Service Reliability in an HC: Considering From the Perspective of Scheduling With Load-Dependent Machine Reliability

Chubo Liu, Kenli Li, *Senior Member, IEEE*, Jie Liang, and Keqin Li, *Fellow, IEEE*

***Abstract*—Considering the fact that a server is more likely to fail if it is highly loaded, in this paper, we involve load impacts on service reliability in a heterogeneous cluster, where servers have load-dependent reliabilities and jobs have resource and execution interval demands. Specifically, each server is specified by a resource capacity and a workload limitation, i.e., the server is expected to perform reliably if its workload is less than the workload limitation. There is also a set of jobs needing to be executed on these servers. Each job is associated with a resource demand and an execution interval, i.e., the time interval that the job is planned to be executed. Our goal is to find a reliable schedule of jobs to servers such that all servers' workload limitations are satisfied. The problem is proved to be strongly NP-complete, which implies that there is not even a pseudo-polynomial time algorithm. Hence, we try our best to find a reliable schedule. To solve the problem, we define a stereogram (specifically defined as a bipartite stereogram) and develop a bipartite stereogram matching-based scheduling method (BSMSM), which combines the matching of our constructed bipartite stereogram with reliable scheduling. Some theoretical results are also derived in this paper. We perform extensive random experiments and the results show that BSMSM can find reliable schedules, i.e., the workloads on all servers can be maintained to be less than or equal to the servers' workload limitations, to a large extent.**

***Index Terms*—Bipartite stereogram, heterogeneous cluster (HC), reliable scheduling, workload limitation.**

I. Introduction

A. Motivation

DURING recent years, there is a rapidly increasing number of cloud infrastructure, platform, and service centers (such as Amazon, Microsoft Azure, Rackspace, etc.) [1]. Due to its cost-effectiveness, maintenance-free convenience, and other advantages, many companies or users have placed their applications out to cloud centers, and to support various service demands, a service provider has to equip more and more computing servers which form a huge heterogeneous cluster (HC) [2]. With tens of thousands of host servers in an HC, it is difficult to ensure that all host servers always work well. Hence, a statistically rare failure event may become common in a cloud center, which seriously impacts the performance and cost [3], [4]. For example, an unreliable HC with a higher probability of server failures inevitably results in more interruption of running virtual machines, which implies a reduction in performance metric. Subsequent recovery or redoing the work also incurs additional energy cost. Therefore, the issue of how to enhance service reliability has become a critical problem.

Reliability is defined as the ability of an item to perform a required function under stated conditions for a stated time period in standard TL9000 [6], [7]. For server reliability, specifically, it can be understood as the probability of the server to perform normally for a given period of time. When considering service reliability in an HC, server reliability is one of the most important factors that should be taken into account [8]. The reason lies in that when a host server crashes, all applications or virtual machines on it will go down. The services delivered by the virtual machines will be interrupted or even need to be re-run from the beginning, which can result in serious performance degradation and cost increase. On the other hand, it influences the appeal to users or potential users in the market. Specifically, due to performance degradation (e.g., long task response time), a user may refuse to use the provided services or choose another service provider, which impacts the long-term revenue of the service provider.

It is well-known that as a server approaches a high level of workload utilization, degradation in performance occurs (e.g., the average running time of an OpenMP program delays as the increase of its parallelism degree [9]–[11]). Many works also establish that workload utilization impacts a server's reliability [12], [13]. For example, in [5], Deng *et al.* established that CPU temperature tends to linearly increase with its utilization [see Fig. 1(a)]. They obtained the results by carefully controlling a server to run SPEC 2006 benchmark [14] and using lm-sensors [15] to record the corresponding temperature changes. Since the failure rates caused by many hard errors [e.g., electromigration

Manuscript received December 10, 2017; revised May 15, 2018 and September 29, 2018; accepted March 22, 2019. Date of publication May 3, 2019; date of current version May 28, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61702170, Grant 61802032, and Grant 61772182, in part by the Key Program of National Natural Science Foundation of China under Grant 61432005, and in part by the National Outstanding Youth Science Program of National Natural Science Foundation of China under Grant 61625202. Associate Editor: C.-Y. Huang. *(Corresponding authors: Kenli Li; Keqin Li.)*

C. Liu, K. Li, and J. Liang are with the College of Information Science and Engineering, Hunan University, Hunan 410082, China, and also with the National Supercomputing Center in Changsha, Hunan 410082, China (e-mail: liuchubo@hnu.edu.cn; lkl@hnu.edu.cn; lxj@hnu.edu.cn).

K. Li is with the College of Information Science and Engineering, Hunan University, Hunan 410082, China, with the National Supercomputing Center in Changsha, Hunan 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2019.2908269

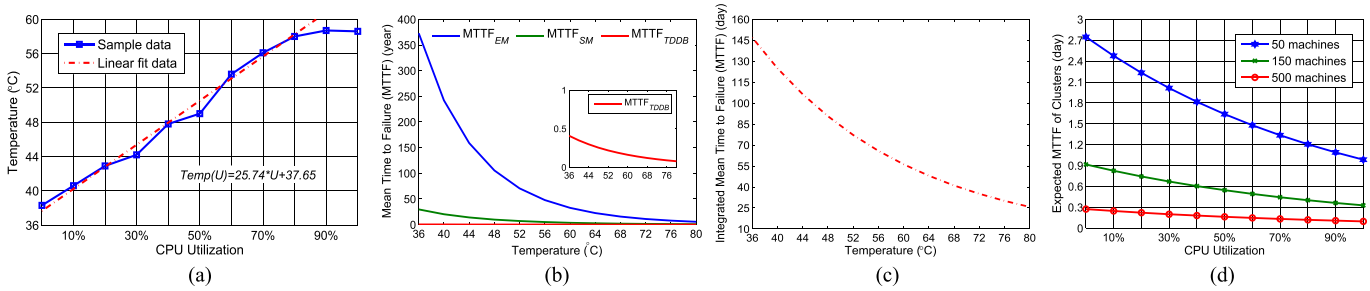

Fig. 1. Illustration of the relationship between CPU utilization and MTTF of a server (cluster). (a) Relationship between CPU temperature and utilization [5]. (b) MTTF caused by three common hard errors (EM, SM, and TDDB). (c) Integrated MTTF caused by EM, SM, and TDDB. (d) Cluster reliability.

(EM), stress migration (SM), time-dependent dielectric breakdown (TDDB)] exponentially increase with the increase of temperature [16], [17], the corresponding reliability (mean-time-to-failure, MTTF) caused by each of these errors tends to exponentially decrease with temperature increment [see Fig. 1(b)]. Due to the linear relationship between utilization and temperature, the reliability of a server caused by EM, SM, or TDDB also tends to decrease with utilization increment. In Fig. 1(c), we plot the shape of the integrated reliability of a server caused by EM, SM, and TDDB. Even though the reliability of a server can be relatively high, when involving tens of hundreds of servers (a cluster), the corresponding reliability can be significantly low [see Fig. 1(d)].

Therefore, the reliability of a server can be impacted by its workload (e.g., CPU utilization) and server workload should be involved when addressing reliability. This conflicts with most previous assumptions that a server maintains a constant reliability level [18], [19]. In addition, many existing works for service reliability are based on replication strategies [20] or rollback recovery relying on checkpointing protocols [21], [22]. Rare works are done from the perspective of servers' workload impacts. Therefore, in this paper, we involve the load impact on the reliability of an HC, in which we associate a workload limitation for each server, i.e., a server is assumed to attain a desired reliability level if its workload is less than its workload limitation. Furthermore, we consider from the perspective of scheduling. Specifically, we try our best to find a schedule of jobs to servers such that all sever load constraints are satisfied.

B. Our Contributions

Considering the fact that a server is more likely to fail if it is highly loaded, in this paper, we involve load impacts on service reliability in an HC, where servers have load-dependent reliabilities and jobs have resource and execution interval demands. Specifically, there are some heterogeneous servers. Each server is specified by a resource capacity and a workload limitation, i.e., the server is expected to perform reliably if its workload is less than the workload limitation. There is also a set of jobs needing to be executed on these servers. Each job is associated with a resource demand and an execution interval, i.e., the time interval that the job is planned to be executed. Our goal is to find a reliable schedule of jobs to servers such that all servers' workload limitations are satisfied. The problem is proved to be strongly NP-complete, which implies that there is not even a pseudo-polynomial time algorithm. Hence, we try our best to find a reliable schedule.

The main contributions and differences of this paper are listed as follows.

1) We involve load impacts on reliability and consider from the perspective of scheduling with machine load limitations to maintain reliability.
2) We accordingly propose a method called bipartite stereogram matching-based scheduling method (BSMSM), in which we consider job–machine mapping as an extended matching problem.
3) A theoretical conclusion is derived from our bipartite stereogram matching-based method (see Theorem III.2), which provides theoretical guarantees for our method to some extent.

We perform extensive random experiments. The results show that BSMSM can find appropriate schedules to a large extent.

C. Related Work

Enhancing the reliability of services is an important aspect of service providers and has received considerable attention from the research community [23]. The main solutions for service reliability are based on replication strategies [19], [20], [24], [25] and rollback recovery relying on checkpointing protocols [21], [22].

Replication is based on the exploitation of redundancy and is used in many fields. One-to-one and one-to-many standbys are two well-known mechanisms. In [26], Xu *et al.* tried to map each primary virtual machine (VM) to a backup VM. A primary VM and its mapping backup node form a survivable group. A task can be completed in time if at least one VM in the survivable group works well. The work takes the bandwidth reservation into account when solving the mapping problem. In this regard, an optimal algorithm that maps a survivable group to the physical data center was proposed [26]. In [27], Usman *et al.* introduced a data semantics and data encoding-based data replication strategy called semantic data replication (SDR), which focuses on high operation availabilities with low costs. In [28], Wang *et al.* proposed replicating bandwidth-intensive social contents in a device-to-device manner. Based on large-scale

measurement studies on social content propagation and user mobility patterns in edge-network regions, they observed that their proposed device-to-device replication strategy can significantly help users download social contents from neighboring peers.

Checkpointing is another widely used mechanism for reliability that functions by periodically saving the execution state as an image file. However, usually, data centers have limited network resources and may readily become congested when a huge number of checkpoint image files are transferred. Attempting to avoid this problem, Zhang *et al.* [29] presented a theoretical delta-checkpoint approach in which the base system only needs to be saved once the first checkpoint completes and subsequent checkpoint images only contain the incrementally modified pages. A theoretical delta-checkpoint approach was implemented by Goiri *et al.* in [30]. A further reduction in network resource consumption was developed by Limrungsi *et al.* [31], who proposed a peer-to-peer checkpoint approach in which the checkpoint images are stored on the neighboring host servers. If the storage server is located in the same pod as the service providing server, it is unnecessary to transfer the checkpoint images via core switches. In [32], Levitin *et al.* tried to formulate and solve optimal dynamic checkpointing policy problems, as well as an integrated optimization problem that finds the optimal combination of checkpointing policy and element activation sequence. They also demonstrated the advantages of using the dynamic checkpointing policy over fixed even checkpoints through examples.

Different from the abovementioned works, we involve workload impacts when addressing reliability. In addition, we consider the reliability of a cluster from the perspective of scheduling. Specifically, we try to guarantee that for each server, its workload is maintained below a threshold to attain a desirable reliability. Obviously, under this scheme, the reliability of a cluster can also be maintained.

The rest of this paper is organized as follows. Section II-B presents the preliminary notations and formulates our reliable scheduling problem. In Section III, we construct our bipartite stereogram and develop a BSMSM. Many analyses and discussions are also shown in this section. In Section IV, we perform extensive random experiments and show the results. The conclusions are presented in Section V.

II. System Model and Problem Definition

A. Model Construction

Since the reliability of a server tends to exponentially decrease with temperature (workload), we can find a tradeoff between them to maintain relatively high reliable execution by controlling workload on the server (see Fig. 2). For a cluster, if we guarantee the reliability of each server, the corresponding reliability of the cluster can be maintained (see Fig. 3).

Therefore, in our model, we associate a workload threshold for each server and assume that if its workload is less than the threshold, the server can attain a desirable reliability level. By appropriate scheduling, we try to guarantee that all server workload thresholds maintain the reliability of the cluster. We formally formulate our problem in the next section.

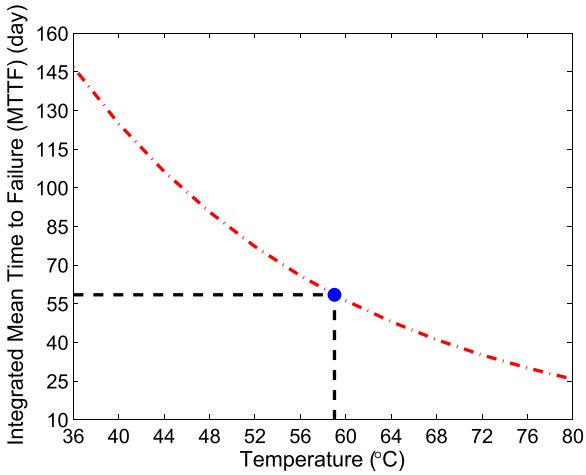

Fig. 2. Tradeoff between reliability and temperature (workload).

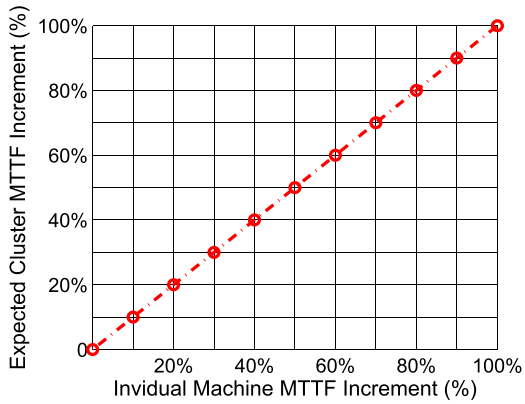

Fig. 3. Impact of each server's reliability improvement on a cluster's reliability.

B. Problem Definition

An HC is equipped with m heterogeneous machines, which are represented by $\mathcal{M} = \{1, \ldots, m\}$. Each machine j ($j \in \mathcal{M}$) is associated with a load capacity C_j and a workload limitation $\bar{u}_j$ ($\bar{u}_j < 1$), i.e., machine j is expected to perform reliably if its workload is less than or equal to $\bar{u}_j$. In this paper, we consider time to be uniformly divided into discrete intervals ranging from 1 to T. Each interval is referred to as a time slot and the set of time slots is denoted as $\mathcal{T} = \{1, \ldots, T\}$. There is a set of jobs $\mathcal{N} = \{1, \ldots, n\}$ needing to be scheduled on the machines in $\mathcal{M}$. Each job i ($i \in \mathcal{N}$) is associated with a resource requirement r_i and an execution interval $\mathcal{I}_i = [s_i, e_i]$, where s_i and e_i ($s_i, e_i \in \mathcal{T}$) are the start-time and end-time slots of job i, respectively. Our goal is to find a scheduling strategy for jobs in $\mathcal{N}$ such that the workloads on all servers are less than or equal to the servers' workload limitations throughout the time slots (see Fig. 4).

We formalize our scheduling problem as follows. As mentioned earlier, each job i ($i \in \mathcal{N}$) is associated with an execution interval $\mathcal{I}_i = [s_i, e_i]$. We say that job i is active at a time slot

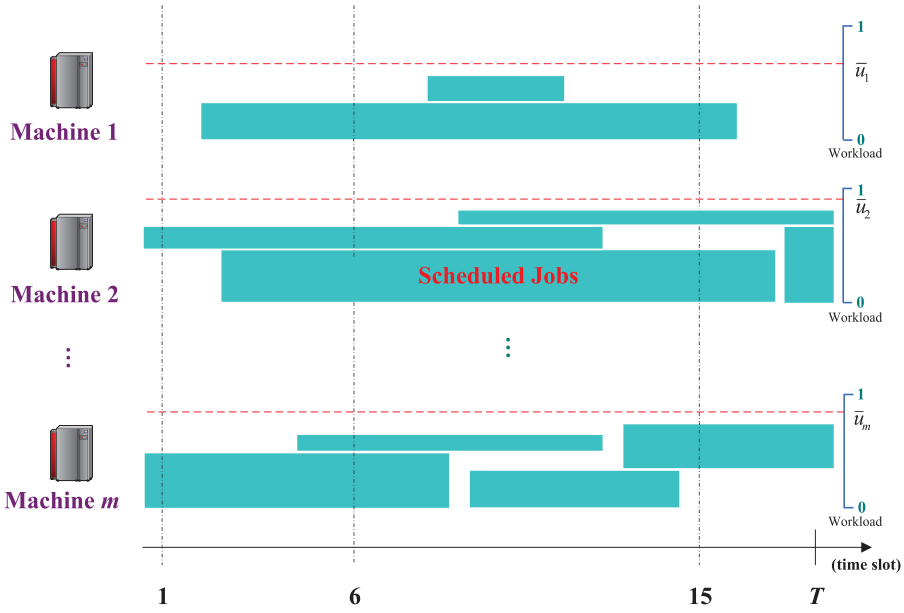

Fig. 4. Scheduling model with load limitation.

$t \in \mathcal{T}$ if $t \in \mathcal{I}_i$. If we denote $\mathcal{A}(t)$ as the set of all jobs active at time slot t ($t \in \mathcal{T}$), we can obtain

$$\mathcal{A}(t) = \left\{ i | \, t \in \mathcal{I}_i \text{ and } i \in \mathcal{N} \right\}. \tag{1}$$

Let Φ_j ($\Phi_j \subseteq \mathcal{N}$) be the set of jobs scheduled on machine j ($j \in \mathcal{M}$). Then, the workload utilization on machine j during time slot t ($t \in \mathcal{T}$) is expressed as

$$u_j\left(\Phi_j, t\right) = \frac{1}{C_j} \sum_{i \in \Phi_j \cap \mathcal{A}(t)} r_i. \tag{2}$$

Our goal is to find a scheduling strategy $\mathbf{\Phi} = \left(\Phi_j\right)_{j \in \mathcal{M}}$ such that the workload limitations of all machines are satisfied as much as possible, i.e., find a strategy to the following reliable scheduling (RELIABLE-SCHED) problem:

$$\forall j \in \mathcal{M} \;\; \forall t \in \mathcal{T} \quad u_j\left(\Phi_j, t\right) \leq \bar{u}_j \tag{3}$$

where $\mathbf{\Phi}$ satisfies $\bigcup_{j \in \mathcal{M}} \Phi_j = \mathcal{N}$ and $\Phi_j \cap \Phi_k = \emptyset$ for all $j, k \in \mathcal{M}$.

Theorem II.1: RELIABLE-SCHED is NP-complete in the strong sense.

Proof: In order to prove the strong NP-completeness of RELIABLE-SCHED, we will use a reduction from the Bin Packing problem [33].

Bin Packing (*Decision version*): Given a set of bins $B_1, B_2, \ldots, B_m$ with the same size V and a list of n items with sizes $s_1, s_2, \ldots, s_n$ to pack, find an m-partition $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_m$ of the set $\{1, 2, \ldots, n\}$ such that

$$\forall k \in \{1, \ldots, m\}, \sum_{i \in \mathcal{S}_k} a_i \leq V.$$

Let us consider the a special instance of RELIABLE-SCHED, in which there are n jobs $J_1, J_2, \ldots, J_n$ to be executed on m machines $M_1, M_2, \ldots, M_m$. There is only one time slot and all the jobs are active in that time slot. Let a_i denote the resource demand of J_i. All servers are with resource capacity V and workload limitation one, i.e., the machines are homogeneous. Then, the Bin Packing problem corresponds to this special instance of RELIABLE-SCHED.

Therefore, the abovementioned special instance of RELIABLE-SCHED and thus, RELIABLE-SCHED is strongly NP-complete. This achieves the proof of strong NP-completeness of RELIABLE-SCHED. ■

III. BIPARTITE STEREOGRAM MATCHING-BASED SCHEDULING ALGORITHM

In this section, we present a reliable scheduling algorithm called BSMSM to RELIABLE-SCHED. Before presenting BSMSM, we first find a feasible fractional solution to RELIABLE-SCHED and construct a stereogram (specifically defined as bipartite stereogram), which is the basis of BSMSM.

A. Problem Transformation for a Feasible Fractional Solution

We first try to transform RELIABLE-SCHED to another much simpler problem so that we can find a feasible fractional solution to RELIABLE-SCHED.

Denote $R_i(t)$ as the resource requirement of job i ($i \in \mathcal{N}$) at time slot t ($t \in \mathcal{T}$). Then

$$R_i(t) = \begin{cases} r_i, & \text{if } t \in \mathcal{I}_i \\ 0, & \text{if } t \notin \mathcal{I}_i \end{cases} \tag{4}$$

where $\mathcal{I}_i$ is the associated execution interval of job i. For each job $i \in \mathcal{N}$ and each machine $j \in \mathcal{M}$, we associate a variable $x_{i,j}$, which represents whether job i is allocated on machine j. Thus, $x_{i,j}$ only takes on integer value 0 (not allocated) or 1 (allocated), and the workload utilization on machine j ($j \in \mathcal{M}$)

at time slot t ($t \in \mathcal{T}$) can be rewritten as

$$u_j(\boldsymbol{x}, t) = \frac{1}{C_j} \sum_{i \in \mathcal{N}} x_{i,j} R_i(t) \tag{5}$$

where $\boldsymbol{x} = (x_{i,j})_{i \in \mathcal{N}, j \in \mathcal{M}}$ is the scheduling indicator for all jobs, and satisfies the constraint $\sum_{j \in \mathcal{M}} x_{i,j} = 1$ for all $i \in \mathcal{N}$.

Notice that, our objective is to try to guarantee that the workload on each machine does not exceed its threshold throughout the time horizon, i.e., $\forall j \in \mathcal{M}\, \forall t \in \mathcal{T}$

$$u_j(\Phi_j, t) \leq \bar{u}_j. \tag{6}$$

However, due to the strong NP-completeness, it is impracticable to obtain such a solution in polynomial time under P $\neq$ NP. Hence, we first try to obtain a feasible fractional solution, and based on the fractional solution, we consider the mapping of jobs to machines from a matching perspective.

Denote $u_j^{\max}(\boldsymbol{x}, t)$ as the maximal workload of machine j ($j \in \mathcal{M}$) among all the T time slots under job allocation strategy $\boldsymbol{x}$, i.e.,

$$u_j^{\max}(\boldsymbol{x}) = \max_{t \in \mathcal{T}} u_j(\boldsymbol{x}, t). \tag{7}$$

To obtain a feasible fractional solution, we incorporate an objective

$$U(\boldsymbol{x}) = \prod_{j \in \mathcal{M}} \left(\bar{u}_j - u_j^{\max}(\boldsymbol{x})\right). \tag{8}$$

The reason why we design such an objective to obtain a fractional solution is that besides satisfying the constraint $\forall j \in \mathcal{M}\, \forall t \in \mathcal{T}$, $u_j(\Phi_j, t) \leq \bar{u}_j$, it can mitigate the workloads among all machines due to the fact that under ideal situation, $U(\boldsymbol{x})$ is maximized when the values $\left(\bar{u}_j - u_j^{\max}(\boldsymbol{x})\right)$ of all machines are equal. We first ignore the integer constraint of $\boldsymbol{x}$ and try to

$$\text{maximize } U(\boldsymbol{x}) \tag{9}$$

subject to the job allocation constraint and machine workload limitation constraint, i.e., we try to solve the following optimization problem (OPT-U):

$$\text{maximize } U(\boldsymbol{x}) = \prod_{j \in \mathcal{M}} \left(\bar{u}_j - u_j^{\max}(\boldsymbol{x})\right) \tag{10}$$

subject to

$$\begin{cases} \sum_{j \in \mathcal{M}} x_{i,j} = 1\, \forall i \in \mathcal{N} \\ u_j^{\max}(\boldsymbol{x}) < \bar{u}_j\, \forall j \in \mathcal{M} \\ x_{i,j} \geq 0\, \forall i \in \mathcal{N}\, \forall j \in \mathcal{M} \end{cases}. \tag{11}$$

Directly solving OPT-U is very hard, because each function $u_j^{\max}(\boldsymbol{x})$ ($j \in \mathcal{M}$) is discontinuous and quite irregular. Hence, to obtain an appropriate solution to OPT-U, we further transform it into another simpler problem. The details are proceeded as follows.

We simplify OPT-U by assuming that the allocation strategies of a job on all machines are the same, i.e., $\forall j \in \mathcal{M}$

$$x_{1,j} = x_{2,j} = \cdots = x_{n,j} = x_j^0. \tag{12}$$

Denote $\boldsymbol{x}^0$ as the new allocation strategy, i.e., $\boldsymbol{x}^0 = (x_j^0)_{j \in \mathcal{M}}$, and satisfies $\sum_{j \in \mathcal{M}} x_j^0 = 1$. Then, we can rewrite $u_j(\boldsymbol{x}, t)$ as

$$u_j(\boldsymbol{x}^0, t) = \frac{x_j^0}{C_j} \sum_{i \in \mathcal{N}} R_i(t) \tag{13}$$

and $u_j^{\max}(\boldsymbol{x})$ as

$$u_j^{\max}(\boldsymbol{x}) = u_j(\boldsymbol{x}^0, \hat{t}) \tag{14}$$

where $\hat{t}$ ($\hat{t} \in \mathcal{T}$) is the time slot with maximal $\sum_{i \in \mathcal{N}} R_i(t)$, i.e.,

$$\hat{t} \in \arg\max_{t \in \mathcal{T}} \left(\sum_{i \in \mathcal{N}} R_i(t)\right). \tag{15}$$

Then, we can formulate the simplified optimization problem (SIMP-OPT-U) as follows:

$$\text{maximize } U(\boldsymbol{x}^0) = \prod_{j \in \mathcal{M}} \left(\bar{u}_j - u_j(\boldsymbol{x}^0, \hat{t})\right) \tag{16}$$

subject to

$$\begin{cases} \sum_{j \in \mathcal{M}} x_j^0 = 1 \\ x_j^0 \geq 0\, \forall j \in \mathcal{M} \\ u_j(\boldsymbol{x}^0, \hat{t}) < \bar{u}_j\, \forall j \in \mathcal{M} \end{cases}. \tag{17}$$

Theorem III.1: The optimal solution to SIMP-OPT-U can formulate a feasible solution to OPT-U.

The abovementioned conclusion can be simply verified by formulating $\boldsymbol{x} = (\boldsymbol{x}_i)_{i \in \mathcal{N}}$ with $\boldsymbol{x}_i = \boldsymbol{x}^0$ for all $i \in \mathcal{N}$, i.e., the allocation strategies of all jobs on a machine $j \in \mathcal{M}$ are the same (equal to x_j^0).

Next, we focus on solving SIMP-OPT-U and try to use the method of Lagrange multiplier.

Notice that, $u_j(\boldsymbol{x}, ts)$ is restricted to be less than $\bar{u}_j$ for all $j \in \mathcal{M}$. We can rewrite the objective of SIMP-OPT-U as

$$U_\Sigma(\boldsymbol{x}^0) = \sum_{j \in \mathcal{M}} \ln\left(\bar{u}_j - u_j(\boldsymbol{x}^0, \hat{t})\right). \tag{18}$$

We first maximize (18) subject to the equality constraint in (17) only, as it is clear that constraint

$$u_j(\boldsymbol{x}^0, \hat{t}) < \bar{u}_j\, \forall j \in \mathcal{M} \tag{19}$$

is inactive, and we will treat the inequality constraint $x_j^0 \geq 0$ ($\forall j \in \mathcal{M}$) as a special case. Hence, the Lagrangian is given by

$$\begin{aligned} L &= \sum_{j \in \mathcal{M}} \ln\left(\bar{u}_j - u_j(\boldsymbol{x}^0, \hat{t})\right) + \alpha \left(\sum_{j \in \mathcal{M}} x_j^0 - 1\right) \\ &= \sum_{j \in \mathcal{M}} \ln\left(\bar{u}_j - \frac{x_j^0 R_\Sigma(\hat{t})}{C_j}\right) + \alpha \left(\sum_{j \in \mathcal{M}} x_j^0 - 1\right) \end{aligned} \tag{20}$$

where $R_\Sigma(\hat{t}) = \sum_{i \in \mathcal{N}} R_i(\hat{t})$. A necessary condition is $\forall j \in \mathcal{M}$

$$\frac{\partial L}{\partial x_j^0} = 0. \tag{21}$$

Solving (21), we get

$$-\frac{R_\Sigma\left(\hat{t}\right)}{\bar{u}_j C_j - x_j^0 R_\Sigma\left(\hat{t}\right)} + \alpha = 0 \tag{22}$$

and

$$x_j^0 = \frac{\bar{u}_j C_j}{R_\Sigma\left(\hat{t}\right)} - \frac{1}{\alpha}. \tag{23}$$

The Lagrange multiplier α needs to be chosen such that the equality constraint $\sum_{j\in\mathcal{M}} x_j^0 = 1$ is satisfied. Hence

$$\sum_{j\in\mathcal{M}} \left(\frac{\bar{u}_j C_j}{R_\Sigma\left(\hat{t}\right)} - \frac{1}{\alpha}\right) = 1 \tag{24}$$

and

$$\alpha = \frac{m R_\Sigma\left(\hat{t}\right)}{\sum_{j\in\mathcal{M}} \bar{u}_j C_j - R_\Sigma\left(\hat{t}\right)}. \tag{25}$$

Substituting the obtained α into (23), we in turn get x_j^0 for all $j \in \mathcal{M}$.

We now need to consider the nonnegative constraint in (17), i.e., $x_j^0 \geq 0\, \forall j \in \mathcal{M}$. Using the results obtained so far, we know that one or more x_j^0 will be nonnegative due to the equality constraint in (17). At the crossroad, we have $x_j^0 = 0$. Setting $x_j^0 = 0$ into (22) gives

$$\alpha_j = \frac{R_\Sigma\left(\hat{t}\right)}{\bar{u}_j C_j}. \tag{26}$$

Equations (23), (25), and (26) give us insight to how negative x_j^0 occurs. If we include all machines in $\mathcal{M}$ as in (24), then we may get $x_j^0 < 0$ due to $\alpha < \alpha_j$. Therefore, some machines may need to be excluded, that is, we set $x_j^0 = 0$ for these machines. We, therefore, sort the machines in non-decreasing order according to their values as given by (26). Denote the sorted sequence as $\boldsymbol{\phi} = (\phi_1, \phi_2, \ldots, \phi_m)$, i.e.,

$$\alpha_{\phi_1} \leq \alpha_{\phi_2} \leq \cdots \leq \alpha_{\phi_m}. \tag{27}$$

Then, the following equation gives a compact representation to the machines that should be included (and the rest excluded):

$$\sum_{l=1}^{d} \left(\frac{\bar{u}_{\phi_l} C_{\phi_l}}{R_\Sigma\left(\hat{t}\right)} - \frac{1}{\alpha_{\phi_d}}\right) \geq 1 \tag{28}$$

where d ($1 \leq d \leq m$) is the minimal positive integer representing the number of machines that should be included, and α is given by the following equation:

$$\sum_{l=1}^{d} \left(\frac{\bar{u}_{\phi_l} C_{\phi_l}}{R_\Sigma\left(\hat{t}\right)} - \frac{1}{\alpha}\right) = 1. \tag{29}$$

Finally, combining with (23), x_j^0 is given as

$$x_{\phi_l}^0 = \frac{1}{d} + \frac{\bar{u}_{\phi_l} C_{\phi_l}}{R_\Sigma\left(\hat{t}\right)} - \frac{1}{d R_\Sigma\left(\hat{t}\right)} \sum_{k=1}^{d} \bar{u}_{\phi_k} C_{\phi_k} \tag{30}$$

for all $1 \leq l \leq d$. For the remaining machines, i.e., machines ϕ_l with $d < \phi_l \leq m$, set $x_{\phi_l}^0 = 0$. The details are presented in Algorithm 1.

Algorithm 1: Calculate-SIMP-OPT-U.

Input: $\mathcal{N}$, $\mathcal{M}$, $\bar{\boldsymbol{u}}$, $\mathbf{C}$, $\boldsymbol{r}$
Output: $\boldsymbol{x}^0$

1 Set $\boldsymbol{x}^0 \leftarrow \mathbf{0}$, $\hat{t} \leftarrow 0$, and a variable $Record \leftarrow 0$;
2 **for** (time slot t from 1 to T) **do**
3 Calculate $R_\Sigma(t) \leftarrow \sum_{i\in\mathcal{N}} R_i(t)$;
4 **if** ($R_\Sigma(t) > Record$) **then**
5 Set $Record \leftarrow R_\Sigma(t)$ and $\hat{t} \leftarrow t$;
6 **end**
7 **end**
8 **for** (machine j from 1 to m) **do**
9 Calculate α_j according to Eq. (26);
10 **end**
11 Sort the machines according to a sequence $\boldsymbol{\phi}$ such that $\alpha_{\phi_1} \leq \alpha_{\phi_2} \leq \ldots \leq \alpha_{\phi_m}$;
12 **for** (variable d from 1 to m) **do**
13 Calculate α_{ϕ_d} as (26), and set variable $S \leftarrow 0$;
14 **for** (iteration l from 1 to d) **do**
15 Set $S \leftarrow S + \left(\frac{\bar{u}_{\phi_l} C_{\phi_l}}{R_\Sigma(\hat{t})} - \frac{1}{\alpha_{\phi_d}}\right)$;
16 **end**
17 **if** ($S \geq 1$) **then break**;
18 **end**
19 **for** (iteration l from 1 to d) **do**
20 Calculate $x_{\phi_l}^0$ as (23), i.e., set $x_{\phi_l}^0 \leftarrow \frac{\bar{u}_{\phi_l} C_{\phi_l}}{R_\Sigma(\hat{t})} - \frac{1}{\alpha_{\phi_d}}$;
21 **end**
22 **return** $\boldsymbol{x}^0$;

By Algorithm 1, we note that Step 1 requires a time of $\Theta(m)$ to initialize $\boldsymbol{x}^0$ and the time complexity of the first for loop (Steps 2–7) is $\Theta(nT)$. The second for loop (Steps 8–10) requires a time of $\Theta(m)$ to calculate α_j for all $j \in \mathcal{M}$. Step 11 requires time $\Theta(m \log m)$ to sort the machines. In the third for loop (Steps 12–18), the number of iterations is at most $\frac{m(m+1)}{2}$ and thus, its time complexity is $\Theta\left(m^2\right)$. Obviously, it takes $\Theta(d)$ for the fourth for loop (Steps 19–21) to calculate x_j^0 ($j \in \mathcal{M}$) with positive values, where d is less than or equal to m. Hence, the time complexity of Calculate-SIMP-OPT-U is $\Theta\left(nT + 3m + m\log m + m^2\right) = \Theta\left(nT + m^2\right)$.

B. Job–Machine Connecting Stereogram Construction

In this section, we construct a stereogram which is the basis of our proposed algorithm. To our knowledge, our constructed stereogram differs from any previous one. Hence, we formally give the definition of our constructed stereogram (called bipartite stereogram) as follows.

Definition III.1: (Bipartite stereogram). A *bipartite stereogram* S is a three-tuple $(\mathcal{N}, G(\mathcal{V}, E_G), E_S)$, where $\mathcal{N}$ is the one-side node set, $G(\mathcal{V}, E_G)$ is a plane graph which locates on the other side. The node set in G is denoted as $\mathcal{V}$, in which nodes are interconnected by edges in E_G. Edge connections between nodes in $\mathcal{N}$ and nodes in $\mathcal{V}$ are incorporated in E_S.

To characterize the relationships between jobs and machines, motivated by the bipartite graph construction in [34], we will construct our bipartite stereogram $S(\boldsymbol{x}) = (\mathcal{N}, G(\mathcal{V}(\boldsymbol{x}), E_G(\boldsymbol{x})), E_S(\boldsymbol{x}))$ based on $\boldsymbol{x}$ and a value $x'(i,w)$ for each edge $(i,w) \in E_S$, where $\mathcal{N}$ is the set of n

job nodes, $G(\mathcal{V}(\boldsymbol{x}), E_G(\boldsymbol{x}))$ is a plane graph in which $\mathcal{V}$ is its node set and E_G is its edge set. Our stereogram construction consists of three stages. At first, we construct an initial bipartite stereogram. One side of the bipartite stereogram consists of job nodes in $\mathcal{N}$. The other side consists of constructed machine sub-nodes $\mathcal{W} = \{w_{j,s}(t)\}_{j\in\mathcal{M},t\in\mathcal{T},1\le s\le k_j(t)}$, where $k_j(t)$ denotes the number of sub-nodes of machine j corresponding to time slot t, and $k_j(t)$ is calculated as

$$k_j(t) = \left\lceil \sum_{i\in\mathcal{A}(t)} x_{i,j} \right\rceil \tag{31}$$

with $\mathcal{A}(t)$ [see (1)] denoting the set of all jobs active at time slot t ($t \in \mathcal{T}$) and $\lceil y \rceil$ denoting the minimal integer that is greater than or equal to y. The $\sum_{t\in\mathcal{T}} k_j(t)$ sub-nodes correspond to machine $j \in \mathcal{M}$. Our bipartite stereogram construction consists of three stages.

Stage I: Machine sub-node and initial bipartite stereogram construction.

The purpose of this stage is to construct an initial bipartite stereogram which depicts the relationships between jobs and machines, according to the obtained fractional solution ($\boldsymbol{x}$). Edges of the initial bipartite stereogram will correspond to job–machine pairs (i, j) if $x_{i,j} > 0$ ($i \in \mathcal{N}$, $j \in \mathcal{M}$). For each positive coordinate of $\boldsymbol{x}$, there will be one or more corresponding edges between job nodes in $\mathcal{N}$ and machine sub-nodes in $\mathcal{W}$. The initial bipartite stereogram and the vector $\boldsymbol{x}'$ are constructed in the following way.

We first sort the jobs in order of nonincreasing resource requirement r_i for all $i \in \mathcal{N}$. Assume that the obtained sequence is $\boldsymbol{\phi} = (\phi_i)_{i\in\mathcal{N}}$, i.e.,

$$r_{\phi_1} \ge r_{\phi_2} \ge \cdots \ge r_{\phi_n}. \tag{32}$$

Then, we construct the edges incident to the sub-nodes corresponding to machine $j \in \mathcal{M}$ as follows.

Case 1: $\forall t \in \mathcal{T}$, if

$$\sum_{1\le l\le n \text{ and } \phi_l\in\mathcal{A}(t)} x_{\phi_l,j} \le 1 \tag{33}$$

then $k_j(t)$ is equal to one and there is only one sub-node $w_{j,1}(t) \in \mathcal{W}$ corresponding to machine j at time slot t. In this case, for each $x_{i,j} > 0$, construct an edge $(i, w_{j,1}(t))$ and set $x'(i, w_{j,1}(t)) = x_{i,j}$.

Case 2: $\forall t \in \mathcal{T}$, if

$$\sum_{1\le l\le n \text{ and } \phi_l\in\mathcal{A}(t)} x_{\phi_l,j} > 1 \tag{34}$$

then find the minimal index l_1 such that

$$\sum_{1\le l\le l_1 \text{ and } \phi_l\in\mathcal{A}(t)} x_{\phi_l,j} \ge 1. \tag{35}$$

Let E_S contain these edges $(\phi_l, w_{j,1}(t))$ for all $1 \le l \le l_1 - 1$ and $\phi_l \in \mathcal{A}(t)$ with $x_{\phi_l,j} > 0$, and set $x'(\phi_l, w_{j,1}(t)) = x_{\phi_l,j}$. In addition, add edge $(\phi_{l_1}, w_{j,1}(t))$ in E_S and set

$$x'(\phi_{l_1}, w_{j,1}(t)) = 1 - \sum_{1\le l\le l_1-1 \text{ and } \phi_l\in\mathcal{A}(t)} x'(\phi_l, w_{j,1}(t)). \tag{36}$$

TABLE I
JOB CONFIGURATION IN EXAMPLE

Job number	1	2	3	4
Load requirement (r_i)	0.1	0.18	0.12	0.1
Execution interval ($\mathcal{I}_i$)	[1, 2]	[2, 3]	[1, 3]	[2, 2]

This ensures that the summation of $\boldsymbol{x}'$ for edges connected to $w_{j,1}(t)$ is exactly equal to one. If

$$\sum_{1\le l\le l_1 \text{ and } \phi_l\in\mathcal{A}(t)} x_{\phi_l,j} > 1 \tag{37}$$

then a fraction of the value $x_{\phi_{l_1},j}$, which equals

$$\sum_{1\le l\le l_1 \text{ and } \phi_l\in\mathcal{A}(t)} x_{\phi_l,j} - 1 \tag{38}$$

is still unassigned. So, we also connect job node ϕ_{l_1} with the next sub-node of machine j corresponding to time slot t, i.e., create an edge $(\phi_{l_1}, w_{j,2}(t))$, and set

$$x'(\phi_{l_1}, w_{j,2}(t)) = \sum_{1\le l\le l_1 \text{ and } \phi_l\in\mathcal{A}(t)} x_{\phi_l,j} - 1. \tag{39}$$

Then, we address the jobs $\phi_l \in \mathcal{A}(t)$ with $l > l_1$, i.e., those jobs with smaller resource requirements on machine j, and construct edges incident to $w_{j,2}(t)$ similar to previous steps. More generally $\forall s = 2, 3, \ldots, k_j(t) - 1$, find the minimal index l_s such that

$$\sum_{1\le l\le l_s \text{ and } \phi_l\in\mathcal{A}(t)} x_{\phi_l,j} \ge s. \tag{40}$$

Add those edges $(\phi_l, w_{j,s}(t))$ for all $l_{s-1} < l \le l_s - 1$ and $\phi_l \in \mathcal{A}(t)$ with $x_{\phi_l,j} > 0$, and set $x'(\phi_l, w_{j,s}(t)) = x_{\phi_l,j}$. Furthermore, add edge $(\phi_{l_s}, w_{j,s}(t))$ in E_S and set

$$x'(\phi_{l_s}, w_{j,s}(t)) = 1 - \sum_{l_{s-1}<l\le l_s-1 \text{ and } \phi_l\in\mathcal{A}(t)} x'(\phi_l, w_{j,s}(t)). \tag{41}$$

If

$$\sum_{1\le l\le l_s \text{ and } \phi_l\in\mathcal{A}(t)} x_{\phi_l,j} > s \tag{42}$$

then, also add edge $(\phi_{l_s}, w_{j,s+1}(t))$ in E_S and set

$$x'(\phi_{l_s}, w_{j,s+1}(t)) = \sum_{1\le l\le l_s \text{ and } \phi_l\in\mathcal{A}(t)} x_{\phi_l,j} - s. \tag{43}$$

For each remaining job $\phi_l \in \mathcal{A}(t)$ with $l_{k_j-1} < l \le n$ and $x_{\phi_l,j} > 0$, construct an edge $\left(\phi_l, w_{j,k_j}(t)\right)$ and set $x'\left(\phi_l, w_{j,k_j}(t)\right) = x_{\phi_l,j}$.

Example. We will use the following instance to give an example of our initial bipartite stereogram construction. The number of time slots T is assumed to be three. Job parameters are shown in Table I. There are $m = 3$ machines with $\bar{u}_1 = 0.7$, $\bar{u}_2 = 0.8$, $\bar{u}_3 = 0.5$, and normalized processing capacity $C_j = 1$ for all $1 \le j \le m$. Then, $R_\Sigma(1) = r_1 + r_3 = 0.22$, $R_\Sigma(2) = r_1 + r_2 + r_3 + r_4 = 0.5$, $R_\Sigma(3) = r_2 + r_3 = 0.3$, and we should

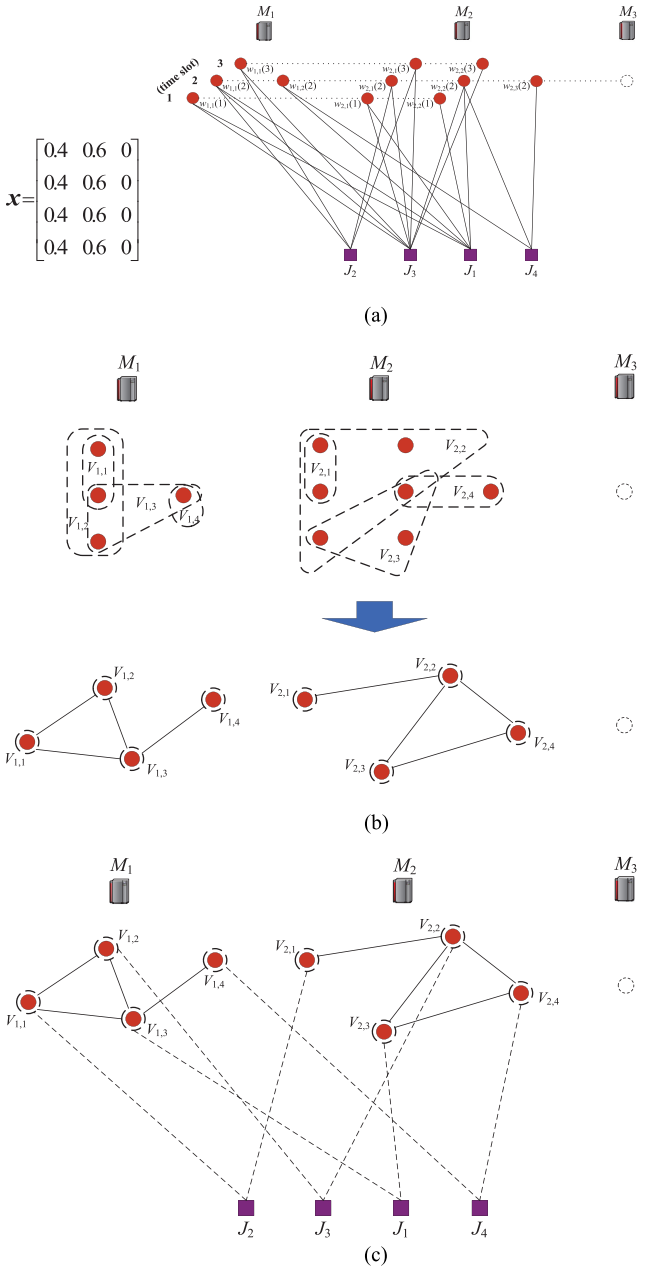

Fig. 5. Three-stage illustration for job–machine connecting stereogram construction. (a) Stage I: A feasible solution $\boldsymbol{x}$ to OPT-U and the corresponding initial bipartite stereogram construction. (b) Stage II: Virtual node building and the connecting relationship construction for each of the machines. (c) Stage III: Relationship stereogram reconstruction ($S(\boldsymbol{x})$) between jobs and the newly constructed virtual nodes.

calculate $\boldsymbol{x}^0$ by using $R_\Sigma(2)$. Sorting the machines, we can obtain a sequence $\boldsymbol{\phi} = (\phi_j)_{1\le j\le m}$, where $\phi_1 = 2, \phi_2 = 1, \phi_3 = 3$. We can calculate $d = 3$ and $\boldsymbol{x}^0 = (0.4, 0.6, 0)$. That is to say, a job is assigned to machine 1 with probability 0.4, machine 2 with probability 0.6, and machine 3 with probability 0. Hence, a feasible fractional solution $\boldsymbol{x}$ and the corresponding initial bipartite stereogram constructed according to Stage I are given in Fig. 5(a).

Stage II: Machine virtual node building and connecting relationship construction among virtual nodes.

After we construct the initial bipartite stereogram, we focus on machine sub-node side, and build virtual nodes and the relationships among them for each machine.

Specifically, for each machine $j \in \mathcal{M}$ with the obtained sub-nodes in Stage I [see Fig. 5(a)], for each job i ($i \in \mathcal{N}$) connected to machine j, we regard the set of connected sub-nodes belonging to machine j as a virtual node. Then, we obtain the newly constructed set of virtual nodes for each machine j, denoted as $\mathcal{V}_j = \{V_{j,1}, V_{j,2}, \ldots, V_{j,q_j}\}$, where q_j is the number newly obtained virtual nodes for machine j. There exists an edge between V_{j,p_1} and V_{j,p_2} ($1 \le p_1, p_2 \le q_j$) if V_{j,p_1} and V_{j,p_2} contain one or more common sub-nodes.

Take Fig. 5(b) for the illustration of Fig. 5(a). From Fig. 5(a), we can observe that for machine M_1, job J_2 connects sub-nodes $w_{1,1}(2)$ and $w_{1,1}(3)$. We construct a virtual node $V_{1,1}$ which consists of $w_{1,1}(2)$ and $w_{1,1}(3)$. Job J_3 connects $w_{1,1}(1)$, $w_{1,1}(2)$, and $w_{1,1}(3)$. We construct a virtual node $V_{1,2}$ which contains $w_{1,1}(1)$, $w_{1,1}(2)$, and $w_{1,1}(3)$. Job J_1 connects $w_{1,1}(1)$, $w_{1,1}(2)$, and $w_{1,2}(2)$. We construct a virtual node $V_{1,3}$ which contains $w_{1,1}(1)$, $w_{1,1}(2)$, and $w_{1,2}(2)$. Job J_4 connects $w_{1,2}(2)$. We construct a virtual node $V_{1,4}$ which contains $w_{1,2}(2)$. There exists an edge between $V_{1,1}$ and $V_{1,2}$ due to which both of them contain $w_{1,1}(2)$ and $w_{1,1}(3)$. Similarly, there exist edges between $Q_{1,1}$ and $Q_{1,3}$, $Q_{1,2}$ and $Q_{1,3}$, $Q_{1,3}$ and $Q_{1,4}$. Hence, we obtain a local graph for M_1. We can similarly obtain a local graph for M_2 [see Fig. 5(b)].

Stage III: Job–machine connecting bipartite stereogram reconstruction.

Based on the newly obtained graph ($G(\boldsymbol{x})$) for all machines in Stage II, we construct our job–machine bipartite stereogram ($S(\boldsymbol{x})$). This construction simply follows a rule that there exists an edge between job J_i ($i \in \mathcal{N}$) and virtual node $V_{j,p}$ ($j \in \mathcal{M}$ and $1 \le p \le q_j$), if there are connections between job J_i and all the sub-nodes contained in $V_{j,p}$. Take Fig. 5(c) as an example, there exists an edge between job J_2 and $Q_{1,1}$ due to that job J_2 connects its sub-nodes $w_{1,1}(2)$ and $w_{1,1}(3)$ [see Fig. 5(a)].

The time complexity of our bipartite stereogram construction can be easily analyzed. Specifically, for each server, the number of sub-nodes is at most nT. As for the virtual node construction stage, we may note that there is at most n virtual nodes corresponding to each machine, and we can implement the construction and connection operations by simply using bit operations, which require time complexity $\Theta(n^2)$. The construction in **Stage III** can be incorporated in **Stage II**. Hence, the time complexity of our bipartite stereogram construction is $\Theta(m(nT + n^2))$.

C. Bipartite Stereogram Matching-Based Scheduling Algorithm

To solve RELIABLE-SCHED, we propose an algorithm called BSMSM, which is developed on a matching of the bipartite stereogram constructed in Section III-B.

To our knowledge, we take the first step to construct a stereogram called bipartite stereogram as in Section III-B and the matching we consider is also different from the matching of a

Algorithm 2: Bipartite-Stereogram-Matching.

Input: a bipartite stereogram S
Output: $\boldsymbol{x}^I$
1 Sort the jobs according to a sequence $\boldsymbol{\phi}$ such that $r_{\phi_1} \geq r_{\phi_2} \geq \ldots \geq r_{\phi_n}$;
2 Set $\mathbf{L} \leftarrow \mathbf{0}$, $\mathbf{R} \leftarrow \mathbf{0}$, and $\boldsymbol{x}^I \leftarrow \mathbf{0}$;
3 **for** (variable l from 1 to n) **do**
4 | Set **visit** $\leftarrow \mathbf{0}$, **mark** $\leftarrow \mathbf{0}$, and $i \leftarrow \phi_l$;
5 | $(\mathbf{L}, \mathbf{R}) \leftarrow$ **Match-Job-Node(**i**, visit, mark, L, R)**;
6 **end**
7 **for** (each job i from 1 to n) **do**
8 | For the machine j containing $L(i)$, set $x^I_{i,j} \leftarrow 1$;
9 **end**
10 **return** $\boldsymbol{x}^I$;

bipartite graph or other matching definitions. Hence, we give the formal matching definition for our bipartite stereogram.

Definition III.2: (Bipartite Stereogram Matching). A *matching* for a bipartite stereogram $S = (\mathcal{N}, G(\mathcal{V}, E_G), E_S)$ is an edge subset of E_S which satisfies that $\forall V \in \mathcal{V}$, if V is in the *matching*, i.e., V has been matched to a node in $\mathcal{N}$, then the *matching* can not cover its adjacent nodes in $G(\mathcal{V}, E_G)$.

1) Matching Algorithm Design: The proposed BSMSM is based on a matching of our constructed bipartite stereogram. Hence, before presenting BSMSM, we first propose an algorithm aiming to find a matching, which is defined in Definition III.2, for the bipartite stereogram. The algorithm details are presented in Algorithm 2.

As shown in **Bipartite-Stereogram-Matching**, we first sort the jobs such that $r_{\phi_1} \geq r_{\phi_2} \geq \cdots \geq r_{\phi_n}$ (see Step 1), where $\boldsymbol{\phi} = (\phi_l)_{1 \leq l \leq n}$ is the ordered sequence. Then, according to $\boldsymbol{\phi}$, we find a matching in $S(\boldsymbol{x})$ for each job node ϕ_l $(1 \leq l \leq n)$ (see Steps 3–6), i.e., for each job node ϕ_l, we find a matching virtual node in $G(\boldsymbol{x})$. Furthermore, in this stereogram matching, if a virtual node in $G(\boldsymbol{x})$ is matched to a job node, then its direct adjacent virtual nodes can not be matched to any job nodes. For example, as shown in Fig. 5(c), if $V_{1,1}$ is matched to job J_2, then its adjacent virtual nodes $V_{1,2}$ and $V_{1,3}$ can not be matched to any job nodes. This is the core difference between our bipartite stereogram matching and the general bipartite graph matching, and the problem becomes more complex. To find a matching node in $G(\boldsymbol{x})$ for each job node ϕ_l (i), **Bipartite-Stereogram-Matching** calls **Match-Job-Node**, which is adapted from the well-known Hungarian algorithm [35] and shown in Algorithm 3.

As mentioned above, the purpose of **Match-Job-Node** (see Algorithm 3) is to find a matching node in $G(\boldsymbol{x})$ for job J_i $(i \in \mathcal{N})$ under the previous matching. Moreover, in this stereogram matching, if a virtual node in $G(\boldsymbol{x})$ is matched to a job node, then its direct adjacent virtual nodes can not be matched to any job nodes. Our algorithm proceeds as follows.

Denote $\mathcal{V}^S(i)$ as the set of virtual nodes in S connected to job node J_i. Take Fig. 5(c) as an example, $\mathcal{V}^S(2)$ denotes the set of $V_{1,1}$ and $V_{2,1}$ which are connected to J_2. Then, **Match-Job-Node** tries to search a virtual node in $\mathcal{V}^S(i)$ to match job

Algorithm 3: Match-Job-Node(i_u**, visit, mark, L, R).**

Input: i_u, **visit**, **mark**, **L**, **R**
Output: $(\mathbf{L}, \mathbf{R})$
1 Construct an empty list $List$ and a list pointer $pt \leftarrow List$. Set $\mathbf{L}^{(old)} \leftarrow \mathbf{L}$ and $\mathbf{R}^{(old)} \leftarrow \mathbf{R}$;
2 Construct a list node Q. Set $Q.i \leftarrow i_u$, $Q.f \leftarrow -1$, $Q.color \leftarrow$ **false**, $Q.prev \leftarrow$ NULL, $Q.next \leftarrow$ NULL. Do operation $pt.add(Q)$ and set $pt \leftarrow pt{->}next$;
3 **while** (pointer pt is not NULL) **do**
4 | Set $i \leftarrow pt.i$ and $ans \leftarrow$ **false**;
5 | **for** (each virtual node $V \in \mathcal{V}^S(i)$) **do**
6 | | **if** ($mark(V)$ or $visit(V)$ is **true**) **then**
7 | | | **continue**;
8 | | **end**
9 | | Set $visit(V) \leftarrow$ **true** and $ans \leftarrow$ **true**;
10 | | **if** (virtual node V has been matched) **then**
11 | | | Construct a list node Q. Set $Q.i \leftarrow R(V)$, $Q.f \leftarrow i$, $Q.color \leftarrow$ **false**. Do operation $pt.add(Q)$;
12 | | **else**
13 | | | Construct a new pointer $pt^{(n)} \leftarrow pt$;
14 | | | **for** (each virtual node $V' \in \mathcal{V}^G(V)$) **do**
15 | | | | **if** (V' has been matched) **then**
16 | | | | | Construct a list node Q. Set $Q.i \leftarrow R(V')$, $Q.f \leftarrow i$, $Q.color \leftarrow$ **false**. Do operation $pt^{(n)}.add(Q)$;
17 | | | | | Set $pt^{(n)} \leftarrow pt^{(n)}{->}next$;
18 | | | | **end**
19 | | | **end**
20 | | **end**
21 | | Set $L(i) \leftarrow V$, $R(V) \leftarrow i$, $pt{->}color \leftarrow$ **true**, and $mark(V) \leftarrow mark(V) + 1$;
22 | | **for** (each virtual node $V' \in \mathcal{V}^G(V)$) **do**
23 | | | Set $mark(V') \leftarrow mark(V') + 1$;
24 | | **end**
25 | | **break**;
26 | **end**
27 | **if** (ans is **false**) **then**
28 | | Set $pt \leftarrow pt{->}prev$ and $pt^{(n)} \leftarrow pt{->}next$;
29 | | Set $i' \leftarrow pt{->}i$ and $V \leftarrow L(i')$;
30 | | **if** ($pt{->}color$ is **true**) **then**
31 | | | Set $mark(V) \leftarrow mark(V) - 1$;
32 | | | **for** (each virtual node $V' \in \mathcal{V}^G(V)$) **do**
33 | | | | Set $mark(V') \leftarrow mark(V') - 1$;
34 | | | **end**
35 | | | Set $pt{->}color \leftarrow$ **false**;
36 | | **end**
37 | | **while** ($pt^{(n)}.f$ is equal to $pt.i$) **do**
38 | | | Set $i \leftarrow pt^{(n)}.i$, $L(i) \leftarrow L^{(old)}(i)$, and $R(L(i)) \leftarrow i$;
39 | | | Set $pt^{(n)} \leftarrow pt^{(n)}{->}next$ and **free**$(pt^{(n)}{->}prev)$;
40 | | **end**
41 | | Set $pt{->}next \leftarrow pt^{(n)}$ and $pt^{(n)}{->}prev \leftarrow pt$;
42 | **else**
43 | | Set $pt \leftarrow pt{->}next$;
44 | **end**
45 **end**
46 **return** $(\mathbf{L}, \mathbf{R})$;

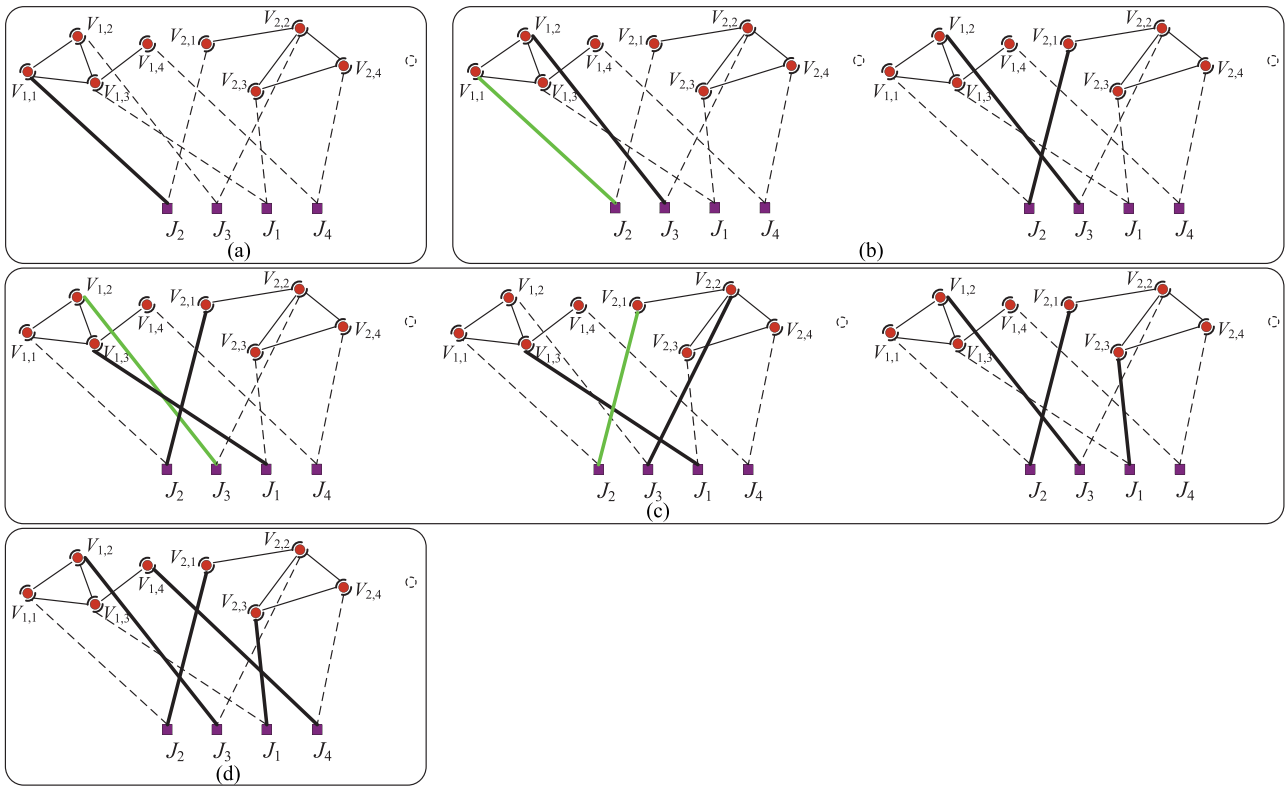

Fig. 6. Matching process illustration. (a) Matching process for J_2. (b) Matching process for J_3. (c) Matching process for J_1. (d) Matching process for J_4.

node J_i and adjust the previous matching. Specifically, we use a boolean vector **visit** to denote whether the virtual nodes in $\mathcal{V}$ are visited under this matching process. We use another integer vector **mark** to denote whether a virtual node is matched (1 means matched and 0, otherwise) or the number of its neighboring virtual nodes which are matched. At first, **Match-Job-Node** constructs a dynamic list $List$ and an associated list pointer pt (Step 1). Then, it iteratively finds a matching for each of the job nodes in $List$ by moving pt and terminates until the iteration reaches the end of $List$ (Steps 3–45).

For the current job node i pointed by pt, we try to find (or adjust) its matching node. We search all of the virtual nodes connected to J_i, i.e., $\mathcal{V}^S(i)$ (Steps 5–26). For each $V \in \mathcal{V}^S(i)$, if V has been visited in this matching process or V has been marked, then skip this node. Otherwise, we set $visit(V)$ as true value (Step 9) and discuss two cases (Steps 10–20).

Case I: If V has been matched (Steps 10–12), then this means that none of its adjacent virtual nodes has been matched and we only need to consider the job node matched to V, denoted as $R(V)$ ($R(V) \in \mathcal{N}$). Under this situation, we construct a list node for $R(V)$ and add it in $List$ next to pt.

Case II: If V does not have to be matched (Steps 12–20), this means that V is an appropriate matching node for J_i or some nodes in $\mathcal{V}^G(V)$ have been matched, where $\mathcal{V}^G(V)$ denotes the set of neighboring nodes of V in $G(\mathcal{V}, E_G)$. If it is the latter, we need to adjust the matching for all job nodes matched to the neighboring nodes of V and add the job nodes in $List$ next to pt (Steps 16–17). Then, we temporally match job node i to V, and mark V and its adjacent nodes in G once more (Steps 21–24).

After the searching process for job node i, i.e., at the end of the for loop (Steps 5–26), we need to decide the next searching job node, which is incorporated in the if sentence (Steps 27–44). More specifically, if the value of ans is false, this means that under the current matching for all job nodes in $List$ before pt, we can not find a matching node for job i. This implies that we need to adjust the matching of some previous job nodes. Hence, we move pt to point its previous job node $i^{'}$, i.e., job node $i^{'}$ needs to be matched to another virtual node, and do minus operation for its marked nodes (Steps 30–36). The while loop (Steps 37–40) is used to delete the job nodes added next to $i^{'}$ in $List$ due to its previous matching.

2) Illustration: Fig. 6 presents the matching process of **Bipartite-Stereogram-Matching** (especially for **Match-Job-Node**) for the bipartite stereogram shown in Fig. 5(c). As shown in Fig. 6(a), at first, **Match-Job-Node** tries to find a matching node for J_2. At the beginning, the link list $List$ only has one list node J_2 (Steps 1 and 2 of Algorithm 3). In the while loop (Steps 3–45), we first scan the node J_2 in $List$ (Step 4) and search its possible matching nodes in $\mathcal{V}^S(2)$, i.e., $V_{1,1}$, $V_{2,1}$ (Steps 5–26). When checking $V_{1,1}$, **Match-Job-Node** finds that $V_{1,1}$ does not have to be matched. Hence, it tries to add the job nodes matched to the neighboring nodes of $V_{1,1}$ in G (i.e., $V_{1,2}$, $V_{1,3}$) (Steps 14–18) and there is no such kind of job nodes. After skipping out of the for loop, pointer pt is moved to the next one which is NULL. Hence, **Match-Job-Node** terminates the matching process of J_2 and finds its matching node $V_{1,1}$ [see Fig. 6(a)]. As for J_3, **Match-Job-Node** first searches its possible matching node $V_{1,2}$ (Step 5–26) and checks whether there are

job nodes matched to some neighboring nodes of $V_{1,2}$ in G (Steps 13–19). It finds that the neighboring node $V_{1,1}$ has been matched by J_2. Hence, it adds a list node J_2 next to J_3 in $List$. Then, pt is moved to the next list node and **Match-Job-Node** tries to find another matching node for J_2 (Step 4). Hence, it searches $V_{2,1}$ and finds that $V_{2,1}$ can be matched to J_2 (Steps 5–26). Then, pt is moved to NULL and **Match-Job-Node** terminates the matching process of J_3 [see Fig. 6(b)].

Similarly, **Match-Job-Node** finds a matching node for J_1 and J_4, respectively [see Fig. 6(c) and (d)]. This ends the matching illustration.

3) Time Complexity and Property Analyses: Next, we analyze the time complexity of **Bipartite-Stereogram-Matching**. Since its main operation is the call of **Match-Job-Node**, we first analyze the time complexity of **Match-Job-Node**. The details are proceeded as follows.

For each job node, there is at most one virtual node corresponding to each of the machines. Then, the number of iterations of the for loop (Steps 5–26) is $\Theta(m)$. Furthermore, in each of the iterations, for a virtual node V, **Match-Job-Node** scans the adjacent virtual nodes of V, which requires time complexity $\Theta(n-1)$ (Steps 10–20). Hence, the time complexity of the for loop (Steps 5–26) is $\Theta(mn)$. Since the main operation of the if sentence (Steps 27–44) lies in the while loop (Steps 37–40), which is also a scanning operation requiring time $\Theta(n-1)$, the time complexity required by one iteration of the while loop (Steps 4–44) is $\Theta(mn)$. Hence, to configure the time complexity of **Match-Job-Node**, we only need to evaluate the overall number of iterations of the while loop (Steps 3–45).

Notice that, during a matching process (i.e., once time execution of **Match-Job-Node**), each virtual node is visited at most once and for each virtual node, there is at most one job node added in $List$. Hence, the overall number of iterations of the while loop (Steps 3–45) is $\Theta(mn)$ and the time complexity of **Match-Job-Node**) $\Theta\left(m^2n^2\right)$. Then, we can easily obtain that the time complexity of **Bipartite-Stereogram-Matching** is $\Theta\left(m^3n^2\right)$.

For the schedule obtained by **Bipartite-Stereogram-Matching**, we can conclude the result as in Theorem III.2.

Specifically, denote i_j ($i_j \in \mathcal{N}$) as the job with maximal workload requirement among all jobs allocated to machine j ($j \in \mathcal{M}$), i.e.,

$$\hat{i}_j \in \underset{i\in\mathcal{N} \text{ and } x^I_{i,j}=1}{\arg\max} (r_i) \tag{44}$$

and $\bar{\rho}_j$ be the ratio between $r_{\hat{i}_j}$ and C_j, i.e., $\bar{\rho}_j = \frac{r_{\hat{i}_j}}{C_j}$. Then, we can conclude the following theorem.

Theorem III.2: For the solution $\boldsymbol{x}^I$ returned by **Bipartite-Stereogram-Matching**, we can conclude

$$u_j^{\max}\left(\boldsymbol{x}^I\right) \le \bar{u}_j + \bar{\rho}_j \; \forall j \in \mathcal{M} \tag{45}$$

where $u_j^{\max}\left(\boldsymbol{x}^I\right)$ [see (7)] denotes the maximal workload utilization of machine j among all the T time slots under job allocation strategy $\boldsymbol{x}^I$.

Proof: Let $r_{j,s}^{\max}(t)$ $(1 \le s \le k_j(t))$ be the maximum of the resource requirements r_i $(i \in \mathcal{N})$ corresponding to edges $(i, w_{j,s}(t)) \in E_S$, and $r_{j,s}^{\min}(t)$ be the analogous minimum. Then, we have $r_{j,s}^{\min}(t) \ge r_{j,s+1}^{\max}(t)$ for all $1 \le s \le k_j(t)-1$ due to the reason that the jobs are sorted in nonincreasing order and consequently mapped to the nodes corresponding to each machine.

Notice that $\forall j \in \mathcal{M}$, there are $\sum_{t\in\mathcal{T}} k_j(t)$ sub-nodes corresponding to machine M_j in the bipartite stereogram [see Fig. 5(a)]. Furthermore, we find a job–machine matching which satisfies that if a virtual node in $\mathcal{V}_j$ is matched to a job node, then its direct adjacent virtual nodes, which contain common machine sub-nodes, can not be matched to any job nodes. This implies that for each sub-node $w_{j,s}(t)$ $(j \in \mathcal{M}, t \in \mathcal{T}, 1 \le s \le k_j(t))$, there will be at most one job scheduled on M_j corresponding to one of the incident edges.

Therefore $\forall j \in \mathcal{M}$ and $\forall t \in \mathcal{T}$

$$\begin{aligned}
\sum_{s=2}^{k_j(t)} \frac{r_{j,s}^{\max}}{C_j} &\le \sum_{s=1}^{k_j(t)-1} \frac{r_{j,s}^{\min}}{C_j} \\
&\le \sum_{s=1}^{k_j(t)-1} \sum_{i:\,(i,w_{j,s}(t))\in S^0(\boldsymbol{x})} x'\left(i, w_{j,s}(t)\right) \frac{r_i}{C_j} \\
&\le \sum_{s=1}^{k_j(t)} \sum_{i:\,(i,w_{j,s}(t))\in S^0(\boldsymbol{x})} x'\left(i, w_{j,s}(t)\right) \frac{r_i}{C_j} \\
&\le \sum_{i\in\mathcal{A}(t)} x_{i,j} \frac{r_i}{C_j} \le \bar{\mu}_j
\end{aligned}$$

and

$$u_j\left(\boldsymbol{x}^I, t\right) \le \bar{u}_j + \frac{r_{j,1}^{\max}(t)}{C_j} \le \bar{u}_j + \bar{\rho}_j.$$

This completes the proof and the result follows. ■

4) BSMSM Design: Notice that, after we obtain $\boldsymbol{x}^I$, we can slightly make adjustments to mitigate the possible excesses, i.e., move some jobs on servers with exceeded workloads to servers without exceeded workloads. Combining the previous bipartite stereogram matching and this adjustment idea, we formulate our scheduling algorithm BSMSM in Algorithm 4.

Compared with **Bipartite-Stereogram-Matching**, algorithm BSMSM tries to mitigate the possible workload excesses from the following two aspects.

1) Decrease the workload limitations of servers by introducing a parameter δ when calculating. By doing so, the schedule returned by **Bipartite-Stereogram-Matching** is more likely to perform reliably due to the matching property concluded in Theorem III.2.
2) Add the scaling operation which involves moving jobs on servers with workload excesses.

As mentioned above, in each iteration of the scaling operation (Steps 6–12) in Algorithm 4, we try to find a machine with workload excess and a job on this machine which can decrease its workload utilization. The time complexity of this process is $\Theta(mn)$. In addition, we try to find another machine which

TABLE II
SYSTEM PARAMETERS

System parameters	(Fixed)–[Varied range] (Increment)
Resource requirement (r_i)	[0.0, 0.4]–[0.0, 0.1-0.5] (0.1)
Scaling parameter (δ)	(0.35)–{0.15, 0.25, 0.35}
# of jobs and servers (n, m)	(300, 100)
Workload limitation ($\bar{u}_j$)	[0.5, 0.9]
Number of time slots (T)	20
Normalized capacity (C_j)	1

Algorithm 4: $\mathcal{B}$ipartite $\mathcal{S}$tereogram $\mathcal{M}$atching based $\mathcal{S}$cheduling $\mathcal{M}$ethod ($\mathcal{BSMSM}$).

Input: $\mathcal{N}$, $\mathcal{M}$, $\bar{\boldsymbol{u}}$, $\boldsymbol{C}$, $\boldsymbol{r}$, δ
Output: $\boldsymbol{x}^I$
Set $\boldsymbol{x}^0 \leftarrow$ **Calculate-SIMP-OPT-U($\mathcal{N}, \mathcal{M}, (1-\delta)\bar{\boldsymbol{u}}, \boldsymbol{C}, \boldsymbol{r}$)**, and $\boldsymbol{x} \leftarrow (\boldsymbol{x}_i)_{i\in\mathcal{N}}$ with $\boldsymbol{x}_i = \boldsymbol{x}^0$;
Construct our bipartite stereogram $S(\boldsymbol{x})$ as shown in Section 3.2;
Set $\boldsymbol{x}^I \leftarrow$ **Bipartite-Stereogram-Matching($S(\boldsymbol{x})$)**;
// Do scaling operation
Set a boolean variable $flag \leftarrow$ **true**;
while ($flag$ is equal to **true**) **do**
  Set $flag \leftarrow$ **false**, and find a machine $j \in \mathcal{M}$ whose workload satisfies $u_j^{\max} > (1-\delta)\,\bar{u}_j$.
  Find a job i on machine j which decreases $u_j^{\max}$ and can be migrated to another machine j' such that $u_{j'}^{\max} \le (1-\delta)\,\bar{u}_{j'}$.
  if (there is no such a machine or job) **then**
    Set $flag \leftarrow$ **false**;
  end
end
return $\boldsymbol{x}^I$;

can execute the selected job without workload excess. Hence, once time moving operation requires time complexity $\Theta\left(m^2 n\right)$. Since there is at most one moving operation in BSMSM for each job and there are total n jobs, we can conclude that the time complexity of the scaling operation in BSMSM is $\Theta\left(m^2 n^2\right)$. Combined with the previous analyzed time complexity results $\Theta\left(m^2\right)$ for Step 1, $\Theta\left(m\left(nT+n^2\right)\right)$ for Step 2, and $\Theta\left(m^3 n^2\right)$ for Step 3, we can conclude that the time complexity of BSMSM is $\Theta\left(m^3 n^2 + mnT\right)$.

IV. EXPERIMENTAL EVALUATION

In this section, we provide extensive random experiments to validate the performance of BSMSM.

A. Parameter Configuration

In our experiments, we consider the schedule of 300 jobs to 100 heterogeneous servers, i.e., the numbers of jobs (n) and servers (m) are 300 and 100, respectively. As shown in Table II, all servers' resource capacities (C_j) are normalized as one and their workload limitations ($\bar{u}_j$) are randomly and uniformly distributed in interval [0.5, 0.9]. The resource demands of the 300 jobs are normalized, and randomly and uniformly distributed in intervals [0.0, 0.1], [0.0, 0.2], [0.0, 0.3], [0.0, 0.4], [0.0, 0.5]. Furthermore, for each interval, we randomly and uniformly generate 100 groups of data for all jobs. The scaling parameter (δ) varies in 0.15, 0.25, and 0.35. Of course, other parameter settings are also feasible.

B. Random Results

Fig. 7 presents the results of a small instance configured as in Table II except that n is set as 20, m is set as 6, and T is set as 10. We compare three types of values, i.e., scaled workload limitation, workload utilization before scaling operation (schedule obtained by **Bipartite-Stereogram-Matching**), and workload utilization after scaling operation (schedule obtained by BSMSM). From Fig. 7, we can observe that when workload limitations are great enough, the results without scaling and those with scaling are the same, i.e., there is no need to scale the schedule obtained by **Bipartite-Stereogram-Matching** due to the fact that the obtained schedule is already reliable, i.e., the overall load utilization on each server does not exceed the server's scaled workload limitation [see Fig. 7(a) and (b)]. On the other hand, when the scaled workload limitations are low enough, workload utilizations on some servers obtained by **Bipartite-Stereogram-Matching** exceed the scaled workload limitation. However, the excesses are not large [see Fig. 7(c)]. Furthermore, our proposed BSMSM can do some slight adjustments to mitigate the excesses.

Fig. 8 presents the results with the variation of resource requirement interval. Since for each interval, we have 100 groups of random data, we present the average workload utilization as well as the maximum workload and minimum workload results. As shown in Fig. 8, we randomly select nine machines (M_7, M_{21}, M_{40}, M_{42}, M_{59}, M_{63}, M_{82}, M_{86}, and M_{95}) out of 100 to show their workload trends with the variation of resource requirement interval. We can observe that both the average and maximum workload utilizations of all machines tend to increase with the increase of interval. In addition, when resource requirement interval is not large ([0.0, 0.1], [0.0, 0.2], or [0.0, 0.3]), the results before scaling and after scaling are the same. However, there are differences between them when the interval is somewhat larger ([0.0, 0.4] or [0.0, 0.5]). The reason lies in that when job resource demands are low, the scaled workload limitations of machines are large enough to pack all jobs, and there is no need to scale the schedule obtained by **Bipartite-Stereogram-Matching**. However, when the interval increases to [0.0, 0.4], the matching results can not satisfy the scaled workload limitations (e.g., M_{40}, M_{63}, M_{86}). Hence, there are some slight adjustments in our BSMSM, i.e., job migrations among machines. Specifically, we can observe that the average workload utilizations of some machines tend to increase (M_7, M_{21}, M_{42}, M_{59}, M_{82}) while some average workload utilizations tend to decrease (M_{40}, M_{63}, M_{86}). Furthermore, all scaled workload utilizations (including maximum workload utilization values) tend to be smaller than the corresponding scaled workload limitations even though some maximum workload utilization values increase (M_{21}, M_{42}, M_{59}, M_{82}, M_{95}). When resource requirement interval is large enough ([0.0, 0.5]), both results before

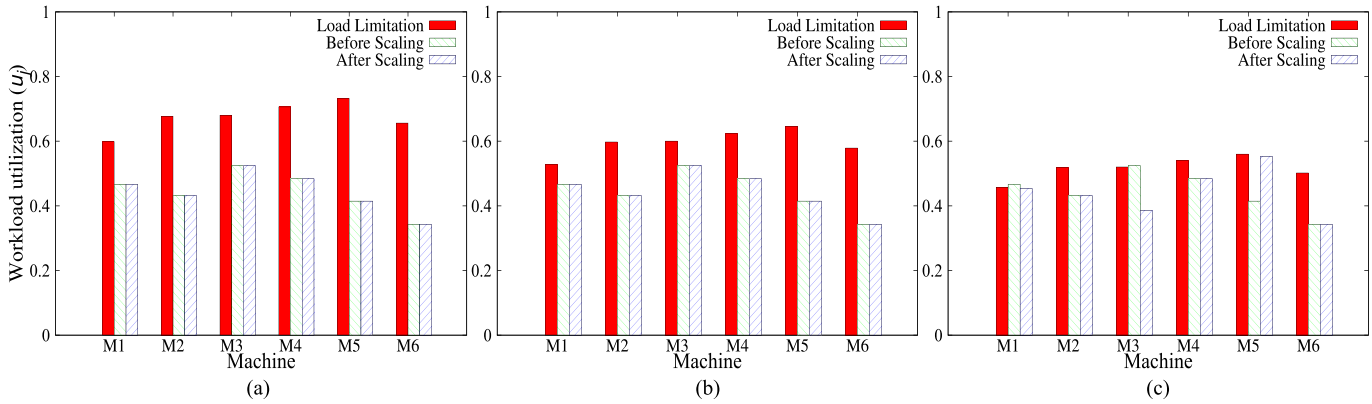

Fig. 7. One small instance with $\delta = 0.15$, 0.25, and 0.35. (a) $\delta = 0.15$. (b) $\delta = 0.25$. (c) $\delta = 0.35$.

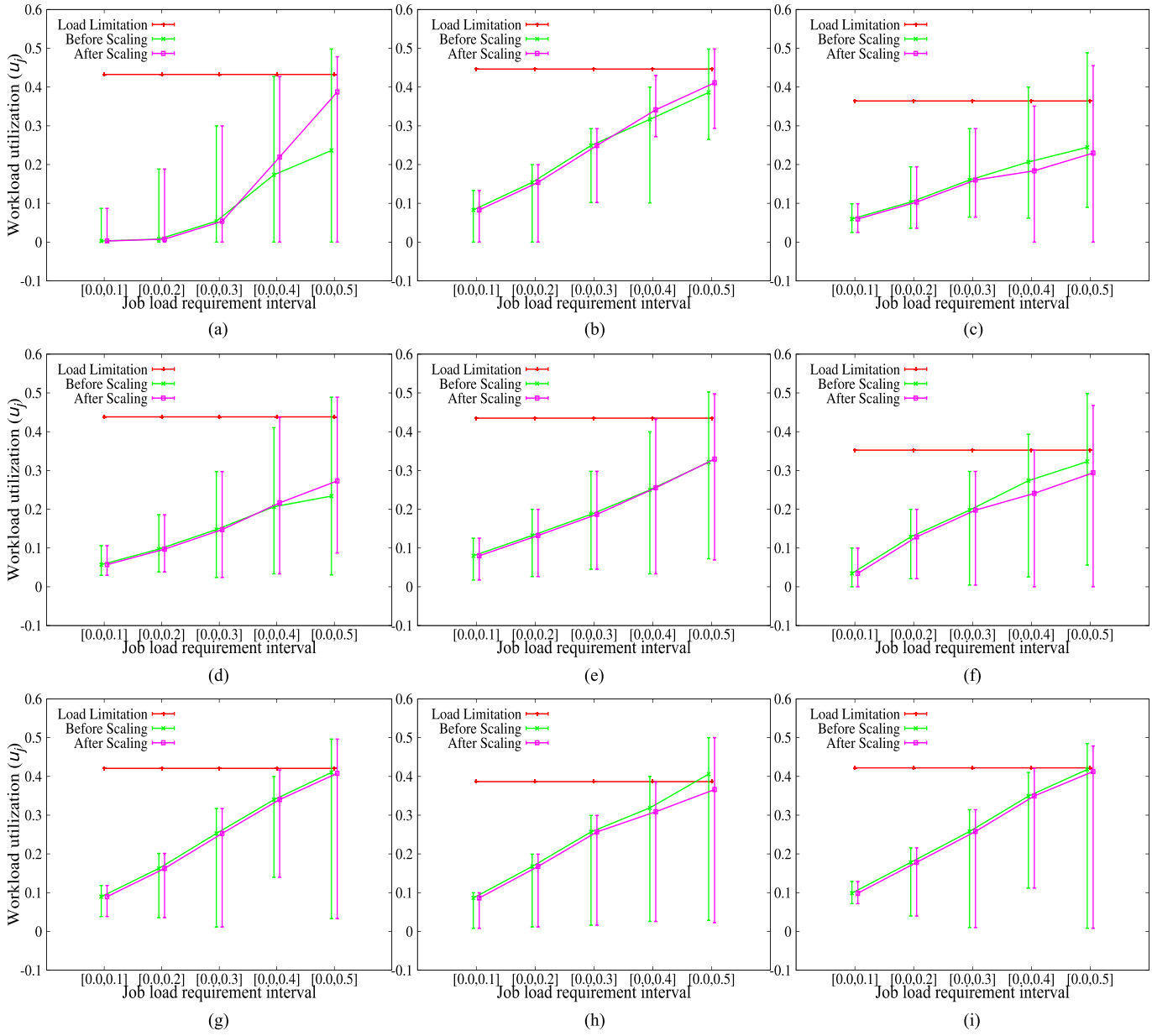

Fig. 8. Results with varied resource requirement interval. (a) M7. (b) M21. (c) M40. (d) M42. (e) M59. (f) M63. (g) M82. (h) M86. (i) M95.

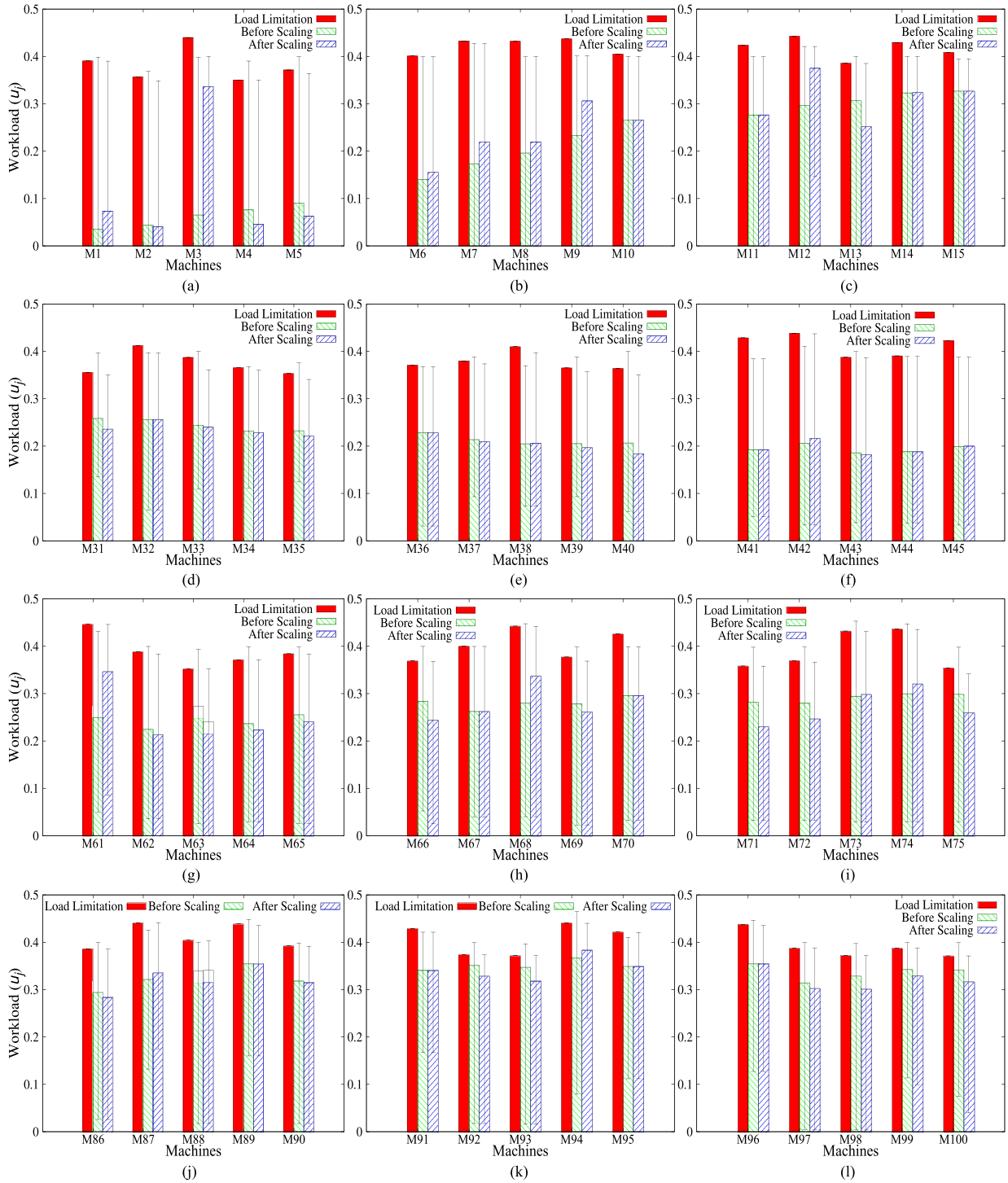

Fig. 9. More specific results with and without scaling. (a) M1–M5. (b) M6–M10. (c) M11–M15. (d) M31–M35. (e) M36–M40. (f) M41–M45. (g) M61–M65. (h) M66–M70. (i) M71–M75. (j) M86–M90. (k) M91–M95. (l) M96–M100.

scaling and after scaling tend to exceed the scaled workload limitations. However, some of the maximum results after scaling are somewhat smaller (M_7, M_{40}, M_{59}, M_{63}, M_{95}). The reason lies in that the scaled workload limitations are too low to satisfy all jobs' resource demands. However, notice that even under this condition, the workloads of all machines tend to be smaller than the lower bound of all original workload limitations.

Fig. 9 presents more specific results with scaling and without scaling (including maximum and minimum workload utilization values). Specifically, we show 60 workload utilizations out of 100. From Fig. 9, we can observe that the increase of average workload utilization does not always imply the increase of maximum or minimum workload utilization (e.g., M_1, M_3, M_6–M_9) even though some workload utilizations are (M_{61}, M_{87}).

TABLE III
RANDOMLY COLLECTED 100 RUNNING JOBS WITH THEIR CPU LOADS FROM TIANHE-1A (NATIONAL SUPERCOMPUTING CENTER IN CHANGSHA)

jobs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
CPU usage (MHz)	1020	1210	1.03	1330	2.52	1180	1.77	3.68	1480	2.17	3.98	3.65	1070	4.52	5.22	6.66	4.95	6.93	6.45	14.23
CPU load (%)	34.81	41.30	0.035	45.39	0.86	40.27	0.061	0.13	50.51	0.074	0.14	0.12	36.52	0.15	0.18	0.23	0.17	0.24	0.22	0.49
jobs	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
CPU usage (MHz)	11.23	7.84	5.21	4.51	1150	1280	1310	3.75	6.85	29.54	16.90	21.64	7.91	17.69	58.28	101.38	14.29	18.96	10.50	2.61
CPU load (%)	0.38	0.27	0.18	0.15	39.25	43.69	44.71	0.13	0.23	1.01	0.58	0.74	0.27	0.60	1.99	3.46	0.49	0.65	0.36	0.089
jobs	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
CPU usage (MHz)	9.08	7.48	1290	7.16	8.07	10.14	9.04	3.99	3.91	8.68	933.33	5.23	147.69	58.04	98.79	10.81	13.19	143.8	30.51	4.59
CPU load (%)	0.31	0.26	44.03	0.24	0.28	0.35	0.31	0.14	0.13	0.30	31.85	0.18	5.04	1.98	3.37	0.37	0.45	4.91	1.04	0.16
jobs	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
CPU usage (MHz)	253.57	34.65	10.68	101.28	48.05	5.71	254.88	30.03	4.29	3.46	3.48	15.0	4.43	11.41	5.58	20.06	17.34	50.96	44.85	18.03
CPU load (%)	8.65	1.18	0.36	3.46	1.64	0.19	8.70	1.02	0.15	0.12	0.12	0.51	0.15	0.39	0.19	0.68	0.59	1.74	1.53	0.62
jobs	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
CPU usage (MHz)	8.74	12.34	1200	6.03	12.49	8.96	3.56	533.33	666.67	11.46	1470	8.43	3.93	6.32	6.05	8.05	21.84	12.11	14.60	20.46
CPU load (%)	0.30	0.42	40.96	0.21	0.43	0.31	0.12	18.20	22.75	0.39	50.17	0.29	0.13	0.22	0.21	0.27	0.75	0.41	0.50	0.70

The reason lies in that the resource demands of the jobs migrated to these machines are not the maximum or the minimum. In addition, when some workload utilizations exceed their machine workload limitations, the scaling operation in our proposed BSMSM works (e.g., M_1, M_{31}, M_{33}, M_{35}). We can also observe that the excesses are not very large. This partly validates the results shown in Fig. 8 and the feasibility of our proposed method.

To show that our method is feasible and not limited to parameter settings, we also perform another set of experiments with different configurations. The different parameter settings with Table II are shown in Table IV, in which we set the workload threshold of each server as a random value uniformly chosen from [0.5, 0.7]. The interval [0.5, 0.7] is determined according to an observation that a server is more likely to crash when its utilization exceeds 60% [36]. We also increase the number of jobs as 600 and set δ as zero.

The results corresponding to Table IV are shown in Fig. 10. We can observe that the results in Fig. 10 show similar shapes as those in Fig. 8. Specifically, when resource requirements of jobs are relatively low (e.g., [0.0, 0.1], [0.0, 0.2], [0.0, 0.3], and [0.0, 0.4]), our method can find a reliable schedule, i.e., workload on each server can be guaranteed less than its threshold. However, when resource requirements of jobs are large enough ([0.0, 0.5]), our method can not find such a reliable schedule. The reason lies in that the overall resource requirement of jobs exceeds the overall capacity of all servers limited by thresholds [see Fig. 10(a)–(l)]. However, we can also observe that no matter what the job requirement interval is, the average results on each server are always smaller than its threshold. The similarities between Figs. 8 and 10 verify that our method is not limited to parameter settings.

C. Application for Real Jobs

To further investigate the feasibility and applicability of our proposed method, we also perform another set of experiments with real job resource usages. Specifically, we randomly collect

TABLE IV
SYSTEM PARAMETERS

Different configurations with Table 2	(Fixed)–[Varied range]
Scaling parameter and # of jobs (δ, n)	(0, 600)
Workload limitation $(\bar{u}_j)$	[0.5, 0.7]

TABLE V
SYSTEM PARAMETERS

Parameter configurations	(Fixed)–[Varied range]
Scaling parameter and # of servers (δ, m)	(0, 50)
# of jobs	[100, 1100]
Workload limitation $(\bar{u}_j)$	[0.5, 0.7]

100 jobs running in Tianhe-1A (National Supercomputing Center in Changsha) and record their occupied CPU loads. The results are shown in Table III. Then, we generate n jobs based on the collected 100 jobs with their corresponding resource usages, i.e., for each generated job, its workload is randomly and uniformly selected from the 100 jobs. Other parameter settings are shown in Table V, in which we also set the workload threshold of each server as a random value uniformly chosen from [0.5, 0.7], which is determined according to an observation that a server is more likely to crash when its utilization exceeds 60% [36]. We vary the number of generated jobs (n) between 100 and 1100, and set δ as zero.

The results corresponding to the configurations in Table V are shown in Figs. 11 and 12.

We can observe that the results in Fig. 11 show similar shapes as those in Figs. 8 and 10. Specifically, with the number of jobs varying from 100 to 875, our method can always find a reliable schedule, i.e., workload on each server can be guaranteed to be less than or equal to its threshold. Further, when the number of jobs is low (e.g., 100 and 300), our method can find a reliable schedule even without scaling operation. However, when the number of jobs is somewhat larger (e.g., 500, 700, and 875). Our method cannot find a reliable schedule without scaling

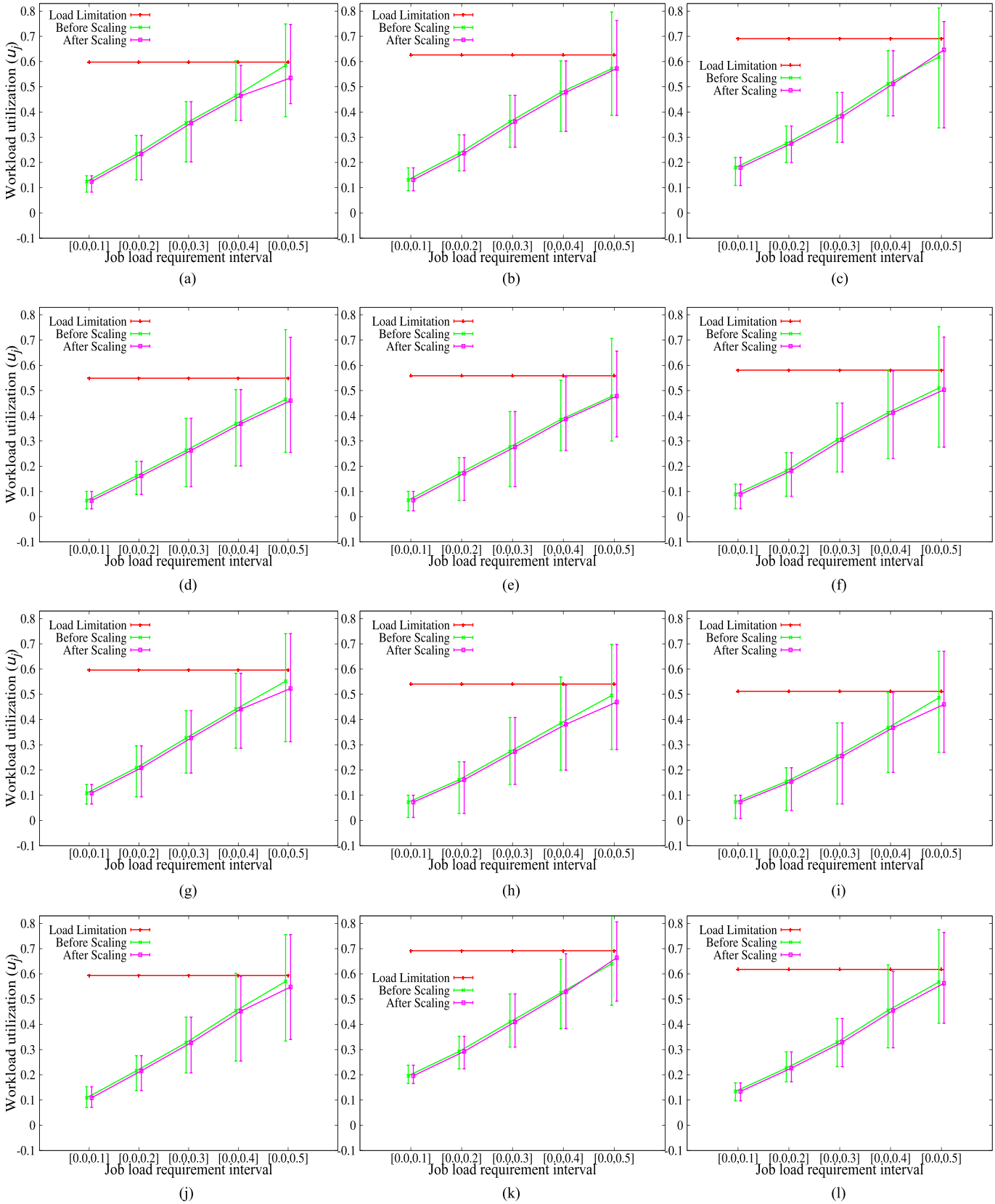

Fig. 10. Another set of results with varied resource requirement interval. (a) M2. (b) M7. (c) M13. (d) M21. (e) M23. (f) M27. (g) M32. (h) M36. (i) M38. (j) M41. (k) M48. (l) M50.

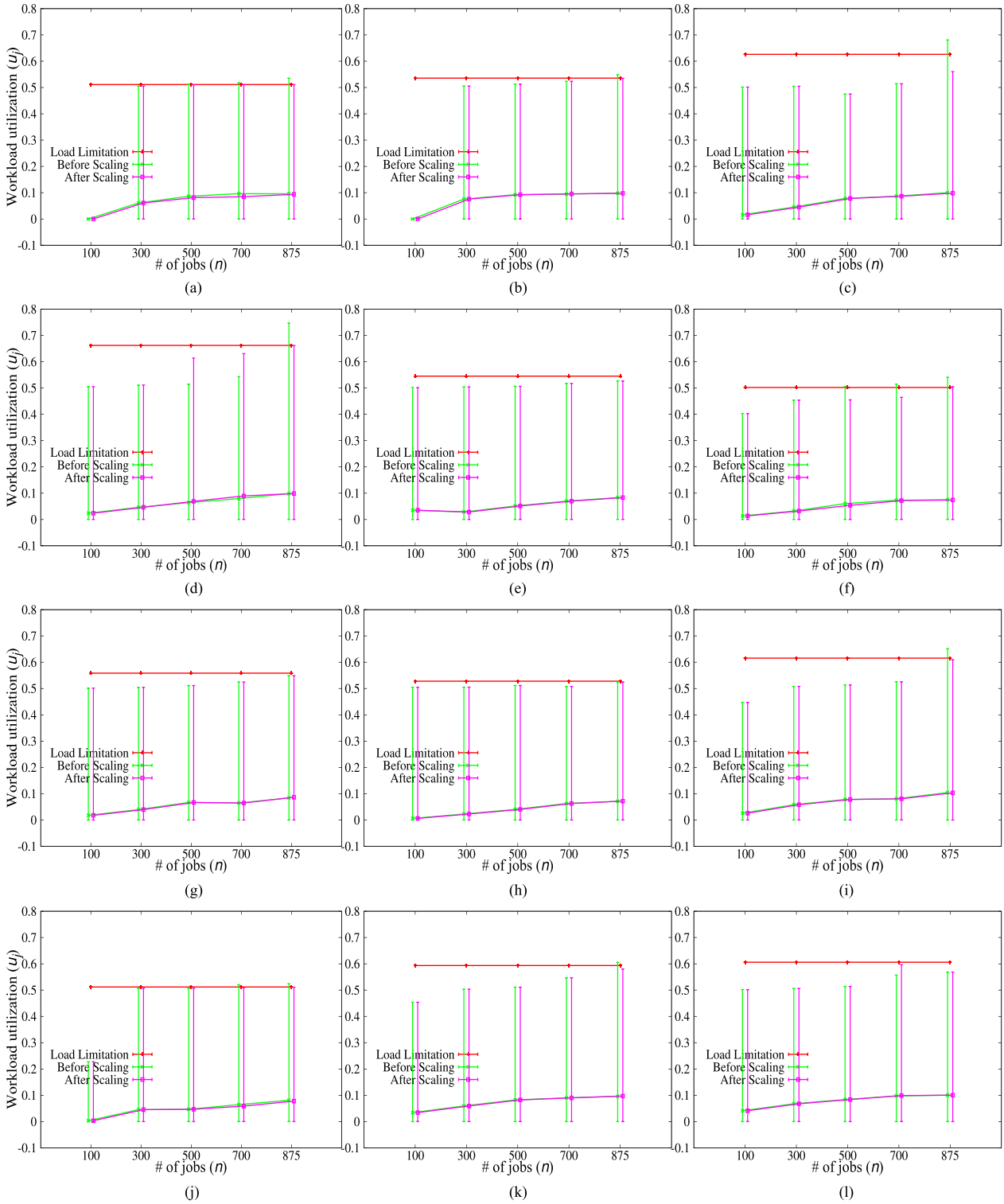

Fig. 11. Another set of results on real job characteristics with varied number of jobs. (a) M1. (b) M3. (c) M7. (d) M12. (e) M17. (f) M20. (g) M23. (h) M30. (i) M35. (j) M38. (k) M41. (l) M49.

operation [e.g., Fig. 11(a)–(d), (f), etc.]. The reason lies in that the schedule returned by **Bipartite-Stereogram-Matching** (without scaling operation) cannot guarantee that the workload on each server is less than or equal to the server's threshold. Nevertheless, from Fig. 11, we can also observe that the exceeds on servers are very small, which verifies the property of **Bipartite-Stereogram-Matching** shown in Theorem III.2. By doing the scaling operation, the maximal workloads on some servers are reduced to be less than their thresholds, while those maximal workloads on some other servers may be increased

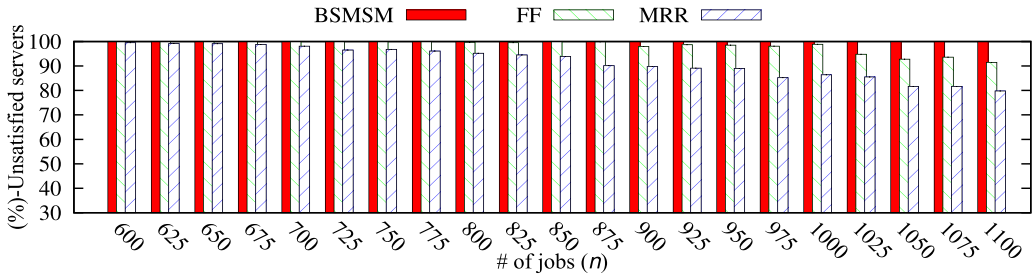

Fig. 12. Comparison results among BSMSM, first fit (FF), and modified round robin (MRR).

[see Fig. 11(c) and (l)]. Nevertheless, the increased maximal workloads are also guaranteed to be less than or equal to the thresholds of the corresponding servers.

We also compare our method with others. Indeed, there are many previous works involving reliability [4], [37]–[47], and most of them [38], [39], [41], [44]–[47] involve exponential function evaluation for reliability. However, their methods and the corresponding metrics cannot be applied to our situation due to the following reasons.

1) In this paper, the reliability of a server is negatively impacted by its workload due to the investigation shown in Section 1.1 (Motivation). Further, the reliability considered in this paper is related to hard errors, i.e., the errors which can result in the crash of the server, rather than software errors. The metrics (especially the exponential metrics) and the corresponding methods presented in the above works can not be applied to our situation.
2) Due to the negative impact of workload on a server's reliability, for each server, we set a workload threshold and assume that if the workload on the server does not exceed the threshold, the reliability target of the server is reached. Hence, in our situation, the reliability of a server is segmented rather than continuous, which enhances the hardness to solve our problem.

Due to the abovementioned reasons, we design our comparison metric as the number of servers whose reliability targets are achieved, i.e., the number of servers whose workloads are less than or equal to their corresponding workload thresholds. We compare our method with first fit (FF), i.e., for each job, find the first server whose workload is still guaranteed after accommodating the job, and modified round robin (MRR), which is modified according to original round robin (RR) idea, i.e., checking servers from the server next to the server accommodating the last allocated job. The only difference between MRR and RR lies in that when checking a server, if its workload threshold is no longer guaranteed, we move to the next server for checking. In both FF and MRR, if no one server can accommodate a job without violating its workload threshold, we randomly and uniformly select a server from the m servers and allocate the job to the selected server.

The comparison results are shown in Fig. 12. In Fig. 12, we present the percentage of servers whose reliability targets are achieved, i.e., whose workload thresholds are satisfied, with the increase of the number of jobs. From Fig. 12, we can observe that when the number of jobs is somewhat low (less than 675), all the three methods can guarantee all servers' reliability targets. However, with the increase of the number of jobs, the percentage of reliable servers obtained by MRR decreases. When the number of jobs becomes further larger (greater than 900), the result obtained by FF begins to drop from 100% and further decreases with the increase of the number of jobs. From Fig. 12, we can also observe that when the number of jobs reaches 1100, the result obtained by our proposed method can be better than those of FF and MRR by around 10% and 20%, respectively. That is to say, for the same HC, the schedule obtained by our method can guarantee the reliability targets of all servers while those of FF and MRR can result in 10% and 20% of servers whose reliability targets are not guaranteed. The reason lies in that we consider job allocation for reliability objective from an overall view and involve the coalition between jobs and servers.

V. Conclusion

Considering the fact that the reliability of a server tends to decrease with the increase of its workload, in this paper, we involved load impacts on service reliability in an HC. Our goal was to find a reliable schedule of jobs to servers such that all servers' workload limitations are satisfied. In this paper, we constructed a reliability model in which each sever had its own workload limitation and formulated the corresponding RELIABLE-SCHED problem. To solve the problem, we accordingly developed a BSMSM, in which we viewed the job–machine mapping as a matching between them. We theoretically proved that the solution obtained by our method can guarantee that the workload on each of the servers cannot exceed its threshold by one job's resource requirement. We also performed extensive random experiments to verify the feasibility of our method.

As part of future directions, we will configure the multiple servers dynamically and study the relationship between the resource demands and the servers. Another direction is to study the minimal cost of a set of selected servers to accommodate a set of services, i.e., service allocation with minimal server cost.

Acknowledgment

The authors would like to thank the Associate Editor and the anonymous reviewers for their comments and suggestions which have significantly improved the quality of the paper.

References

[1] K. Chard and K. Bubendorfer, "Co-operative resource allocation: Building an open cloud market using shared infrastructure," *IEEE Trans. Cloud Comput.*, vol. 7, no. 1, pp. 183–195, Jan./Mar. 2019.

[2] B. Mao, S. Wu, and H. Jiang, "Improving storage availability in cloud-of-clouds with hybrid redundant data distribution," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 633–642.
[3] M. Adams *et al.*, "An introduction to designing reliable cloud services," Microsoft Corporation, Redmond, WA, USA, 2014.
[4] P. Sun, Y. Dai, and X. Qiu, "Optimal scheduling and management on correlating reliability, performance, and energy consumption for multiagent cloud systems," *IEEE Trans. Rel.*, vol. 66, no. 2, pp. 547–558, Jun. 2017.
[5] W. Deng, H. Jin, X. Liao, F. Liu, L. Chen, and H. Liu, "Lifetime or energy: Consolidating servers with reliability control in virtualized cloud datacenters," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci.*, 2012, pp. 18–25.
[6] E. Bauer and R. Adams, *Reliability and Availability of Cloud Computing*. Hoboken, NJ, USA: Wiley, 2012.
[7] C. Liu, K. Li, and K. Li, "A game approach to multi-servers load balancing with load-dependent server availability consideration," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2018.2790404.
[8] A. Zhou, S. Wang, Z. Zheng, C.-H. Hsu, M. R. Lyu, and F. Yang, "On cloud service reliability enhancement with optimal resource usage," *IEEE Trans. Cloud Comput.*, vol. 4, no. 4, pp. 452–466, Oct./Dec. 2016.
[9] J. Zhang, X. Lu, and D. K. Panda, "High performance MPI library for container-based HPC cloud on infiniband clusters," in *Proc. 45th Int. Conf. Parallel Process.*, 2016, pp. 268–277.
[10] C. Liu, K. Li, C. Xu, and K. Li, "Strategy configurations of multiple users competition for cloud service reservation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 508–520, 2016.
[11] K. Li, C. Liu, K. Li, and A. Y. Zomaya, "A framework of price bidding configurations for resource usage in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2168–2181, Aug. 2016.
[12] R. K. Iyer, S. E. Butner, and E. J. McCluskey, "A statistical failure/load relationship: Results of a multicomputer study," *IEEE Trans. Comput.*, vol. C-31, no. 7, pp. 697–706, Jul. 1982.
[13] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 337–350, Oct./Dec. 2010.
[14] *SPEC CPU2006, Standard Performance Evaluation Corporation*, Gainesville, VA, USA, 2006. [Online]. Available: http://www.spec.org/cpu2006/
[15] [Online]. Available: http://www.lm-sensors.org
[16] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *Proc. Int. Symp. Comput. Architecture*, 2004, pp. 276–287.
[17] K. Swaminathan, N. Chandramoorthy, C. Y. Cher, R. Bertran, and P. Bose, "Bravo: Balanced reliability-aware voltage optimization," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2017, pp. 97–108.
[18] D. I. Heimann, N. Mittal, and K. S. Trivedi, "Availability and reliability modeling for computer systems," *Adv. Comput.*, vol. 31, pp. 175–233, 1990.
[19] O. Beaumont, L. Eyraud-Dubois, and H. Larchevêque, "Reliable service allocation in clouds," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, 2013, pp. 55–66.
[20] K. Ferreira *et al.*, "Evaluating the viability of process replication reliability for exascale systems," in *Proc. IEEE Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 1–12.
[21] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, Art. no. 33.
[22] F. Cappello, H. Casanova, and Y. Robert, "Checkpointing vs. migration for post-petascale supercomputers," in *Proc. IEEE 39th Int. Conf. Parallel Process.*, 2010, pp. 168–177.
[23] A. Zhou *et al.*, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Trans. Services Comput.*, vol. 10, no. 6, pp. 902–913, Nov./Dec. 2017.
[24] W. Li, Y. Yang, and D. Yuan, "Ensuring cloud data reliability with minimum replication by proactive replica checking," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1494–1506, May 2016.
[25] C. Liu, K. Li, and K. Li, "Minimal cost server configuration for meeting time-varying resource demands in cloud centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 11, pp. 2503–2513, Nov. 2018.
[26] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue, "Survivable virtual infrastructure mapping in virtualized data centers," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 196–203.
[27] A. Usman, R. Schadek, and O. Theel, "A novel highly available data replication strategy exploiting data semantics, coding techniques and prior at-hand knowledge," in *Proc. IEEE 22nd Pacific Rim Int. Symp. Dependable Comput.*, 2017, pp. 301–310.
[28] Z. Wang, L. Sun, M. Zhang, H. Pang, E. Tian, and W. Zhu, "Propagation- and mobility-aware D2D social content replication," *IEEE Trans. Mobile Comput.*, vol. 16, no. 4, pp. 1107–1120, Apr. 2017.
[29] M. Zhang, H. Jin, X. Shi, and S. Wu, "VirtCFT: A transparent VM-level fault-tolerant system for virtual clusters," in *Proc. IEEE 16th Int. Conf. Parallel Distrib. Syst.*, 2010, pp. 147–154.
[30] Í. Goiri, F. Juli, J. Guitart, and J. Torres, "Checkpoint-based fault-tolerant infrastructure for virtualized service providers," in *Proc. IEEE Netw. Oper. Manage. Symp.*, 2010, pp. 455–462.
[31] N. Limrungsi, J. Zhao, Y. Xiang, T. Lan, H. H. Huang, and S. Subramaniam, "Providing reliability as an elastic service in cloud computing," in *Proc. IEEE Int. Conf. Commun.*, 2012, pp. 2912–2917.
[32] G. Levitin, L. Xing, Y. Dai, and V. M. Vokkarane, "Dynamic checkpointing policy in heterogeneous real-time standby systems," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1449–1456, Aug. 2017.
[33] S. Ghosh and A. H. Gebremedhin, "Parallelization of bin packing on multicore systems," in *Proc. IEEE 23rd Int. Conf. High Perform. Comput.*, 2016, pp. 311–320.
[34] D. B. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem," *Math. Program.*, vol. 62, no. 1/3, pp. 461–474, 1993.
[35] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logistics*, vol. 2, no. 1/2, pp. 83–97, 1955.
[36] A. Roy, R. Ganesan, and S. Sarkar, "Keep it moving: Proactive workload management for reducing SLA violations in large scale SAAS clouds," in *Proc. IEEE Int. Symp. Softw. Rel. Eng.*, 2014, pp. 421–430.
[37] A. B. M. B. Alam, M. Zulkernine, and A. Haque, "A reliability-based resource allocation approach for cloud computing," in *Proc. IEEE Int. Symp. Cloud Service Comput.*, 2017, pp. 249–252.
[38] L. Zhang, K. Li, Z. Wen, C. Peng, and K. Li, "Contention-aware reliability management scheme for parallel tasks scheduling in heterogeneous computing systems," in *Proc. Green Sustain. Comput. Conf.*, 2017, pp. 1–6.
[39] H. Youness, A. Omar, and M. Moness, "Fault tolerant heterogeneous MPSOC schedule length minimization based on platform reliability," in *Proc. Japan-Egypt Int. Conf. Electron., Commun. Comput.*, 2014, pp. 88–93.
[40] H. Lee and M. A. A. Faruque, "GPU architecture aware instruction scheduling for improving soft-error reliability," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 3, no. 2, pp. 86–99, Apr./Jun. 2017.
[41] S. Guo, H. Z. Huang, Z. Wang, and M. Xie, "Grid service reliability modeling and optimal task scheduling considering fault recovery," *IEEE Trans. Rel.*, vol. 60, no. 1, pp. 263–274, Mar. 2011.
[42] A. Naithani, S. Eyerman, and L. Eeckhout, "Optimizing soft error reliability through scheduling on heterogeneous multicore processors," *IEEE Trans. Comput.*, vol. 67, no. 6, pp. 830–846, Jun. 2018.
[43] Y. Liu, R. Li, and Q. Li, "Reliability analysis of cloud computing systems with different scheduling strategies under dynamic demands," in *Proc. Int. Conf. Inf. Sci. Control Eng.*, 2017, pp. 1108–1113.
[44] W. Abdulal and S. Ramachandram, "Reliability-aware genetic scheduling algorithm in grid environment," in *Proc. Int. Conf. Commun. Syst. Netw. Technol.*, 2011, pp. 673–677.
[45] T. Wei, X. Chen, and S. Hu, "Reliability-driven energy-efficient task scheduling for multiprocessor real-time systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 10, pp. 1569–1573, Oct. 2011.
[46] I. Assayad, A. Girault, and H. Kalla, "Scheduling of real-time embedded systems under reliability and power constraints," in *Proc. IEEE Int. Conf. Complex Syst.*, 2012, pp. 1–6.
[47] C. Chen, "Task scheduling for maximizing performance and reliability considering fault recovery in heterogeneous distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 521–532, Feb. 2016.

Chubo Liu received the B.S. and Ph.D. degrees in computer science and technology from Hunan University, Changsha, China, in 2011 and 2016, respectively.

He is currently an Associate Professor of Computer Science and Technology with Hunan University. He has published over ten papers in journals, such as the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON CLOUD COMPUTING, *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, and *Theoretical Computer Science*. His research interests include game theory, approximation and randomized algorithms, cloud and edge computing.

Kenli Li received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2003.

From 2004 to 2005, he was a Visiting Scholar with the University of Illinois at Urbana-Champaign, Champaign, IL, USA. He is currently the Dean and a Full Professor of Computer Science and Technology with Hunan University and the Deputy Director of the National Supercomputing Center, Changsha, China. He has published more than 160 research papers in international conferences and journals, such as the IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON SIGNAL PROCESSING, *Journal of Parallel and Distributed Computing*, ICPP, and CCGrid. His major research interests include parallel computing, high-performance computing, grid and cloud computing.

Prof. Li is an Editorial Board Member of the IEEE TRANSACTIONS ON COMPUTERS. He is a Senior Member and an outstanding member of the CCF.

Jie Liang received the B.S. degree in communication engineering from Henan Normal University, Xinxiang, China, in 2011, and the M.S. degree in information and communication engineering, in 2014 from Hunan University, Changsha, China, where she is currently working toward the Ph.D. degree.

Her research interests include modeling and scheduling of distributed computing systems, optimization and parallel algorithms, game theory, grid and cloud computing.

Keqin Li is a SUNY Distinguished Professor of Computer Science. He has authored coauthored more than 630 journal articles, book chapters, and refereed conference papers. His current research interests include parallel computing, high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of Things, and cyber-physical systems.

Prof. Li has received several Best Paper Awards. He is currently serving or has served on the editorial boards of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING.