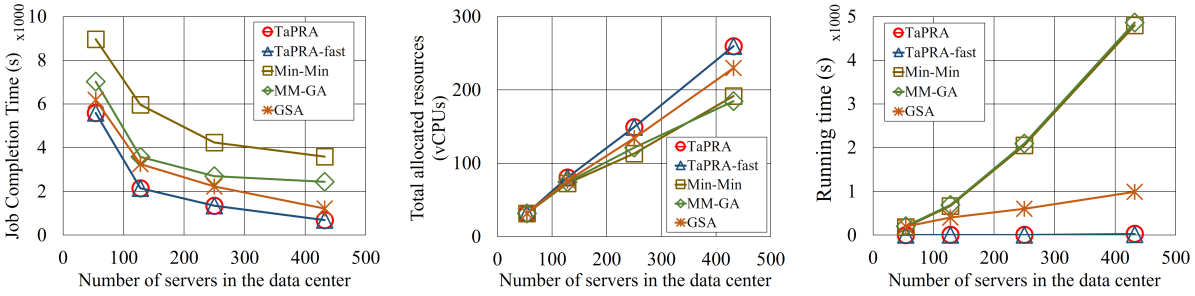
**GSA.** Gan *et al.* [11] proposed a genetic simulated annealing (GSA) algorithm which merged simulated annealing into a genetic algorithm. In each generation, the GSA algorithm generates a set of offspring using crossovers and mutations and then uses a simulated annealing procedure to further optimize those offspring.

Note that the above three algorithm are modified to consider the energy consumption constraint. Specifically, if a schedule generated by these algorithms violates the energy consumption constraint, the REC procedure is called to reduce the energy consumption for that schedule.




(a) JCT (b) Total allocated resources (c) Running time

Fig. 4. Performance of TaPRA when scheduling a job with different numbers of tasks on a FatTree data center with 128 servers.




(a) JCT (b) Total allocated resources (c) Running time

Fig. 5. Performance of TaPRA when scheduling a job with 100 tasks on a FatTree data center with different numbers of servers.

#### 6.1.4 *Evaluation Results of TaPRA and TaPRA-fast*

**Performance with Increasing Number of Tasks.** In this simulation, we study how TaPRA and TaPRA-fast performs as the number of tasks in the job $J$ increases from 40 to 200. We use a 8-array FatTree data center containing 128 servers.

Fig. 4(a) shows the JCT generated by each algorithm. While the JCT generally increases along with the expansion of the input job, TaPRA generates the smallest JCT. When the input job contains 200 tasks, the JCT of TaPRA is 100% smaller than that of Min-Min, 50% smaller than that of MM-GA, and 30% smaller than that of GSA. Meanwhile, TaPRA-fast generates almost the same JCT compared to TaPRA. Fig. 4(b) shows that the amount of resources allocated by each algorithm. We observe that when the number of tasks is small, TaPRA and TaPRA-fast allocate the largest amount of resources. When the number of tasks becomes large, all of the algorithms allocate similar amount of resources, however, TaPRA and TaPRA-fast generate much smaller JCT than Min-Min, MM-GA, and GSA. This observation shows that TaPRA and TaPRA-fast utilizes the available resources more efficiently. We attribute this to the analytical solution (16) proposed for the OptRA problem, which is used by TaPRA and TaPRA-fast to optimally allocate resource for any given task placement plan.

At last, Fig. 4(c) shows the algorithm running time. TaPRA and TaPRA-fast have much smaller running time than other algorithms. Specifically, when the job contains 100 tasks, the running time of TaPRA and TaPRA-fast is about 6 seconds, while that of GSA is 180 seconds and that of Min-Min and MM-GA is around 4500 seconds.

**Performance with Increasing Number of Servers.** In this simulation, we demonstrate how our algorithms perform as the number of servers in the FatTree data center increases from 54 to 432, i.e., the number of pods increases from 6 to 12. We fix the number of tasks in the job at 100.

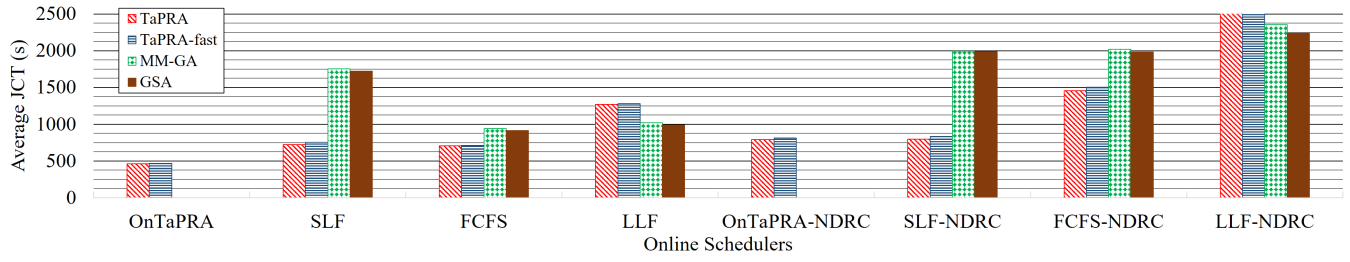
Fig. 5(a) shows the JCT of TaPRA and TaPRA-fast We can see that the JCT of TaPRA is similar to that of TaPRA-fast, 80% smaller than that of GSA, 260% smaller than that of MM-GA, and 430% smaller than that of Min-Min. Fig. 5(b) shows the total amount of resources allocated by each algorithm. We observe that when the size of the data center is small, all algorithms allocate similar amount of resources, but when the data center becomes larger (containing more servers), TaPRA and TaPRA-fast allocate more resources than other algorithms. When the data center contains 432 servers, TaPRA and TaPRA-fast allocate around 15% more resources than GSA and about 40% more resoureces than Min-Min and MM-GA. One possible reason is that TaPRA and TaPRA-fast are able to generate better task placement plan with more available resources to be allocated.

At last, Fig. 5(c) shows the algorithm running time. When the data center contains 12 pods (i.e., 432 servers), the running time of TaPRA is 33 seconds, while that of TaPRA-fast, GSA, Min-Min, and MM-GA is 15 seconds, 1000 seconds, 4700 seconds, and 4800 seconds respectively.

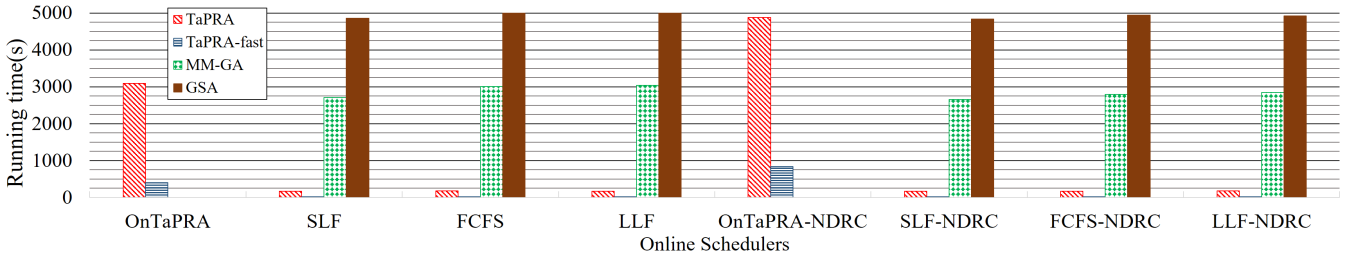
**Summary.** In offline simulations, we examine the performance of TaPRA and TaPRA-fast when scheduling a single input job. Generally, TaPRA and TaPRA-fast are able to place tasks in a way that more resources can be allocated and are able to utilize the allocated resources better. Benefiting from these properties, TaPRA and TaPRA-fast reduce the JCT by 40%-430% compared to the state-of-the-art algorithms. Moreover, the running time of these two algorithms are about 30 times faster than GSA and more than 200 times faster than Min-Min and GSA, which makes them more applicable in practice.

### 6.2 Performance of the OnTaPRA Scheduler

In this section, we examine the performance of the OnTaPRA scheduler through online simulations.




(a) JCT.




(b) Running time.

Fig. 6. Performance of the OnTaPRA scheduler.

#### 6.2.1 Simulation Setup

An online simulation starts from an initial state without any ongoing job. Subsequently, jobs start to arrive and the scheduler is called to schedule those jobs. Jobs stop arriving at a certain time. Once all jobs are scheduled, the online simulation finishes.

**Data Center.** A 6-array FatTree architecture with 54 servers is used in the simulation. The initial computing capacity of each server is 10 vCPUs.

**Jobs.** Jobs arrive at a rate following a Poisson distribution with $\mu = 0.15$/second and stop arriving at 3600 seconds.

Based on statistics of traces of workloads running on an 12000-server Google compute cell over a period approximately a month long, in May 2011 [38], we set the number of tasks in a job using a Weibull distribution [39] with scale parameter $A = 0.5$ and scale parameter $B = 0.8$. The maximum number of tasks in a job is set to 70. The task load follows a uniform distribution in $(0, 1000]$ seconds. The efficiency matrices $\lambda_{ij}$ are generated using the same approach used in the offline simulations.

**Constraints.** The resources availability constraints and energy consumption constraints of a job $J_i$ are generated whenever the scheduler is to schedule that job, using the same approach used in the offline simulations. Specially, we set $\beta_{DC}$, $\beta_{pod}$, and $\beta_{rack}$ as 0.2, 0.2, and 0.3 respectively.

Each data point in the results is an average of 10 simulations performed on an Intel 2.5GHz processor.

#### 6.2.2 Evaluation Metrics

**Average Online JCT.** The online JCT ($JCT_{on}$) of a job $J_i$ is the length of the time period between the arriving time and the completion time of that job. It composed of the total waiting time of this job (denoted by $JWT$) and the offline JCT ($JCT_{off}$) generated by the scheduling algorithm, i.e.,

$$JCT_{on} = JWT + JCT_{off}. \tag{59}$$

**Total Running Time.** Total running time is the running time of a scheduler from the beginning of a simulation to the end.

#### 6.2.3 Comparison Schedulers and Algorithms

We compare OnTaPRA with three other schedulers.

**FCFS.** The First Come First Serve (FCFS) scheduler is a classic scheduler which is still widely used in many scheduling systems because it is easily deployed and runs fast. In our simulations, whenever a job arrives the system, the FCFS scheduler address all waiting jobs in the decreasing order of their waiting time. If a job cannot be scheduled, it stays in the waiting queue; otherwise, it is executed according to the scheduling result. If the FCFS scheduler is DRC enabled, it then uses the DRC procedure to allocate residual capacity.

**Smallest Load First (SLF).** The SLF scheduler periodically address the jobs in $Q_{wait}$ in the increasing order of their total load. We believe that generally a job's completion time is positively related to its total load, i.e., a small total load means a short JCT. Therefore, the SLF scheduler may be a good approximation of the SJF scheduling policy. By comparing to the SLF scheduler, we can better examine the performance of the OnTaPRA scheduler. The SLF scheduler can use the DRC procedure to allocate residual capacity.

**Largest Load First (LLF).** The LLF scheduler is a scheduler that periodically addresses the jobs in the waiting queue in the descending order of their total load, which is opposite to the SLF scheduling policy used in the OnTaPRA scheduler. The LLF schedule can also use the DRC procedure to allocate residual capacity.

**Comparison Algorithms.** To make the comparison more comprehensive, each scheduler uses four different algorithms to schedule the arrived jobs, including TaPRA, TaPRA-fast, MM-GA and GSA. Min-Min is not included as MM-GA is guaranteed to be no worse than Min-Min.

#### 6.2.4 Evaluation Results of OnTaPRA

Fig. 6 shows the simulation results of each combination of schedulers and scheduling algorithms. Note that the MM-GA and GSA algorithms are not used by OnTaPRA, as their running time is relatively large; when using them, the running time of OnTaPRA is too large to be practical. To

examine the impact of the DRC procedure, we also perform simulations for each scheduler without the DRC procedure. In Fig. 6, the term "NDRC" appended after the scheduler name indicates that the scheduler is DRC disabled.

Fig. 6(a) shows the average JCT generated by each scheduler and Fig. 6(b) shows the running time of each scheduler. Based on the results, we have several observations:

**Impact of Schedulers.** Generally, the OnTaPRA scheduler generates the smallest average JCT. In the DRC enabled simulations, when using TaPRA or TaPRA-fast, the average JCT generated by OnTaPRA is about 60% smaller compared to the SLF and FCFS scheduler and about 170% smaller than the LLF scehduler. When compared to other schedulers using MM-GA or GSA, OnTaPRA reduces the average JCT by 100%-280%. In the DRC disabled simulations, when using TaPRA or TaPRA-fast, OnTaPRA generates 5%, 80%, and 234% smaller average JCT compared to SLF, FCFS, and LLF respectively.

We can also see that when using TaPRA or TaPRA-fast, while SLF performs similar to FCFS and about 80% better than LLF in DRC enabled simulations, it performs 80% better than FCFS and 225% better than LLF. But when using MM-GA or GSA, SLF performs worse than FCFS and LLF in DRC enabled simulations. These results show that the performance of SLF can be impacted by the DCR procedure and the scheduling algorithm. Whereas, the OnTaPRA scheduler always performs the best.

**Impact of Scheduling Algorithm.** Generally, TaPRA and TaPRA-fast show better performance than MM-GA and GSA, while TaPRA is about 5% better than TaPRA-fast. In DCR enabled simulations, TaPRA and TaPRA-fast reduces the average JCT by nearly 80% and 140% compared to MM-GA and GSA in the SLF and FCFS scheduler respectively, while in the LLF scheduler, TaPRA and TaPRA-fast performs actually worse than MM-GA and GSA. We attribute this to the better performance of TaPRA and TaPRA-fast on allocating resources because of which the LLF scheduler allocates most of the available resources to the larger jobs and therefore increases the completion time of the jobs with smaller load. We have similar observation in DRC disabled observations: TaPRA and TaPRA-fast performs 40%-150% better than MM-GA and GSA in the SLF and FCFS schedulers but worse than MM-GA and GSA in the LLF scheduler.

However, we can also see that the LLF scheduler performs worst among all schedulers; whereas, the OnTaPRA scheduler with TaPRA or TaPRA-fast performs the best.

**Impact of the Distribute Residual Capacity (DRC) Procedure.** The DRC procedure temporarily distributes the residual capacity of servers to the running tasks to accelerate the completion of running jobs. We can see that the DCR procedure has signification impact on the average JCT. Without the DCR procedure the average JCT generated by OnTaPRA, SLF, FCFS, and LLF is increased by 70%, 15%, 110%, and 120% respectively.

**Scheduler Running Time.** Fig. 6(b) shows the overall running time of each scheduler. Generally, the running time of using TaPRA and TaPRA-fast is much smaller than using MM-GA and GSA, which confirms our observation in offline simulations. Meanwhile, when using TaPRA or TaPRA-fast, the running time of OnTaPRA is larger than other schedulers, because of the higher complexity of OnTaPRA. Furthermore, the running time of using TaPRA-fast is about 10 times faster than using TaPRA in all schedulers.

**Summary.** In online simulations, we demonstrate the performance of the OnTaPRA scheduler and the TaPRA/TaPRA-fast algorithms. The results show that: (a) the OnTaPRA scheduler with TaPRA/TaPRA-fast has the best performance: it reduces the average JCT to 60%-280% compared by existing schedulers; (b) the proposed DRC procedure has great impact on the average JCT: without this procedure, the average JCT can be increased by up to 120%; (c) while TaPRA-fast performs 5% worse than TaPRA, when using TaPRA-fast, the running time of OnTaPRA is 10 times smaller than using TaPRA, which makes OnTaPRA+TaPRA-fast an very applicable choice in practice.

## 7 Conclusion

In this paper, we focused on the problem of scheduling embarrassingly parallel jobs in cloud, in which there is a need to determine the task placement plan and the resource allocation plan for jobs composed of independent tasks with the goal of minimizing the Job Completion Time (JCT). We first studied how to optimally allocate resources with predetermined task placement and proposed an analytical solution. In the following, we formulate the problem of scheduling a single job (SJS) as a NLMIP problem and present an relaxation with an equivalent Linear Programming problem. We further propose an algorithm named TaPRA and its simplified version: TaPRA-fast that solve the SJS problem. At last, to address multiple jobs in online scheduling, we propose an online scheduler named OnTaPRA.

We evaluated the performance of the TaPRA and TaPRA-fast algorithms and the OnTaPRA scheduler by comparing them with the state-of-the-art algorithms and schedulers in offline and online simulations. The simulation results show that: (a) TaPRA and TaPRA-fast reduce the JCT by 40%-430% compared to the state-of-the-art algorithms and their running time is more than 30 times smaller. (b) The OnTaPRA scheduler when using TaPRA/TaPRA-fast reduces the average JCT by 60%-280% compared to exsiting schedulers. (c) TaPRA-fast can be 10 times faster than TaPRA with around 5% performance degradation compared to TaPRA, which makes the use of TaPRA-fast very applicable in practice.

## References


[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[3] T. Gunarathne, T.-L. Wu, J. Y. Choi, S.-H. Bae, and J. Qiu, "Cloud computing paradigms for pleasingly parallel biomedical applications," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 17, pp. 2338–2354, 2011.

[4] T. Mathew, K. C. Sekaran, and J. Jose, "Study and analysis of various task scheduling algorithms in the cloud computing environment," in *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*. IEEE, 2014, pp. 658–664.

[5] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 63, 2015.

[6] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, pp. 2687–2699, 2015.

[7] S. H. Adil, K. Raza, U. Ahmed, S. S. A. Ali, and M. Hashmani, "Cloud task scheduling using nature inspired meta-heuristic algorithm," in *2015 International Conference on Open Source Systems & Technologies (ICOSST)*. IEEE, 2015, pp. 158–164.

[8] Z. Tang, L. Jiang, J. Zhou, K. Li, and K. Li, "A self-adaptive scheduling algorithm for reduce start time," *Future Generation Computer Systems*, vol. 43, pp. 51–60, 2015.

[9] C.-W. Tsai, W.-C. Huang, M.-H. Chiang, M.-C. Chiang, and C.-S. Yang, "A hyper-heuristic scheduling algorithm for cloud," *Cloud Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 236–250, 2014.

[10] P. Kumar and A. Verma, "Independent task scheduling in cloud computing by improved genetic algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 5, 2012.

[11] G. Guo-ning, H. Ting-lei, and G. Shuai, "Genetic simulated annealing algorithm for task scheduling based on cloud computing environment," in *2010 International Conference on Intelligent Computing and Integrated Systems*, 2010.

[12] J. G. Koomey *et al.*, "Estimating total power consumption by servers in the us and the world," 2007.

[13] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *Cloud Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 168–180, 2014.

[14] M. Liaqat, S. Ninoriya, J. Shuja, R. W. Ahmad, and A. Gani, "Virtual machine migration enabled cloud resource management: A challenging task," *arXiv preprint arXiv:1601.03854*, 2016.

[15] G. Han, W. Que, G. Jia, and L. Shu, "An efficient virtual machine consolidation scheme for multimedia cloud computing," *Sensors*, vol. 16, no. 2, p. 246, 2016.

[16] Y. Mhedheb, F. Jrad, J. Tao, J. Zhao, J. Kołodziej, and A. Streit, "Load and thermal-aware vm scheduling on the cloud," in *Algorithms and Architectures for Parallel Processing*. Springer, 2013, pp. 101–114.

[17] Y. Shen, Z. Bao, X. Qin, and J. Shen, "Adaptive task scheduling strategy in cloud: when energy consumption meets performance guarantee," *World Wide Web*, pp. 1–19, 2016.

[18] Q. Zhao, C. Xiong, C. Yu, C. Zhang, and X. Zhao, "A new energy-aware task scheduling method for data-intensive applications in the cloud," *Journal of Network and Computer Applications*, vol. 59, pp. 14–27, 2016.

[19] S. Hosseinimotlagh, F. Khunjush, and R. Samadzadeh, "Seats: smart energy-aware task scheduling in real-time cloud computing," *The Journal of Supercomputing*, vol. 71, no. 1, pp. 45–66, 2015.

[20] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters," *Future Generation Computer Systems*, vol. 37, pp. 141–147, 2014.

[21] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 6, pp. 1107–1117, 2013.

[22] S. Shin, Y. Kim, and S. Lee, "Deadline-guaranteed scheduling algorithm with improved resource utilization for cloud computing," in *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE, 2015, pp. 814–819.

[23] D. Liu and N. Han, "An energy-efficient task scheduler in virtualized cloud platforms," *International Journal of Grid and Distributed Computing*, vol. 7, no. 3, pp. 123–134, 2014.

[24] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on iaas cloud systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 5, pp. 666–677, 2012.

[25] Y. Ge and G. Wei, "Ga-based task scheduler for the cloud computing systems," in *Web Information Systems and Mining (WISM), 2010 International Conference on*, vol. 2. IEEE, 2010, pp. 181–186.

[26] D. Li and J. Wu, "Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 3, pp. 810–823, March 2015.

[27] A. Bohra and V. Chaudhary, "Vmeter: Power modelling for virtualized clouds," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, April 2010, pp. 1–8.

[28] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10, 2010, pp. 39–50.

[29] 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.

[30] C. Gu, H. Huang, and X. Jia, "Power metering for virtual machine in cloud computing-challenges and opportunities," *Access, IEEE*, vol. 2, pp. 1106–1116, 2014.

[31] N. Kim, J. Cho, and E. Seo, "Energy-credit scheduler: an energy-aware virtual machine scheduler for cloud systems," *Future Generation Computer Systems*, vol. 32, pp. 128–137, 2014.

[32] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[33] B. Veeravalli, D. Ghose, V. Mani, and T. G. Robertazzi, "Scheduling divisible loads in parallel and distributed systems," *Los Almitos: IEEE Computer Society Press, California*, 1996.

[34] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, vol. 36, no. 5, pp. 63–68, 2003.

[35] Z. Zhang and T. G. Robertazzi, "Scheduling divisible loads in gaussian, mesh and torus network of processors," *IEEE Transactions on Computers*.

[36] "Dynamic scaling of cpu and ram for vms in apache cloudstack," https://cwiki.apache.org/confluence/display/CLOUDSTACK/Dynamic+scaling+of+CPU+and+RAM.

[37] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[38] J. L. Hellerstein, W. Cirne, and J. Wilkes, "Google cluster data," *Google research blog, Jan*, 2010.

[39] L. M. Leemis, *Reliability: probabilistic models and statistical methods*. Prentice-Hall, Inc., 1995.



**Li Shi** received his B.E. degree in electrical and computer engineering from Shanghai Jiao Tong University, Shanghai, China, in 2010. He is currently pursing his Ph.D. degree in computer engineering at Stony Brook University, Stony Brook, NY. His research interests include task scheduling and resource allocation in data centers, cloud computing, data center network, software defined network, etc.

**Zhemin Zhang** received the Ph.D from Stony Brook University, Stony Brook, NY, USA, in 2014, the B.E. in computer science and the B.Sci. in electrical engineering from Huazhong University of Science and Technology, Wuhan, China, in 2009. He is presently an Assistant Professor in the Dept. of Computer Science at Xiamen University, Xiamen, China. His research interests include parallel and distributed processing, data center network, high speed optical network, on-chip network, etc.

**Thomas G. Robertazzi** received the Ph.D from Princeton University, Princeton, NJ, in 1981 and the B.E.E. from the Cooper Union, New York, NY in 1977. He is presently a Professor in the Dept. of Electrical and Computer Engineering at Stony Brook University, Stony Brook N.Y. He has published extensively in the areas of parallel processing scheduling, telecommunications and performance evaluation. Prof. Robertazzi has also authored, co-authored or edited six books in the areas of networking, performance evaluation, scheduling and network planning. He is a Fellow of the IEEE and since 2008 co-chair of the Stony Brook University Senate Research Committee.