Multi-Resource Partial-Ordered Task Scheduling in Cloud Computing

Chaokun Zhang*, Yong Cui*, Rong Zheng†, Jinlong E*, and Jianping Wu*
*Tsinghua University, Beijing 100084, P. R. China
†McMaster University, Hamilton, ON, Canada

***Abstract*—In this paper, we investigate the scheduling problem with multi-resource allocation in cloud computing environments. In contrast to existing work that focuses on flow-level scheduling, which treats flows in isolation, we consider dependency among subtasks of applications that imposes a partial order relationship in execution. We formulate the problem of Multi-Resource Partial-Ordered Task Scheduling (MR-POTS) to minimize the makespan. In the first stage, the proposed Dominant Resource Priority (DRP) algorithm decides the collection of subtasks for resource allocation by taking into account the partial order relationship and characteristics of subtasks. In the second stage, the proposed Maximum Utilization Allocation (MUA) algorithm partitions multiple resources among selected subtasks with the objective to maximize the overall utilization. Both theoretical analysis and experimental evaluation demonstrate the proposed algorithms can approximately achieve the minimal makespan with high resource utilization. Specifically, a reduction of 50% in makespan can be achieved compared with existing scheduling schemes.**

I. Introduction

The emergence of the clouds [1] plays an important role in facilitating many complex tasks, such as search query, data analytic and interactive perception. A **task** refers to a work unit of an application from the beginning of its execution to the final response returned to a waiting user [2]. A typical cloud computing task involves a number of subtasks, where the onset of one subtask may depend on the completion of other subtasks. The execution of a subtask may incur multiple resources such as CPU, memory, and network bandwidth, and has its unique resource demand characteristics. Therefore, it is important to manage cloud resources efficiently to facilitate tasks with complex dependency relations.

Many existing scheduling algorithms either focus on flow scheduling and queueing [3], [4], or treat the subtasks in a sequential manner [5], [6]. Among scheduling algorithms that consider multiple resources, the main objective is to provide fairness and resource isolation. For example, DRF [7] proposes the concept of *dominant resource* and allocates resources among jobs based on their dominant resources. DRFQ [5] and subsequent algorithms such as GMR^3 [6] provide fair queueing among flows. While these flow scheduling algorithms target per-flow fairness, they tend to ignore the dependency among flows. This may lead to prolonged task completion time [2].

Designing optimal schedules for tasks consisting of dependent subtasks with multi-resource requirements is non-trivial. First, in the presence of multiple resources, there typically exists a trade-off between fairness and the completion time of tasks. *Task Serialization* (TS) strategies, such as FIFO-LM [2], service tasks one at a time. It helps finishing tasks sooner but if naively extended to multi-resource scenarios, may result in low utilization and starvation. In contrast, *Fair Sharing* (FS) strategies, such as DRF [7], service multiple tasks in parallel by allocating a portion of resources to each. FS can provide an abstraction of resource isolation but may lead to prolonged worst-case task completion time (called *makespan*). As a result, it is desirable to design task scheduling algorithms that minimize the makespan of tasks while avoiding starvation.

Second, the dependency among subtasks within a task further complicates resource allocation decisions. On one hand, breaking down a task into subtasks allows more fine-grained resource sharing. On the other hand, the dependency limits the schedulability of subtasks and consequently affects the completion of subtasks. Worse still, existing scheduling solutions such as Critical Path method (CPM) [8] that consider precedence constraints, typically assume the processing times of subtasks are known in advance and remain fixed. This assumption is no longer valid in presence of multiple resources, where resource sharing necessitates scheduling-dependent processing times.

In this paper, we break down the workflow of a task into subtasks and model their dependency using **partial order** represented by a Directed Acyclic Graph (DAG). A *Multi-Resource Partial-Ordered Task Scheduling* (MR-POTS) problem is formulated to minimize the makespan and is shown to be NP-hard. As a result, polynomial time heuristic algorithms are needed. We devise a two-stage scheduling policy for MR-POTS. In the first stage, the proposed *Dominant Resource Priority* (DRP) algorithm decides the collection of subtasks for resource allocation by taking into account the partial order relationship and characteristics of subtasks. In the second stage, we propose the *Maximum Utilization Allocation* (MUA) algorithm to partition multiple resources

among selected subtasks with the objective to maximize the overall resource utilization.

To the best of our knowledge, this is the first work that considers partial-ordered task scheduling with multi-resource requests in cloud computing environments. Rigorous analysis shows that the proposed algorithms have low time complexity and high resource utilization. Experimental evaluation demonstrates that the proposed algorithms have superior performance compared with existing scheduling algorithms that do not explicitly consider partial order relation among subtasks and/or multi-resource requirements. Specifically, a reduction of 50% in makespan can be achieved over existing scheduling schemes.

II. Problem Formulation

Let $\mathbb{T}$ be the set of tasks to be scheduled. A task $i \in \mathbb{T}$ can be divided into a collection of subtasks $\{(i,j)\}$. Consider a finite set of resource types $\mathbb{R}$. Let c^r be the capacity of resource r ($r \in \mathbb{R}$). The resource demands of the jth subtask of the ith task are denoted as $\{d^r_{i,j}, r \in \mathbb{R}\}$. Here, a demand for a resource type r is the **total amount** of resource r needed to complete a subtask [9]. For instance, when a subtask has a resource demand of 10 CPU cycles and is allocated 2 CPUs at a time. It thus takes 5 time slots to complete the subtask. A resource, denoted by $r^*_{i,j}$ is considered the *dominant resource* for subtask (i,j) if $r^*_{i,j} = \arg\max_{r\in\mathbb{R}} \frac{d^r_{i,j}}{c^r}$. Correspondingly, we define $c^*_{i,j}$ as the capacity of $r^*_{i,j}$ and $d^*_{i,j}$ as the demand of subtask (i,j) for the (dominant) resource.

Now, we model the partial order relation among all subtasks of a task as a DAG (see Figure 1). In DAG, each node is associated with a subtask and its resource demands, and each edge represents the dependency between two tasks. The DAG captures the predecessors and successors of a subtask. Specifically, $\Gamma^-_i(j)$ ($\Gamma^+_i(j)$) denotes the predecessor (successor) set of subtask (i,j). A subtask (i,j) cannot be scheduled until the completion of all its predecessors in $\Gamma^-_i(j)$; a subtask is considered to start the moment it is eligible, even if resources are not yet allocated to it immediately. Let $\mathbb{S}$ be the set of all such currently eligible subtasks, and $|\mathbb{S}|$ be the size of $\mathbb{S}$.

For task i, denote its start time and completion time by t^S_i and t^C_i, respectively. The processing time of task i is thus $t^P_i = t^C_i - t^S_i$. Similarly, the start, completion, and processing time of a subtask (i,j) are respectively $t^S_{i,j}$, $t^C_{i,j}$, and $t^P_{i,j}$, where $t^P_{i,j} = t^C_{i,j} - t^S_{i,j}$. Finally, we consider multi-resource task scheduling that operates in a slotted system. Let $x^r_{i,j}(t)$ be the amount of resource r allocated to subtask (i,j) in slot t.

Constraints: In the MR-POTS problem, there are several constraints. First, due to the partial order relation-

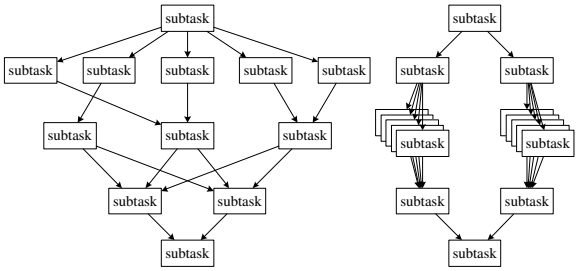


Fig. 1. Two instances of partial-ordered task topology.

ship among the tasks, the following equations should hold:

$$t^S_{i,j} = \begin{cases} \max\limits_{j^-\in\Gamma^-_i(j)} t^C_{i,j^-} & \Gamma^-_i(j) \neq \emptyset \\ t^S_i & \Gamma^-_i(j) = \emptyset \end{cases} \quad \forall (i,j) \in \mathbb{S}. \tag{1}$$

Second, it is easy to see that,

$$\begin{cases} x^r_{i,j}(t) = 0 & t \notin [t^S_{i,j}, t^C_{i,j}) \\ x^r_{i,j}(t) \geq 0 & t \in [t^S_{i,j}, t^C_{i,j}) \end{cases} \quad \forall (i,j) \in \mathbb{S}, r \in \mathbb{R}. \tag{2}$$

Third, in task scheduling, the resource allocation for all subtasks cannot exceed the total capacity of the respective resource at any time, that is,

$$\sum_{(i,j)\in\mathbb{S}} x^r_{i,j}(t) \leq c^r \quad \forall r \in \mathbb{R}, t \geq 0. \tag{3}$$

Fourth, the resource allocation for subtask (i,j) is no more than the demand for subtask (i,j) at any time, i.e, $x^r_{i,j}(t) \leq d^r_{i,j}$. Furthermore, the accumulative allocation $x^r_{i,j}(t)$ of resource r for subtask (i,j) over time is equal to its demand $d^r_{i,j}$, that is,

$$\sum_{t=t^S_{i,j}}^{t^C_{i,j}} x^r_{i,j}(t) = d^r_{i,j} \quad \forall (i,j) \in \mathbb{S}, r \in \mathbb{R}. \tag{4}$$

Finally, similar to DRF [7], we assume that the allocation to a subtask (i,j) of each resource type is proportional to the respectively demand, namely,

$$\frac{x^r_{i,j}(t)}{d^r_{i,j}} = \frac{x^{r'}_{i,j}(t)}{d^{r'}_{i,j}} \quad \forall (i,j) \in \mathbb{S}, t \geq 0, r \neq r', \tag{5}$$

for all $r, r' \in \mathbb{R}$ where $d^r_{i,j}$ and $d^{r'}_{i,j}$ are greater than zero.

Objective: Our objective is to minimize the maximum tail completion time of tasks. Denote the makespan of all tasks by $\mathcal{D}$. Formally, $\mathcal{D} = \max_{i\in\mathbb{T}} t^C_i$. We are now in the position to formulate the MR-POTS problem.

$$\begin{aligned} \mathbf{P1}: \quad & \min \quad \mathcal{D} \\ & \text{s.t.} \quad \text{Constraints (1) - (5).} \end{aligned} \tag{6}$$

MR-POTS is a min-max optimization problem with non-convex constraints, and can be proved NP-hard by a reduction from PCS problem (See [SS9] in [10]).

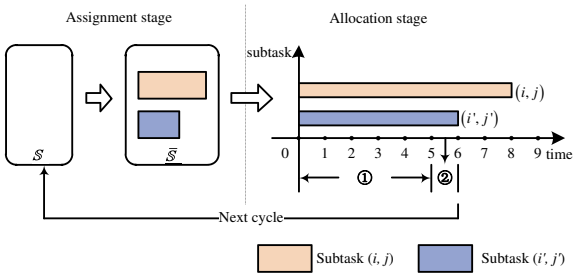


Fig. 2. Two stages of scheduling process, i.e., assignment stage and allocation stage.

III. Algorithm Design

In this section, we focus on developing a polynomial-time heuristic algorithm that achieves low makespan. The algorithm is executed when there is a new task arrival or a subtask completion. Each cycle of a task scheduling decision is made in two stages. The first stage, the *assignment stage*, selects suitable subtasks to be scheduled. In selecting subtasks, the key observation is that when two subtasks have an identical dominant resource, concurrently scheduling them would lead to under-utilization of the resources. Therefore, it is desirable to select tasks with non-conflicting dominant resources. Furthermore, the assignment stage also needs to take into account the starting time and the workload of subtasks to avoid starvation. In the second stage, the *allocation stage*, the resources are allocated to the selected subtasks with the objective of maximizing the total resource utilization. Due to distinctive dominant resource requirements among subtasks selected from the first stage, maximizing resource utilization tends to minimize the makespan. The two-stage process is illustrated in Figure 2.

A. Resource Allocation with Maximum Utilization

We first consider the resource allocation problem among subtasks with *non-conflicting dominant resources* in the allocation stage. This condition is guaranteed by the assignment stage to be discussed next subsection. Let $\bar{\mathbb{S}} \subseteq \mathbb{S}$ be the set of subtasks to be scheduled, and $|\bar{\mathbb{S}}|$ be the cardinality of $\bar{\mathbb{S}}$. Formally, the non-conflicting dominant resource constraint is given by

$$r^*_{i,j} \neq r^*_{i',j'}, \forall (i,j),(i',j') \in \bar{\mathbb{S}}. \tag{7}$$

We further divide the resource allocation stage into the *stable period* (See ① in Figure 2), where the scheduled subtask sets and their resource allocations $x^r_{i,j}(t)$ remain the same; and the *sprint period* (See ② in Figure 2), where the subtask with the least remaining resource demand that can be satisfied within one slot will be scheduled and completed in the slot. The rationale behind the division of the stable and the sprint periods is that it can reduce the tail completion time.

Algorithm 1 MUA Algorithm

Input: $d^r_{i,j}, c^r, CurrentTime, \ \forall (i,j) \in \bar{\mathbb{S}}, r \in \mathbb{R}$
Output: $x^r_{i,j}(t)$, $t^C_{i,j}$
1: Solve P2 for $k_{i,j}$ $\forall (i,j) \in \bar{\mathbb{S}}$;
2: $\Delta t = \min_{(i,j)} \lceil (d^*_{i,j} - c^*_{i,j})/(k_{i,j} d^*_{i,j}) \rceil$;
3: **for all** $(i,j) \in \bar{\mathbb{S}}$ **do**
4: $\quad x^r_{i,j}(t) = k_{i,j} d^r_{i,j} \ \ \forall r \in \mathbb{R}$ $t \in$ stable period;
5: $\quad d^r_{i,j} = d^r_{i,j} - x^r_{i,j}(t) \cdot \Delta t \ \ \forall r \in \mathbb{R}$ $t \in$ stable period;
6: **end for**
7: $t = CurrentTime + \Delta t$;
8: **for all** $(i,j) \in \bar{\mathbb{S}}$ **do**
9: $\quad x^r_{i,j}(t) = (1 - \lceil (d^*_{i,j} - c^*_{i,j})/(k_{i,j} d^*_{i,j}) \rceil k_{i,j}) d^r_{i,j} \ \ \forall r \in \mathbb{R}$ $t \in$ sprint period;
10: $\quad d^r_{i,j} = d^r_{i,j} - x^r_{i,j}(t) \ \ \forall r \in \mathbb{R}$ $t \in$ sprint period;
11: $\quad$ **if** $d^*_{i,j} == 0$ **then**
12: $\quad\quad \Theta \leftarrow (i,j)$;
13: $\quad$ **end if**
14: **end for**
15: $t^C_{i,j} = ++t$;
16: **return** $x^r_{i,j}(t)$, $t^C_{i,j}$, Θ $\forall t \in [CurrentTime, t^C_{i,j}]$;

In the stable period, from (5), we define $k_{i,j}(t) = \frac{x^r_{i,j}(t)}{d^r_{i,j}}, \forall (i,j) \in \bar{\mathbb{S}}$. Since resource allocation remains constant during the stable period, we can drop the index t. In other words, $k_{i,j}$ is the percentage of the demands of subtask (i,j) satisfied in one slot. Thus, (3) can be rewritten as

$$\sum_{(i,j) \in \bar{\mathbb{S}}} \frac{d^r_{i,j}}{c^r} k_{i,j} \leq 1, \forall r \in \mathbb{R}. \tag{8}$$

We now formulate the maximum total utilization problem for the stable period as follows,

$$\mathbf{P2}: \quad \max \ \ \mathcal{U} = \sum_{\substack{(i,j) \in \bar{\mathbb{S}} \\ r \in R}} \left(\frac{d^r_{i,j}}{c^r} k_{i,j} \right)$$
$$\text{s.t.} \quad \text{Constraint (8)}$$
$$0 \leq k_{i,j} \leq 1 \qquad \forall (i,j) \in \bar{\mathbb{S}}. \tag{9}$$

Obviously, P2 is a linear programming problem (LP) with $|\bar{\mathbb{S}}|$ unknowns and can be solved using a LP solver.

Algorithm 1 summarizes the aforementioned resource allocation process called the Maximum Utilization Allocation (MUA) algorithm. During the stable period (Line 3–6), eligible subtask will be allocated resources to ensure the maximum utilization. During the sprint period (Line 8–14), the subtasks that can complete within one slot will be allocated the needed resources one by one until no more subtask can be scheduled. The set Θ denotes all finished subtasks (Line 12).

B. Task Assignment with Dominant Resource Priority

In MUA, we require that all subtasks have no conflicting dominant resources. Additionally, it is well known that a scheduling policy with the maximum resource utilization may lead to starvation of certain tasks [11]. Thus, the main objective of the assignment stage is to select the proper set of subtasks, and to eventually give each subtask scheduling opportunity. The decision

of task assignment is further complicated by partial order relations among subtasks. In some cases, it may be beneficial to first schedule a certain subtask whose successors allow high utilization of resources.

To address the above problem, we propose Dominant Resource Priority (DRP). It first groups subtasks based on their dominant resources. Then, the scheduler assigns a priority value to each subtask, where a smaller priority value means higher priority. The subtask with the smallest priority will be added to $\bar{\mathbb{S}}$ followed by a subtask that has the next smallest priority and does not have conflicting dominant resources among the remaining eligible subtasks. Figure 3 illustrates the execution of the algorithm discussed in more details next.

1) Grouping by Dominant Resource: In DRP, we first divide all schedulable subtasks and successor subtasks into groups according to their dominant resources, shown in Step ① of Figure 3. Subtasks in each group with the same dominant resource will be scheduled serially. Since there are $|\mathbb{R}|$ types of resources and we can only schedule one subtask per group, the number of subtasks that can be simultaneously serviced is at most $|\mathbb{R}|$, i.e., $|\bar{\mathbb{S}}| \leq |\mathbb{R}|$.

2) Inner-Group Subtask Priority Setting: After the subtasks have been divided into different groups, the next question is how to decide the scheduling order for subtasks in each group. Intuitively, if two subtasks start at different times, First-Come First-Served (FCFS) is preferred; and if two subtasks have different processing times on their dominant resources, Shortest Task First (STF) is preferred. In other words, both the start time and the resource demands of subtasks should be taken into account in the scheduling order. To do so, we propose the following DRP metric,

$$DRP(i,j) = t^S_{i,j} + \frac{d^*_{i,j}}{c^*_{i,j}}, \tag{10}$$

where $d^*_{i,j}/c^*_{i,j}$ corresponds to the shortest processing time of subtasks (i,j) if it uses its dominant resource exclusively. $DRP(i,j)$ is similar to the notion of virtual time [12] that measures the amount of workload [3], and has been adopted by various scheduling policies [5]. The subtask with the smallest DRP will be scheduled first.

Unlike DRFQ in middlebox [5] that considers homogeneous tasks, the resource demands of successor subtasks can be different from the current subtask. A successor subtask may have the smallest DRP in its corresponding group (see the rightmost dashed box of resource $r3$ in Figure 3). It is desirable to schedule its predecessor subtasks sooner to shorten the overall tail task completion time (Step ② in Figure 3).

Unfortunately, (10) is only applicable to subtasks with a known start time. We cannot determine the start time of a subtask until its predecessor subtasks are completed. Therefore, we propose a recursive approach to approximate DRP. Denote $\hat{t}^S_{i,j^+}$ the estimated start time for the successor subtask (i,j^+). $\hat{t}^S_{i,j^+}$ can be estimated from the completion time of its predecessors as

$$\hat{t}^S_{i,j^+} = \max_{j' \in \Gamma_i^-(j^+)} \left(t^S_{i,j'} + \frac{d^*_{i,j'}}{c^*_{i,j'}} \right) \quad \forall j^+ \in \Gamma_i^+(j). \tag{11}$$

The DRP of the successor subtask (i,j^+) with dominant resource $r^*_{i,j}$ can then be approximated by

$$DRP(i,j^+) = \hat{t}^S_{i,j^+} + \frac{d^*_{i,j^+}}{c^*_{i,j^+}} \quad \forall j^+ \in \Gamma_i^+(j). \tag{12}$$

Once the DRPs of successor subtasks are computed, for each resource group, if the DRPs of the top two subtasks differ no more than 1 slot, and the successor subtask of the 2nd best subtask has the smallest DRP value in its respective resource group, we will include the 2nd best subtask in the set of subtasks to be scheduled using MUA. Otherwise, the subtask with the lowest DRP value will be selected.

The DRP algorithm is summarized in Algorithm 2. Upon the arrival of a task, the scheduler computes its subtasks' DRPs according to (10) and (12), and inserts the subtask into the task set $\mathbb{S}$ in the ascending order of the DRP value (Line 1). It then finds the subtask with the minimum DRP in each dominant resource group (Line 3). The scheduler next identifies whether there exist some successor subtasks with the smallest DRP in their dominant resource groups (Line 5–10). Line 8 indicates all selected subtasks (temporarily saved in Θ) will be included in $\bar{\mathbb{S}}$. If no successor subtask has the smallest DRP value, all subtasks in $\bar{\mathbb{S}}$ will be allocated resources (Line 11–13). MUA algorithm will next be called to decide the actual amount of resources allocated to each task in $\bar{\mathbb{S}}$ (Line 14). Finally, if all subtasks in $\mathbb{S}$ are completed, the objective $\mathcal{D}$ of **P1** is calculated according to $t^C_{i,j}$ (Line 17). Otherwise, DRP algorithm will continue to schedule the remaining subtasks in $\mathbb{S}$.

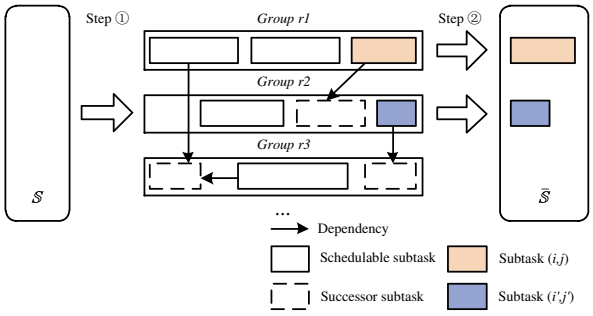


Fig. 3. DRP algorithm process description. ① Grouping by dominant resource; ② Selecting subtasks by the smallest DRP.

C. Properties of the DRP Algorithm

In this section, we analyze some key properties of the DRP algorithm. For clarity, all proofs are deferred to the appendices.

Algorithm 2 DRP Algorithm

Input: task i
Output: $\mathcal{D}$
$\mathbb{S} \leftarrow$ Insert each subtask (i,j) in task i according to $DRP(i,j)$;
while $\mathbb{S} \neq \emptyset$ **do**
  $\bar{\mathbb{S}} \leftarrow$ Get the eligible subtask with the minimum DRP value in each dominant resource group;
  $\Theta = \emptyset$;
  for all $(i,j) \in \bar{\mathbb{S}}$ **do**
    Get every subtask (i,j') with the minimum DRP value in each dominant resource group;
    if $\exists j' \in \Gamma_i^+(j)$ **then**
      $\Theta \leftarrow (i,j)$;
    end if
  end for
  if $\Theta \neq \emptyset$ **then**
    $\bar{\mathbb{S}} = \Theta$;
  end if
  $(x_{i,j}^r(t), t_{i,j}^C, \Theta) = \text{MUA}((d_{i,j}^r)_{|\bar{\mathbb{S}}|\times|\bar{\mathbb{S}}|}, c^r, CurrentTime)$ $t \in [CurrentTime, t_{i,j}^C]$;
  $\mathbb{S} = \mathbb{S} - \Theta$;
end while
Compute $\mathcal{D}$ according to $t_{i,j}^C$;
return $\mathcal{D}$;

Theorem 1. *The computation complexity of the DRP algorithm is* $\mathcal{O}(|\mathbb{S}||\mathbb{R}|^{log_2 7})$.

In practice, the number of resource types is small (usually only CPU, memory, link, etc.), so that the time complexity of $\mathcal{O}(|\mathbb{R}|^{log_2 7})$ only slightly impacts the scheduling performance. Therefore, we conclude that the complexity of DRP algorithm is linear to the size of schedulable subtasks. The next result characterizes the resource utilization of DRP.

Theorem 2. *There exists a lower bound for the utilization of resource allocation as follow,*

$$\mathcal{U} \geq \begin{cases} |\bar{\mathbb{S}}|/|\mathbb{R}| & \textit{stable period} \\ 1/|\mathbb{R}| & \textit{sprint period} \end{cases}. \tag{13}$$

On average,

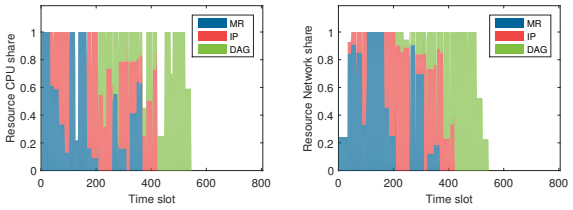
$$\mathcal{U} \geq \frac{1}{|\mathbb{R}|}\left(|\bar{\mathbb{S}}| - \left(|\bar{\mathbb{S}}| - 1\right) \min_{(i,j)\in\bar{\mathbb{S}}} \frac{c_{i,j}^*}{d_{i,j}^*}\right). \tag{14}$$

Theorem 2 indicates that DRP has high resource utilization, which is also corroborated by our simulation results in Section IV.

Theorem 3. *DRP ensures no starvation in MR-POTS.*

IV. Performance Evaluation

In the experiments, we consider three types of workloads, *MR*, *IP*, and *DAG*. The first workload is derived from the Google cluster data [13], which represents 29 days' worth of cell information of MapReduce from May 2011. The tasks in the workload have diverse resource demands with microsecond-level timestamps. Second, a Gestris-style workload [14] is constructed by the trace collected from running the interactive perception application, and its conceptual subtask structure



(a) CPU resource sharing (b) Network resource sharing

Fig. 4. Dynamics of resource allocation over time under DRP.

is shown in Figure 1 in Section II. We also generate synthetic partial-ordered tasks to produce the last workload. In this case, each task has a random number of subtasks with random amounts of demands following Poisson arrival processes. For the Baselines, we have implemented three baseline algorithms, namely, FIFO-LM [2] as a representative TS scheduling strategy, DRFQ [5] as a representative FS strategy, and CPM [8] which considers precedence constraints. In the case of multiple resources, we treat the dominant resource of a subtask as the input to FIFO-LM.

A. Adaption of Resource Allocation to Dynamic Workloads

To see how the resource allocation changes with dynamic workloads, we extract three instances of tasks, one for each type of workload, i.e., *MR*, *IP*, and *DAG*. The three tasks arrive at time slots 0, 30, and 150. The resource demands include CPU, memory, disk, and network resources, and resulting allocation for CPU and network resources are shown in Figure 4. Allocations for the other two resources have similar characteristics, and are thus omitted.

From the figure we observe that, when Task *IP* arrives at time slot 30, the resource allocation for *MR* task changes. The spikes (e.g., the first one at time slot 46) in resource share in Figure 4 mark the occurrence of the sprint period. Recall that a sprint period only lasts one time slot. While the total resource utilization in most cases is 100%, there are situations when the subtasks have conflicting dominant resources, the resources cannot be fully utilized. For example, from time slot 124 to 134, all the subtasks of *MR* enter into reduce phase, and all the subtasks of *IP* correspond to gesture matching; the subtasks with conflicting network resources thus lead to low CPU utilization.

It is also worth mentioning that, when Task *DAG* arrives at time slot 150, it is not allocated resources immediately. This is due to the fact that DRP tries to accelerate the completion of backlogged subtasks.

Figure 5 compares the average resource utilization over time for four scheduling policies. Additionally, we plot the lower bound from Theorem 2. From the figure, we find that the lower bound can indeed bound the performance of DRP. The utilization of DRP goes to zero

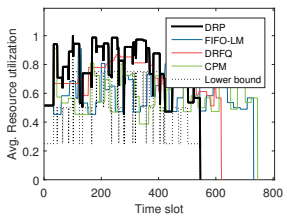
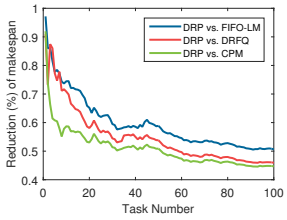
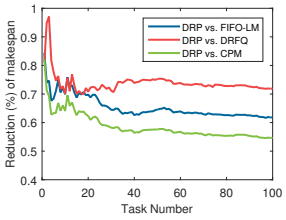


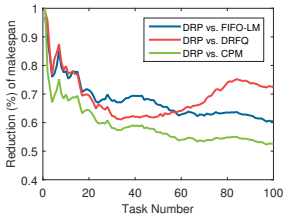
Fig. 5. Average resource utilization.



(a) Under *MR* workload



(b) Under *IP* workload



(c) Under *DAG* workload

Fig. 6. Makespan ratios under diverse workloads.

after 545 slots since all tasks have been completed. In general, we see that DRP has higher resource utilization than other schemes and shorter completion time. Again, the (downward) spikes in the DRP resource utilization are due to the sprint periods. This indicates that in some cases, a shorter processing time comes at the expense of intermittently decreased resource utilization.

B. *Efficiency*

In this experiment, we evaluate the makespan of DRP using trace data under the *MR*, *IP*, and *DAG* workloads. Figure 6(a) shows the ratio of makespans of DRP with respect to other schemes as the number of tasks increases under *MR*. We see that DRP contributes to a 40% to 50% reduction in makespan compared to FIFO-LM, and even higher reduction over DRFQ and CPM. As the number of tasks grows, DRP shows more reduction in makespan. This suggests that DRP can perform well in large-scale and heavy-load settings for the MapReduce applications.

Similar observations can be made under the *IP* and *DAG* workloads where DRP consistently achieves shorter makespan than that of other schemes (Figure 6(b) and Figure 6(c)). Interestingly, when comparing the performance of DRFQ and FIFO-LM, we find that FIFO-LM generally incurs smaller makespans than DRFQ in the *MR* tasks but higher makespans in the *IP* tasks. Under the *DAG* workload, the relation is not so clear-cut and depends on the number of tasks. In contrast, DRP combines TS and FS scheduling and manages to outperform both types of policies.

V. Conclusion

In this paper, we considered the problem of minimizing the makespan for partially ordered tasks with multi-resources demands. Towards this goal, the MUA and DRP algorithms were proposed for resource allocation and task assignment respectively. MUA and DRP account for the partial order relations among subtasks and try to get the best of both worlds of task serialization and fair sharing strategies. Both theoretical analysis and experimental evaluation showed that DRP has low makespan and high resource utilization.

Acknowledgment

This work is supported by the National High Technology Development 863 Program of China (2015AA015701), National Natural Science Foundation of China (61120106008 and 61422206), Tsinghua National Laboratory for Information Science and Technology (TNList), and National Science and Engineering Council of Canada under Discovery Grant and Ontario Center of Excellence.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. ACM SIGCOMM*, 2014, pp. 431–442.

[3] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proc. ACM SIGCOMM*, 1995, pp. 231–242.

[4] J. C. Bennett and H. Zhang, "WF^{2}Q: worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, 1996, pp. 120–128.

[5] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *Proc. ACM SIGCOMM*, 2012, pp. 1–12.

[6] W. Wang, B. Liang, and B. Li, "Low complexity multi-resource fair queueing with bounded delay," in *Proc. IEEE INFOCOM*, 2014, pp. 1914–1922.

[7] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: fair allocation of multiple resource types," in *Proc. USENIX NSDI*, 2011, p. 14.

[8] H. Kasahara and S. Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE TOC*, vol. 33, no. 11, pp. 1023–1029, 1984.

[9] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from Google compute clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[10] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. New York: Freeman, 1979.

[11] D. G. Feitelson and L. Rudolph, "Toward convergence in job schedulers for parallel supercomputers," *Springer IPPS*, vol. 1162, pp. 1–26, 1996.

[12] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *ACM TON*, vol. 1, no. 3, pp. 344–357, 1993.

[13] (2011) Google Cloud Platform. gs://clusterdata-2011-2. [Online]. Available: https://cloud.google.com/storage/

[14] P. S. Pillai, L. B. Mummert, S. W. Schlosser, R. Sukthankar, and C. J. Helfrich, "Slipstream: scalable low-latency interactive perception on streaming data," in *Proc. ACM NOSSDAV*, 2009, pp. 43–48.