Adaptive Scheduling of Parallel Jobs on Functionally Heterogeneous Resources

Yuxiong He Hongyang Sun Wen-Jing Hsu

Abstract

A parallel program usually incurs operations on multiple processing resources, interleaving computations, I/Os, and communications, where each task can only be executed on a processor of a matching category. Many parallel systems also embed special-purpose processors like vector units, floating-point co-processors, and various I/O processors. Presently, there is no provably good scheduling algorithm that ensures efficient use of multiple resources with functional heterogeneity.

This paper presents K-RAD, an algorithm that adaptively schedules parallel jobs on multiple processing resources without requiring prior information about the jobs, such as their release times and parallelism profiles. Let K denote the number of categories of heterogenous resources and P_{max} denote the maximum number of processors among all categories. We show that, for any set of jobs with arbitrary release times, K-RAD is $(K + 1 - 1/P_{max})$-competitive with respect to the makespan. This competitive ratio is provably the best possible for any non-clairvoyant deterministic algorithms for K-resource scheduling. We also show that K-RAD is $(4K + 1 - 4K/(|\mathcal{J}| + 1))$-competitive with respect to the mean response time for any batched job set $\mathcal{J}$. For the special case of $K = 1$, i.e., scheduling on homogeneous resources, the best existing mean response time bound for online non-clairvoyant algorithm is $2 + \sqrt{3} \approx 3.73$ proved by Edmonds et al. in STOC'97. We show that K-RAD is 3-competitive with respect to the mean response time when $K = 1$, which offers the best competitive ratio to date.

Technical Report TR-07-01, Center for Advanced Information System, School of Computer Engineering, Nanyang Technological University, Singapore. The revised version of this paper is published in the 2007 International Conference on Parallel Processing (ICPP-07). The authors' e-mail addresses are: {heyu0006, sunh0007, hsu}@ntu.edu.sg

This research was supported in part by Singapore-MIT Alliance.

1 Introduction

Scheduling parallel jobs on multiprocessors has been an important area of research in computer science. Most prior work is dedicated to the scheduling of jobs that require a single type of processing resources. However, many parallel systems embed special-purpose processors such as vector units, floating-point co-processors and various I/O processors. Application programs also generally involve interleaving of different types of tasks, where a task of a specific type can only be carried out on the matching type of resource. A scheduler that can handle heterogeneous resources is therefore needed.

Menasce and Almeida [28] defined two distinct forms of heterogeneity in high-performance computing systems, namely *performance heterogeneity* and *function heterogeneity*. Performance heterogeneity exists in systems that contain general-purpose processors of different speeds. A task can execute on any of the processors, but it will execute faster on some than others. In [26], performance heterogeneity was further classified as uniformly related machines and unrelated machines. In the case of uniformly related machines, the speed of each machine is the same for all jobs, but different among the machines. For unrelated machines, the speed of each machine is different for each job. Executing parallel programs on processors with different speed has been studied extensively [2, 9–11, 26, 32] in the scheduling theory. Function heterogeneity, on the other hand, exists in systems that contain various types of processors, such as vector units, floating-point co-processors, and I/O processors. With functional heterogeneity, not all of the tasks can be executed on all of the functional units. Hamidzadeh, Lilja and Atif [17] proposed a functionally heterogeneous model that incorporates performance heterogeneity by assigning an infinite computation time to a task on the wrong functional units. They studied a dynamic self-adjusting scheduling algorithm (SASH) for this type of heterogeneous systems empirically. Functional heterogeneity was also defined on a coarse level for machines with different types of parallel architectures such as SIMD, MIMD and vectors, etc in [21, 22] and more recently for the cell processor [1]. However, to the best of our knowledge, there is no general algorithm that offers provable efficiency for scheduling parallel applications on functionally heterogeneous resources.

In this paper, we propose a ***K-resource*** scheduling model for functionally heterogeneous resources and show a provably efficient scheduling algorithm. We suppose that the processors and the tasks are classified into K categories, and a task of a given category can only be executed on a processor of the matching category. In this model, any two tasks of a job can be executed concurrently

as long as they do not violate certain precedence constraints. Moreover, we study the ***online non-clairvoyant*** version of this problem, where the characteristics of the jobs such as the release times and the parallelism profiles are unknown to the scheduling algorithm a priori. For this model, we propose a scheduling algorithm called K-RAD, which extends the homogeneous resource scheduling algorithm RAD [18, 19] to the heterogeneous resources. RAD is inspired from the previous work on homogeneous resource scheduling, which has suggested that non-clairvoyant algorithms such as round robin (RR) and dynamic equi-partitioning (DEQ) [27,33,37] perform well in a multiuser environment [8, 12, 13, 18, 29]. Round robin ensures that each uncompleted job receives an equal amount of processing time by ***time-sharing*** processors among jobs, while DEQ gives each job an equal share of processors unless the job requests for less by ***space-sharing*** processors among jobs. However, DEQ algorithm can only be applied when the number of jobs in the system is no more than the number of processors and round robin works naturally for the other case where the number of jobs exceeds that of the available processors. RAD combines the two algorithms to work under both cases, and is proved to have excellent performance in terms of makespan and mean response time for any set of jobs. The good performance of RAD motivates us in this research to extend it to the scheduling of heterogenous resources and analyze its performance. In this work, however, we apply K-RAD with instantaneous parallelism of the jobs rather than the historical feedback parallelism as was done in [18,19]. This analysis enables us to prove a tighter bound on the mean response time of batched jobs for RAD on homogeneous resource scheduling. The same analysis can be shown in the case of feedback parallelism as well to obtain similar results.

Related Work

Parallel job scheduling on homogeneous resources has been studied both empirically [25, 27, 33, 36] and theoretically [8, 12, 14, 18, 19, 29, 32]. Many researchers have proven various competitive ratios in terms of makespan and mean response time for the problem of scheduling n jobs on P identical processors.

For makespan, Shmoys, Wein and Williamson [32] proved lower bounds of online non-clairvoyant scheduling. They showed that the competitive ratio is at least $(2-1/P)$ for any deterministic online algorithm, and at least $(2-1/\sqrt{P})$ for any randomized online algorithm with an oblivious adversary. Brecht, Deng, and Gu [8] proved that DEQ using instantaneous parallelism is $(2-1/P)$-competitive, and therefore is optimal.

For mean response time, Motwani, Phillips and Torng [29] showed that for jobs with arbitrary release times, no deterministic algorithm can achieve competitiveness better than $\Omega(n^{1/3})$, and no randomized algorithm can achieve competitiveness better than $\Omega(\log n)$. However, for batched jobs, Deng and Dymond [12] proved that DEQ with instantaneous parallelism is $(4 - 4/n)$-competitive. Edmonds, Chinn, Brecht and Deng [14] showed that equipartitioning (EQUI) that allocates an equal number of processors to all the jobs is $(2 + \sqrt{3})$-competitive. In our previous work [18, 19], we presented two adaptive schedulers using RAD with historical parallelism feedback, which offer O(1)-competitiveness.

We will now look at the related work on heterogeneous resource scheduling with respect to both performance and function heterogeneity. Only makespan is considered for both cases in most work.

Scheduling with performance heterogeneity considers processors with different speed [2, 9–11, 32]. For uniformly related machines, Shmoys, Wein, and Williamson [32] presented an $O(\log P)$-competitive algorithm, which matches the lower bound. Chudak and Shmoys [10] also proved a similar competitive algorithm using linear programming relaxation. Chekuri and Bender [9] presented a more efficient algorithm with the same competitive ratio. For unrelated machines, Davis and Jaffe [11] showed a number of $O(\sqrt{m})$-competitive algorithms for this problem.

Scheduling functionally heterogenous resources were studied empirically [17, 21] and theoretically [16, 23, 24, 32] as well. The theoretical work are mostly under the job-shop scheduling model, in which each job consists of a chain of heterogeneous tasks and there is only one machine from each type of resources. Shmoys, Stein and Wein [31] also generalized the job-shop scheduling to the DAG-shop scheduling with multiple processors from each resource. In the DAG-shop model, rather than a chain, a partial order among the tasks of a job may be specified. However, no two tasks from the same job can be executed concurrently. In addition, the job-shop model usually uses ***offline*** scheduling, which assumes that all the jobs' resource requirements and release times are known in advance.

Our Results

In this paper, we propose the K-resource scheduling model that captures the functional heterogeneity of resources on multiprocessors. We also extend the RAD algorithm in homogeneous resource system to K-RAD in K-resource scheduling model. We formally prove the efficiency of K-RAD with respect to both makespan and mean response time. Our main analytical results and contributions are:

- We show that any deterministic online non-clairvoyant algorithm is at best $(K+1-1/P_{max})$-competitive for makespan in K-resource system, where P_{max} denotes the maximum number of processors among the K categories of resources.
- We show that K-RAD is $(K+1-1/P_{max})$-competitive with respect to the makespan for any job set with arbitrary release times. Since it matches the lower bound, K-RAD is indeed an optimal deterministic algorithm for non-clairvoyant K-resource scheduling in terms of the makespan.
- We show that K-RAD is $(4K+1-4K/(|\mathcal{J}|+1))$-competitive with respect to the mean response time for any batched job set $\mathcal{J}$. For the special case of $K=1$, i.e., scheduling of homogeneous resources, we show that K-RAD is 3-competitive with respect to the mean response time for batched jobs. To the best of our knowledge, this offers the smallest known competitive ratio with respect to mean response time for the scheduling of parallel jobs on homogeneous multiprocessors.

The remainder of the paper is organized as follows. Section 2 describes the job model, K-resource scheduling model, and the objective functions. Section 3 presents the K-RAD algorithm. Section 4 shows the lower bound of K-RAD with respect to the makespan, followed by the competitive analysis of K-RAD on makespan in Section 5. Section 6 and Section 7 present the analysis of K-RAD for mean response time. Section 8 briefly concludes the paper.

2 Scheduling Model and Objective Functions

Our scheduling problem consists of a collection of independent jobs $\mathcal{J} = \left\{J_1, J_2, \ldots, J_{|\mathcal{J}|}\right\}$ to be scheduled on a collection of processors in K-resource systems. The processors and the tasks are categorized into K types, and a task can only be executed on a processor with the matching type. We refer to the processors belonging to a category α as ***α-processors***, and the tasks running on α-processors as ***α-tasks***. For each category $\alpha \in \{1, \ldots, K\}$, there are P_α number of α-processors in the system. In this section, we formalize the job model, the scheduling model, and present the optimization criteria of makespan and mean response time.

Job Model

We model the execution of a multi-threaded job J_i as a dynamically unfolding directed acyclic graph (**dag**) such that $J_i = (V(J_i), E(J_i))$, where $V(J_i)$ and $E(J_i)$ represent the sets of J_i's vertices and

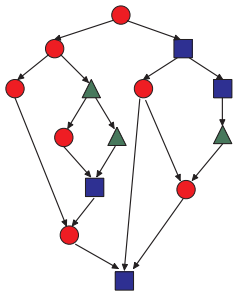

Figure 1: A job represented by a 3-DAG with 3 different types of tasks given by circles, squares, and triangles respectively.

edges respectively. As a natural extension to the conventional dag [3–7, 15, 20, 30], a parallel job with heterogenous tasks is represented as a K-color dag, or **K-DAG** in short. A K-DAG has up to K types of vertices, and each ***α-vertex*** represents a unit-time α-task where $\alpha \in \{1, \ldots, K\}$. For a job $J_i \in \mathcal{J}$, $V(J_i, \alpha)$ represents the set of α-vertices of the job. Define $V(J_i) = \cup_{\alpha=1,\ldots,K} V(J_i, \alpha)$. The ***α-work*** $T_1(J_i, \alpha)$ corresponds to the total number of α-vertices in the K-DAG, i.e., $T_1(J_i, \alpha) = |V(J_i, \alpha)|$. Each edge $e \in E(J_i)$ from vertex u to vertex v represents a dependency between the two corresponding tasks, regardless of their types. The precedence relationship $u \prec v$ holds if and only if there exists a path from vertex u to v in $E(J_i)$. The ***span*** or the ***critical path length*** $T_\infty(J_i)$ corresponds to the number of nodes on the longest chain of the precedence dependencies. The ***release time*** $r(J_i)$ is the time at which job J_i becomes first available for processing.

Figure 1 shows an example of a K-DAG job J_i with three different types of tasks represented by three different types of vertices. In this example, the circular vertices, the square vertices and the triangular vertices denote α_1-tasks, α_2-tasks and α_3-tasks of the job respectively. The work of each task is $T_1(J_i, \alpha_1) = 7$, $T_1(J_i, \alpha_2) = 4$ and $T_1(J_i, \alpha_3) = 3$. The span of the job is $T_\infty(J_i) = 7$.

K-Resource Scheduling Model

A schedule $\chi = (\tau, \pi_1, \pi_2, \ldots, \pi_K)$ of a job set $\mathcal{J}$ is defined by $K + 1$ mappings. The mapping $\tau : \cup_{J_i \in \mathcal{J}} V(J_i) \to \{1, 2, \ldots, \infty\}$ maps the vertices of the jobs in the job set $\mathcal{J}$ to the set of time steps. For each resource category $\alpha \in \{1, \ldots, K\}$, there is a mapping $\pi_\alpha : \cup_{J_i \in \mathcal{J}} V(J_i, \alpha) \to \{1, 2, \ldots, P_\alpha\}$ that maps the set of α-vertices of the jobs in the job set $\mathcal{J}$ to the set of α-processors on the machine. A valid schedule must preserve the precedence relationship of each job. For any two vertices $u, v \in V(J_i)$ of the job J_i, if $u \prec v$, then $\tau(u) < \tau(v)$, i.e., the task denoted by vertex u must be executed before the task denoted by vertex v. A valid schedule must also ensure that one processor can only be

assigned to one job at any given time. For any two vertices $u, v \in \cup_{J_i \in \mathcal{J}} V(J_i, \alpha)$, both $\tau(u) = \tau(v)$ and $\pi_\alpha(u) = \pi_\alpha(v)$ are true if and only if $u = v$.

Objective Functions

Our scheduler uses makespan and mean response time as the performance measures, which are defined as follows.

Definition 1 The ***completion time*** $T_\chi(J_i)$ of a job J_i under a schedule χ is the time at which the schedule completes the execution of the job, i.e., $T_\chi(J_i) = \max_{v \in V(J_i)} \tau(v)$. The ***makespan*** of a given job set $\mathcal{J}$ under the schedule χ is the time taken to complete all jobs in the job set $\mathcal{J}$, i.e., $\mathrm{T}_\chi(\mathcal{J}) = \max_{J_i \in \mathcal{J}} T_\chi(J_i)$.

Definition 2 The ***response time*** $\mathrm{R}_\chi(J_i)$ of a job J_i under a schedule χ is the duration between its release time $r(J_i)$ and the completion time $T_\chi(J_i)$, i.e., $\mathrm{R}_\chi(J_i) = T_\chi(J_i) - r(J_i)$. The ***total response time*** of a job set $\mathcal{J}$ under a schedule χ is given by $\mathrm{R}_\chi(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} \mathrm{R}_\chi(J_i)$ and the ***mean response time*** is $\overline{\mathrm{R}}_\chi(\mathcal{J}) = \mathrm{R}_\chi(\mathcal{J}) / |\mathcal{J}|$.

We will make use of ***competitive analysis*** to show the performance of our scheduler in terms of the makespan and the mean response time, that is, we will compare an online non-clairvoyant scheduling algorithm against an optimal offline algorithm. Let $\mathrm{T}^*(\mathcal{J})$ denote the makespan produced by the optimal algorithm on a job set $\mathcal{J}$, and $\mathrm{T}_{\chi(A)}(\mathcal{J})$ denote the makespan produced by a deterministic algorithm A for the same job set. Algorithm A is said to be ***c-competitive*** in terms of the makespan if there exists a constant b such that $\mathrm{T}_{\chi(A)}(\mathcal{J}) \leq c \cdot \mathrm{T}^*(\mathcal{J}) + b$ holds for any job set.

3 K-RAD Algorithm

In this section, we present K-RAD algorithm, which is an extension of RAD algorithm that schedules jobs on homogeneous resources. RAD unifies the space-sharing dynamic equi-partitioning (DEQ) algorithm and the time-sharing round robin (RR) algorithm to handle systems with different workload. Specifically, when the number of jobs in the system is greater than the number of processors, RAD schedules the jobs in a batched, round-robin fashion, which allocates one processor to each job with an equal share of time. When the number of jobs in the system is at most the number of processors, RAD utilizes the DEQ job scheduler, which gives each job an equal share of spatial

allotments unless a job requests for less. To schedule jobs with heterogeneous tasks on heterogeneous processors, K-RAD assigns one RAD scheduler to each category α of processors to schedule the α-tasks of all jobs in the job set, where $\alpha = 1, \ldots, K$. In the remaining part of this section, we will discuss the RAD algorithm in detail.

For each job $J_i \in \mathcal{J}$, define the ***α-desire*** $d(J_i, \alpha, t)$ to be the total number of ready α-tasks or the *instantaneous α-parallelism* of J_i at time step t and the ***α-allotment*** $a(J_i, \alpha, t)$ to be the total number of α-processors allotted to J_i at time step t. Note that an uncompleted job J_i at any time t has total desire $\sum_{\alpha=1\ldots K} d(J_i, \alpha, t) \geq 1$, although for some $\alpha \in \{1, \ldots, K\}$, its α-desire might be zero. If a job J_i has non-zero α-desire at time step t, we say that J_i is ***α-active***; otherwise, it is ***α-inactive***. For each category α of processors, let $\mathcal{J}(\alpha, t)$ denote the set of α-active jobs at time step t, i.e., $\mathcal{J}(\alpha, t) = \{J_i \in \mathcal{J} : d(J_i, \alpha, t) > 0\}$.

Whenever $|\mathcal{J}(\alpha, t)| \leq P_\alpha$, RAD uses DEQ to partition processors among the active jobs. DEQ gives each job a fair share of allotment unless a job requests for less. As a result, all jobs requesting less processors tend to get what they request, while the other jobs requesting a larger number of processors will get an equal number of processors, which we call ***mean deprived allotment***.

Once $|\mathcal{J}(\alpha, t)| > P_\alpha$, RAD will start a round-robin (RR) cycle to allot processors among the α-active jobs. In order to ensure fairness, all α-active jobs that have been scheduled once in the RR cycle will be marked and RAD maintains the marked and unmarked α-active jobs in two separate queues. One queue denoted as Q contains the α-active jobs that have not been marked since the beginning of the cycle and the other queue denoted as Q' contains the α-active jobs that have been marked. At any time step in the cycle, if there are more than P_α jobs in Q, RAD always schedules P_α jobs at the beginning of Q. When there are less than P_α jobs in Q, in order not to waste processors, RAD will move $\min(|Q'|, P_\alpha - |Q|)$ jobs from queue Q' to Q, and partition the processors to the jobs in Q by DEQ algorithm. Such a time step also indicates the completion of the RR cycle, and all jobs will be unmarked. Figure 2 shows the pseudo-code of the RAD algorithm. The main procedure RAD will be called before every step t, and it in turn calls sub-procedures DEQ or Round-Robin depending on the relationship between the number of processors and the number of unmarked α-active jobs at t.

Under the schedule of RAD, at any time step t for a category α of processors, an active job J_i can be either ***α-satisfied*** if its α-allotment is equal to its α-desire, i.e., $a(J_i, \alpha, t) = d(J_i, \alpha, t)$, or

```
DEQ (α, t, Q, P)
 1  if Q = ∅    return
 2  S ← {J_i ∈ Q : d(J_i, α, t) ≤ P/|Q|}
 3  if S = ∅    ▷ Jobs requesting a large number of processors get an equal share of processors
 4      for each J_i ∈ Q
 5          a(J_i, α, t) ← P/|Q|
 6      return
 7  else    ▷ Jobs requesting less processors get what they request
 8      for each J_i ∈ S
 9          a(J_i, α, t) ← d(J_i, α, t)
10      DEQ(α, t, Q − S, P − Σ_{J_i∈S} d(J_i, α, t))

ROUND-ROBIN (α, t, Q, P)
1  S ← the first P jobs from Q
2  for each J_i ∈ S
3      a(J_i, α, t) ← 1
4      mark J_i to indicate that J_i has been scheduled once in the RR cycle

RAD (α, t, J, P)
1  Q ← {J_i ∈ J : J_i is unmarked and α-active}
2  Q' ← {J_i ∈ J : J_i is marked and α-active}
3  if |Q| > P
4      ROUND-ROBIN(α, t, Q, P)
5  else
6      Move min(|Q'|, P − |Q|) jobs from Q' to Q
7      DEQ(α, t, Q, P)
8      unmark all jobs to indicate the completion of the RR cycle
```

Figure 2: Pseudo-code for the RAD scheduler. It includes the main procedure RAD, and two sub-procedures DEQ that implements the DEQ algorithm and ROUND-ROBIN that implements the round-robin algorithm. Before each time step t, K-RAD calls RAD to partition α-processors among jobs.

α-deprived if its α-allotment is less than its α-desire, i.e., $a(J_i, \alpha, t) < d(J_i, \alpha, t)$. Moreover, we define a job J_i to be ***∀-satisfied*** if it is α-satisfied for all $\alpha = 1, \ldots, K$, and the job is ***∃-deprived*** otherwise.

4 Makespan Lower Bounds

In this section, we will present two lower bounds on the makespan for scheduling any job set with arbitrary release time. We will also show a lower bound on the competitive ratio for any deterministic online non-clairvoyant algorithm.

We first introduce the following term for convenience.

Definition 3 The ***total α-work*** of a job set $\mathcal{J}$ is

$$T_1(\mathcal{J}, \alpha) = \sum_{J_i \in \mathcal{J}} T_1(J_i, \alpha) \ , \tag{1}$$

where $T_1(J_i, \alpha)$ is the α-work of job $J_i \in \mathcal{J}$.

Let $\mathrm{T}^*(\mathcal{J})$ denote the makespan of the job set $\mathcal{J}$ scheduled by an optimal clairvoyant scheduler. Since any scheduler on K-DAG can do no better than the optimal scheduler on any single type of task in the K-DAG, so based on the lower bounds [8] for jobs with a single type of tasks, we obtain the following two lower bounds on the makespan for any set of jobs represented by K-DAGs with arbitrary release time:

$$\mathrm{T}^*(\mathcal{J}) \geq \max_{J_i \in \mathcal{J}} (r(J_i) + T_\infty(J_i)) \ , \tag{2}$$

$$\mathrm{T}^*(\mathcal{J}) \geq \max_{\alpha=1,\ldots,K} (T_1(\mathcal{J}, \alpha)/P_\alpha) \ . \tag{3}$$

We will now show a lower bound on the competitive ratio for any deterministic online algorithm. Here, we consider an adversarial setting as with many other online algorithms. Upon knowing the strategy of any deterministic algorithm, the adversary can always make that algorithm perform badly at each step so that the competitive ratio is maximized. The following theorem gives a lower bound on the competitive ratio for any deterministic online algorithm in terms of makespan.

Theorem 1 *Any deterministic online non-clairvoyant algorithm can be no better than* $(K + 1 - 1/P_{max})$*-competitive with respect to makespan for* K*-resource scheduling, where* $P_{max} = \max_{\alpha=1,\ldots,K} P_\alpha$.

Proof. Without loss of generality, let us assume $P_K = P_{max}$. Consider a job set $\mathcal{J}$ with $n = mP_1P_K$ jobs as shown in Figure 3, where m is a large integer. All except one of the n jobs have only one unit of 1-task. The other job J_i's critical path length is $T_\infty(J_j) = K + mP_K - 1$ and it is highlighted by the dark nodes in the figure. Level 1 of J_i has one unit of 1-task, and each of the subsequent level $\alpha \in \{2, \ldots, K-1\}$ has exactly $mP_\alpha P_K$ units of α-task, all of which depend on one single task from the previous level. Level K of J_i has $mP_K(P_K - 1) + 1$ units of K-task, one of which is followed by a chain of K-task of length $mP_K - 1$.

To schedule this set of jobs, an optimal offline scheduler $\mathcal{S}$ will always execute the ready tasks of the job J_i on the critical path first so that the tasks at the subsequent level can be executed as early

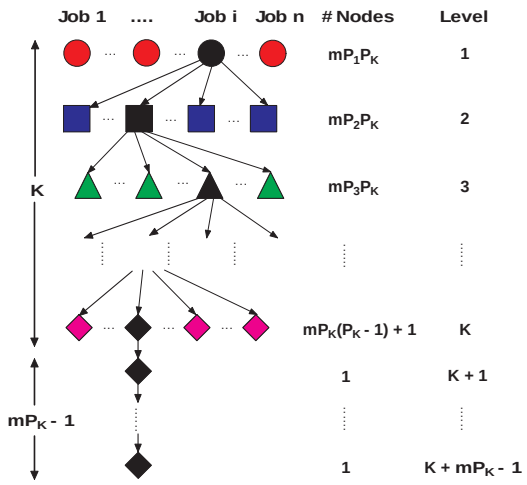

Figure 3: This example shows a set of n jobs that can force any deterministic online scheduler to have a competitive ratio of $K+1-K/P_{max}$ in terms of makespan in an adversary setting. The total number of tasks on a level is indicated on the right of that level. While an optimal clairvoyant scheduler will choose to execute the ready tasks on the critical path (denoted by the dark nodes of job J_i in the figure) first, an adversary can force a deterministic online scheduler to choose the "wrong" tasks for execution; The delayed execution of the tasks on the critical path thus prevents the efficient use of the resources.

as possible by other types of processors. For $\alpha = 1, \ldots, K$, all α-tasks of the job J_i will be ready at time $\alpha - 1$. The K-tasks, as the last type of tasks, can be completed in mP_K time steps by executing one unit of K-task on the critical path at each step. Therefore, the optimal scheduler $\mathcal{S}$ produces a makespan of $\mathrm{T}^*(\mathcal{J}) = K + mP_K - 1$.

To schedule the same set of jobs, any deterministic nonclairvoyant algorithm $\mathcal{A}$ can be prevented from performing well by the adversary in such a way that the tasks of the job J_i on the critical path are always executed last among the ready tasks. Such an adversarial strategy forces algorithm $\mathcal{A}$ to execute different types of tasks in a sequential manner. Therefore, in the worst case, $\mathcal{A}$ will take $\mathrm{T}(\mathcal{J}) \geq mKP_K + mP_K - m$ time steps.

Thus, the competitive ratio is given by

$$\begin{aligned} \frac{\mathrm{T}(\mathcal{J})}{\mathrm{T}^*(\mathcal{J})} &\geq \frac{mKP_K + mP_K - m}{K + mP_K - 1} \\ &= \frac{KP_K + P_K - 1}{\frac{K-1}{m} + P_K}. \end{aligned}$$

Let $m \gg K$, then $\frac{K-1}{m}$ approaches 0. Therefore, we have $\frac{\mathrm{T}(\mathcal{J})}{\mathrm{T}^*(\mathcal{J})} \geq \frac{KP_K+P_K-1}{P_K} = K + 1 - \frac{1}{P_{max}}$. □

5 K-RAD Makespan Analysis

This section shows that K-RAD is $(K + 1 - 1/P_{max})$-competitive with respect to the makespan. Since this competitive ratio matches the lower bound given in Section 4, it shows that K-RAD is optimal for a deterministic online algorithm in terms of makespan.

Suppose that $\mathrm{T}(\mathcal{J})$ is the makespan of job set $\mathcal{J}$ scheduled by K-RAD. Thus $I = [1, 2, \ldots, \mathrm{T}(\mathcal{J})]$ is the time interval in which K-RAD schedules the job set. Since $\mathcal{J}$ denotes any job set with arbitrary job release times, we may have subintervals of I, in which no job is active. We will refer to any subinterval $l = [t_1, \ldots, t_2] \in I$ as an ***idle interval*** if no job is active in l, i.e., all the jobs released before l are completed by $t_1 - 1$, and no new jobs are released until $t_2 + 1$. Thus no work is done during interval l. In order to analyze the makespan of K-RAD algorithm, we will first prove a lemma that bounds the makespan of any job set scheduled by K-RAD without any idle interval. Then we will relax this constraint and give the main theorem.

The following lemma bounds the makespan of any job set scheduled by K-RAD without any idle interval.

Lemma 2 *Suppose that* K-RAD *schedules a job set $\mathcal{J}$ on P_α number of α-processors for each $\alpha = 1, \ldots, K$. If there are no idle intervals during the schedule, then job set $\mathcal{J}$ completes in*

$$\mathrm{T}(\mathcal{J}) \leq \sum_{\alpha=1,\ldots,K} \frac{T_1(\mathcal{J}, \alpha)}{P_\alpha} + \left(1 - \frac{1}{P_{max}}\right) \max_{J_i \in \mathcal{J}} \left(T_\infty(J_i) + r(J_i)\right) \tag{4}$$

time steps.

Proof. Suppose that job J_k is the last job completed among the jobs in $\mathcal{J}$. Let $R(J_k)$ denote the set of time steps before J_k is released and let $S(J_k)$ and $D(J_k)$ denote the sets of $\forall$-satisfied and $\exists$-deprived time steps for J_k respectively. Since J_k is the last job completed, and $R(J_k)$, $S(J_k)$ and $D(J_k)$ are clearly disjoint sets, we have $\mathrm{T}(\mathcal{J}) = |R(J_k)| + |S(J_k)| + |D(J_k)|$. Now, we will bound these three sets separately.

(1) To bound $|R(J_k)|$: Clearly, there are $r(J_k)$ time steps before the release of job J_k, i.e.,

$$|R(J_k)| = r(J_k) \ . \tag{5}$$

Now we calculate the total amount of work done on these steps. Let $T_1'(\mathcal{J}, \alpha)$ denote the total α-work

done before the release of J_k. Since there are no idle intervals, each step in $R(J_k)$ completes at least one unit of work. Thus, we have $\sum_{\alpha=1,\ldots,K} T_1'(\mathcal{J},\alpha) \geq |R(J_k)| = r(J_k)$.

(2) To bound $|S(J_k)|$: On each $\forall$-satisfied step t for job J_k, all of the ready tasks of J_k are executed, so the span of J_k is reduced by 1. Therefore, the total number of $\forall$-satisfied steps for job J_k is at most the span $T_\infty(J_k)$ of J_k, i.e.,

$$|S(J_k)| \leq T_\infty(J_k) \ . \tag{6}$$

Let $T_1''(\mathcal{J},\alpha)$ denote the total α-work done on $\forall$-satisfied steps of J_k. Since each such step completes at least one unit of work, we have $\sum_{\alpha=1,\ldots,K} T_1''(\mathcal{J},\alpha) \geq |S(J_k)|$.

(3) To bound $|D(J_k)|$: We will first calculate the number of α-deprived steps of job J_k for each resource category α. Let $D(J_k,\alpha)$ denote the set of α-deprived steps for job J_k. According to K-RAD, on each deprived step $t \in D(J_k,\alpha)$, K-RAD must have allotted all α-processors to jobs. Thus, the total α-allotment on such a step is always equal to the total number of α-processors P_α. Since jobs always use allotted processors productively, there are P_α units of α-work done on such a time step. The total amount of α-work done on $D(J_k,\alpha)$ steps is $T_1(\mathcal{J},\alpha) - T_1'(\mathcal{J},\alpha) - T_1''(\mathcal{J},\alpha)$. We have $|D(J_k,\alpha)| \leq \left(T_1(\mathcal{J},\alpha) - T_1'(\mathcal{J},\alpha) - T_1''(\mathcal{J},\alpha)\right)/P_\alpha$.

We now bound the total number of $\exists$-deprived steps $|D(J_k)|$ for job J_k. Since $D(J_k) = \cup_{\alpha=1,\ldots,K} D(J_k,\alpha)$, we have $|D(J_k)| \leq \sum_{\alpha=1,\ldots,K} |D(J_k,\alpha)|$. Thus, we obtain

$$\begin{aligned} |D(J_k)| &\leq \sum_{\alpha=1,\ldots,K} \frac{T_1(\mathcal{J},\alpha) - T_1'(\mathcal{J},\alpha) - T_1''(\mathcal{J},\alpha)}{P_\alpha} \\ &\leq \sum_{\alpha=1,\ldots,K} \frac{T_1(\mathcal{J},\alpha)}{P_\alpha} - \frac{r(J_k) + |S(J_k)|}{P_{max}}. \end{aligned} \tag{7}$$

To bound $T(\mathcal{J})$: from Equation (5) and Inequalities (6) and (7), we can conclude that the makespan of the job set $\mathcal{J}$ is

$$\begin{aligned} \mathrm{T}(\mathcal{J}) &= |R(J_k)| + |D(J_k)| + |S(J_k)| \\ &\leq r(J_k) + \sum_{\alpha=1,\ldots,K} \frac{T_1(\mathcal{J},\alpha)}{P_\alpha} - \frac{r(J_k) + |S(J_k)|}{P_{max}} + |S(J_k)| \end{aligned}$$

$$
\begin{aligned}
&\leq \sum_{\alpha=1,\ldots,K} \frac{T_1(\mathcal{J},\alpha)}{P_\alpha} + \left(1 - \frac{1}{P_{max}}\right)\left(T_\infty(J_k) + r(J_k)\right) \\
&\leq \sum_{\alpha=1,\ldots,K} \frac{T_1(\mathcal{J},\alpha)}{P_\alpha} + \left(1 - \frac{1}{P_{max}}\right)\max_{J_i\in\mathcal{J}}\left(T_\infty(J_i) + r(J_i)\right).
\end{aligned}
$$

□

The following theorem gives the competitive ratio of K-RAD with respect to the makespan.

Theorem 3 K-RAD *is* $(K + 1 - 1/P_{max})$*-competitive with respect to the makespan for any set of jobs with arbitrary release time, where* $P_{max} = \max_{\alpha=1,\ldots,K} P_\alpha$.

Proof. Recall that $I = [1, 2, \ldots, \mathrm{T}(\mathcal{J})]$ is the time interval in which K-RAD schedules the job set $\mathcal{J}$. We will consider two cases depending on whether I contains any idle intervals or not.

Case 1 : I does not contain idle intervals.

If I does not contain idle intervals, the makespan can be bounded by Inequality (4) from Lemma 2. Since $\sum_{\alpha=1,\ldots,K} T_1(\mathcal{J},\alpha)/P_\alpha \leq K \max_{\alpha=1,\ldots,K} T_1(\mathcal{J},\alpha)/P_\alpha$ and both $\max_{\alpha=1,\ldots,K} T_1(\mathcal{J},\alpha)/P_\alpha$ and $\max_{J_i\in\mathcal{J}}\left(T_\infty(J_i) + r(J_i)\right)$ are the lower bounds for makespan. Therefore, we obtain

$$
\begin{aligned}
\mathrm{T}(\mathcal{J}) &\leq K \max_{\alpha=1,\ldots,K} \frac{T_1(\mathcal{J},\alpha)}{P_\alpha} + \left(1 - \frac{1}{P_{max}}\right)\max_{J_i\in\mathcal{J}}\left(T_\infty(J_i) + r(J_i)\right) \\
&\leq \left(K + 1 - \frac{1}{P_{max}}\right)\mathrm{T}^*(\mathcal{J}). \qquad (8)
\end{aligned}
$$

Case 2 : I contains idle intervals.

If I contains idle intervals, let $l = [t_1, \ldots, t_2]$ be the last such interval in I. Clearly, job set $\mathcal{J}$ can be divided into two disjoint subsets $\mathcal{J}_1$ and $\mathcal{J}_2$, such that $\mathcal{J}_1$ contains all the jobs completed before t_1 and $\mathcal{J}_2$ contains all the jobs released after t_2. Since K-RAD completes $\mathcal{J}_1$ in $t_1 - 1$ time steps, the optimal scheduler $\mathcal{S}$ should complete $\mathcal{J}_1$ in no more than that amount of time. Therefore both K-RAD and $\mathcal{S}$ will wait till time $t_2 + 1$ to start scheduling $\mathcal{J}_2$. Let $\mathrm{T}^*(\mathcal{J}_2)$ denote the makespan of $\mathcal{J}_2$ scheduled by the optimal scheduler $\mathcal{S}$, then we have $\mathrm{T}^*(\mathcal{J}) = t_2 + \mathrm{T}^*(\mathcal{J}_2)$. From case 1, we know that K-RAD completes $\mathcal{J}_2$ in $\mathrm{T}(\mathcal{J}_2) \leq (K + 1 - 1/P_{max})\mathrm{T}^*(\mathcal{J}_2)$ time steps. Therefore, the makespan of the job set $\mathcal{J}$ scheduled by K-RAD is

$$
\begin{aligned}
\mathrm{T}(\mathcal{J}) &= t_2 + \mathrm{T}(\mathcal{J}_2) \\
&\leq t_2 + \left(K + 1 - \frac{1}{P_{max}}\right)\mathrm{T}^*(\mathcal{J}_2)
\end{aligned}
$$

$$\leq \left(K+1-\frac{1}{P_{max}}\right)(t_2+\mathrm{T}^*(\mathcal{J}_2))$$
$$= \left(K+1-\frac{1}{P_{max}}\right)\mathrm{T}^*(\mathcal{J}). \quad (9)$$

Inequalities (8) and (9) indicate that under both cases, K-RAD achieves $(K+1-1/P_{max})$-competitiveness with respect to the makespan. □

6 Mean Response Time Lower Bounds

In this section, we present two lower bounds on the mean response time for scheduling any batched job set. We first introduce two equivalent definitions for a useful notion called *squashed sum*, which helps to establish the lower bound.

Definition 4 Given a list $\langle a_i \rangle$ of m nonnegative numbers, let $f : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}$ be a permutation satisfying $a_{f(1)} \leq a_{f(2)} \leq \cdots \leq a_{f(m)}$. The ***squashed sum*** of list $\langle a_i \rangle$ is defined as

$$\text{sq-sum}(\langle a_i \rangle) = \sum_{i=1}^{m}(m-i+1)a_{f(i)} \ . \quad (10)$$

By observing that the above permutation f on the list $\langle a_i \rangle$ gives the minimum value for the squashed sum formulation described by Equation (10), we obtain the following equivalent definition for the squashed sum of list $\langle a_i \rangle$

$$\text{sq-sum}(\langle a_i \rangle) = \min_{g \in \Upsilon} \sum_{i=1}^{m}(m-i+1)a_{g(i)} \ , \quad (11)$$

where $\Upsilon = \{g | g : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}\}$ denotes the set of all permutations on $\{1, 2, \ldots, m\}$.

We now define two terms for the lower bounds of the mean response time.

Definition 5 The ***squashed α-work area*** of a job set $\mathcal{J}$ on P_α number of α-processors is

$$\text{swa}(\mathcal{J}, \alpha) = \frac{1}{P_\alpha}\text{sq-sum}(\langle T_1(J_i, \alpha) \rangle) \ , \quad (12)$$

where $T_1(J_i, \alpha)$ is the α-work of job $J_i \in \mathcal{J}$.

Definition 6 The ***aggregate span*** of a job set $\mathcal{J}$ is

$$T_\infty(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} T_\infty(J_i) \ , \quad (13)$$

where $T_\infty(J_i)$ is the span of job $J_i \in \mathcal{J}$.

Let $\overline{\mathrm{R}^*}(\mathcal{J})$ denote the mean response time of the job set $\mathcal{J}$ scheduled by an optimal offline scheduler. Since any scheduler on K-DAG can do no better than the optimal scheduler on any single type of tasks in the K-DAG, from the mean response time lower bounds [13, 34, 35] for jobs having single type of tasks, we obtain the following two lower bounds for the mean response time on any set of batched jobs represented by K-DAGs:

$$\overline{\mathrm{R}^*}(\mathcal{J}) \geq T_\infty(\mathcal{J}) / |\mathcal{J}| \ , \tag{14}$$

$$\overline{\mathrm{R}^*}(\mathcal{J}) \geq \max_{\alpha=1,\ldots,K} \mathrm{swa}(\mathcal{J}, \alpha) / |\mathcal{J}| \ . \tag{15}$$

Thus the optimal total response time $\mathrm{R}^*(\mathcal{J})$ has lower bounds of $T_\infty(\mathcal{J})$ and $\max_{\alpha=1,\ldots,K} \mathrm{swa}(\mathcal{J}, \alpha)$.

7 K-RAD Mean Response Time Analysis

In this section, we will analyze the mean response time of the K-RAD algorithm. To begin with, we first present a supporting lemma and a notation that are useful to the mean response time analysis. Then we will define two cases for the system workload and we will show that under light workload, K-RAD has a competitive ratio of $2K + 1 - 2K/(|\mathcal{J}| + 1)$ while under heavy workload, K-RAD performs worse off by at most a factor of two in the worst case.

Lemma 4 *Let $\langle a_i \rangle$ and $\langle b_i \rangle$ be two lists of m nonnegative numbers that satisfy $b_i = a_i + s_i$, where $0 \leq s_i \leq h$ for $i = 1, 2, \ldots, m$, and h is a positive number. Let l denote the number of instances of s_i that have value h, i.e., $l = |\{s_i | s_i = h\}|$ and $P = \sum_{i=1}^{m} s_i$. If $l > 0$, then we have*

$$\text{sq-sum}(\langle b_i \rangle) \geq \text{sq-sum}(\langle a_i \rangle) + \frac{P(l+1)}{2} \ .$$

Proof. Let $\Upsilon = \{g | g : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}\}$ denote the set of all permutations from $\{1, 2, \ldots, m\}$ to $\{1, 2, \ldots, m\}$. Let $f : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}$ denote a permutation that satisfies $b_{f(1)} \leq b_{f(2)} \leq \cdots \leq b_{f(m)}$. Then, according to Equation (10) in Definition 4, we have

$$\text{sq-sum}(\langle b_i \rangle) = \sum_{i=1}^{m} (m - i + 1) b_{f(i)}$$

$$
\begin{aligned}
&= \sum_{i=1}^{m}(m-i+1)(a_{f(i)}+s_{f(i)}) \\
&= \sum_{i=1}^{m}(m-i+1)a_{f(i)} + \sum_{i=1}^{m}(m-i+1)s_{f(i)} \\
&\geq \min_{g\in\Upsilon}\sum_{i=1}^{m}(m-i+1)a_{g(i)} + \min_{k\in\Upsilon}\sum_{i=1}^{m}(m-i+1)s_{k(i)} \\
&= \text{sq-sum}\left(\langle a_i\rangle\right) + \text{sq-sum}\left(\langle s_i\rangle\right).
\end{aligned}
$$

Now we will show that sq-sum $(\langle s_i\rangle) \geq P(l+1)/2$. To simplify the notation, rename the elements of the list $\langle s_i\rangle$ such that $s_1 \leq s_2 \leq \cdots \leq s_{m-l} < s_{m-l+1} = s_{m-l+2} = \cdots = s_m$. Since $\sum_{i=1}^{m} s_i = P$, we have $s_{m-l+1} = s_{m-l+2} = \cdots = s_m = h = (P - \sum_{i=1}^{m-l} s_i)/l$. Then the squashed sum of $\langle s_i\rangle$ is

$$
\begin{aligned}
\text{sq-sum}\left(\langle s_i\rangle\right) &= \sum_{i=1}^{m}(m-i+1)s_i \\
&= \sum_{i=1}^{m-l}(m-i+1)s_i + \sum_{i=m-l+1}^{m}(m-i+1)s_i \\
&= \sum_{i=1}^{m-l}(m-i+1)s_i + \frac{(P-\sum_{i=1}^{m-l}s_i)}{l}\sum_{i=1}^{l} i \\
&= \sum_{i=1}^{m-l}(m-i+1)s_i + \frac{(P-\sum_{i=1}^{m-l}s_i)}{l}\frac{l(l+1)}{2} \\
&= \frac{\sum_{i=1}^{m-l}(2m-2i+2)s_i}{2} + \frac{(P-\sum_{i=1}^{m-l}s_i)(l+1)}{2} \\
&= \frac{P(l+1)}{2} + \frac{\sum_{i=1}^{m-l}(2m-2i-l+1)s_i}{2} \\
&\geq \frac{P(l+1)}{2}.
\end{aligned}
$$

The last inequality holds because $s_i \geq 0$ and $2m-2i-l+1 > 0$ for $i = 1, 2, \ldots, m-l$. □

We will introduce a notation — ***t-suffix***, in order to simplify the presentation of the response time analysis. For any time step t, t-suffix, denoted as $\overrightarrow{t} = \{t, t+1, \ldots, T(\mathcal{J})\}$, represents the set of time steps from t to the completion of the job set $\mathcal{J}$. We will be interested in the suffix of a job, namely, the tasks that remain to be executed after a given time step. For a job $J_i \in \mathcal{J}$, define the t-suffix of the job $J_i\left(\overrightarrow{t}\right)$ to be the portion of the job induced by those vertices in $V(J_i)$ that execute on or after time step t, i.e.,

$$
J_i\left(\overrightarrow{t}\right) = \left(V\left(J_i\left(\overrightarrow{t}\right)\right), E\left(J_i\left(\overrightarrow{t}\right)\right)\right),
$$

where $V\left(J_i\left(\overrightarrow{t}\right)\right) = \{v \in V(J_i) : \tau(v) \geq t\}$ and $E\left(J_i\left(\overrightarrow{t}\right)\right) = \left\{(u,v) \in E(J_i) : u,v \in V\left(J_i\left(\overrightarrow{t}\right)\right)\right\}$. The t-suffix of the job set $\mathcal{J}$ is

$$\mathcal{J}\left(\overrightarrow{t}\right) = \left\{J_i\left(\overrightarrow{t}\right) : J_i \in \mathcal{J} \text{ and } V\left(J_i\left(\overrightarrow{t}\right)\right) \neq \emptyset\right\} .$$

Thus, we have $\mathcal{J} = \mathcal{J}\left(\overrightarrow{1}\right)$, and the number of incomplete jobs at time step t is the number $\left|\mathcal{J}\left(\overrightarrow{t}\right)\right|$ of nonempty jobs in $\mathcal{J}\left(\overrightarrow{t}\right)$.

Now we will define two cases for the system workload and analyze the performance of RAD under each one of them separately. Recall that at any time t, $\mathcal{J}(\alpha,t)$ denotes the set of α-active jobs for $\alpha = 1,\ldots,K$. We say that the system has *light workload* when $|\mathcal{J}(\alpha,t)| \leq P_\alpha$ at *any* time t during the schedule for *all* $\alpha = 1,\ldots,K$. In this case, K-RAD utilizes only the DEQ algorithm. On the other hand, the system is considered to have *heavy workload* when $|\mathcal{J}(\alpha,t)| > P_\alpha$ for *some* $\alpha = 1,\ldots,K$ at *some* time t. In this case K-RAD utilizes both DEQ and RR algorithms during the schedule.

Analysis of K-RAD under light workload

The following theorem gives the competitive ratio for the mean response time produced by K-RAD scheduler under light system workload.

Theorem 5 K-RAD *is* $(2K+1-2K/(|\mathcal{J}|+1))$*-competitive with respect to the mean response time for any batched job set* $\mathcal{J}$*, if at any time* t*,* $|\mathcal{J}(\alpha,t)| \leq P_\alpha$ *for each* $\alpha = 1,\ldots,K$*, i.e., the number of jobs that require processors never exceeds the number of available processors.*

Proof. Suppose that K-RAD schedules a batched job set $\mathcal{J}$ on a machine with P_α number of α-processors for $\alpha = 1,\ldots,K$. We will show that the total response time of $\mathcal{J}$ can be bounded by

$$\mathrm{R}(\mathcal{J}) \leq \left(2 - \frac{2}{|\mathcal{J}|+1}\right)\left(\sum_{\alpha=1,\ldots,K} \mathrm{swa}\left(\mathcal{J},\alpha\right)\right) + T_\infty\left(\mathcal{J}\right) . \tag{16}$$

Since $\sum_{\alpha=1,\ldots,K} \mathrm{swa}\left(\mathcal{J},\alpha\right) \leq K \max_{\alpha=1,\ldots,K} \mathrm{swa}\left(\mathcal{J},\alpha\right)$ and both $\max_{\alpha=1,\ldots,K} \mathrm{swa}\left(\mathcal{J},\alpha\right)$ and $T_\infty\left(\mathcal{J}\right)$ are lower bounds for the total response time, Inequality (16) indicates that K-RAD is $(2K+1-2K/(|\mathcal{J}|+1))$-competitive with respect to the total response time, or equivalently with respect to the mean response time under light workload. Now, we will prove Inequality (16) by induction on the remaining execution time of the job set $\mathcal{J}\left(\overrightarrow{t}\right)$.

Basis: $t = T(\mathcal{J}) + 1$. When $t = T(\mathcal{J}) + 1$, we have $\mathcal{J}\left(\overrightarrow{t}\right) = \emptyset$. It follows that $\mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) = 0$, $\mathrm{swa}\left(\mathcal{J}\left(\overrightarrow{t}\right), \alpha\right) = 0$ for $\alpha = 1, \ldots, K$, and $T_\infty\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) = 0$. Thus, the claim holds trivially.

Induction: $1 \le t \le T(\mathcal{J})$. Let $n = \left|\mathcal{J}\left(\overrightarrow{t}\right)\right|$ denote the number of incomplete jobs at time t. Since $n \ge \left|\mathcal{J}\left(\overrightarrow{t+1}\right)\right|$, we have $2 - 2/(n+1) \ge 2 - 2/\left(\left|\mathcal{J}\left(\overrightarrow{t+1}\right)\right| + 1\right)$. Suppose that Inequality (16) holds at time step $t+1$, i.e.,

$$\mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) \le \left(2 - \frac{2}{n+1}\right)\left(\sum_{\alpha=1,\ldots,K} \mathrm{swa}\left(\mathcal{J}\left(\overrightarrow{t+1}\right), \alpha\right)\right) + T_\infty\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) . \tag{17}$$

We will prove that it still holds at time step t, i.e.,

$$\mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) \le \left(2 - \frac{2}{n+1}\right)\left(\sum_{\alpha=1,\ldots,K} \mathrm{swa}\left(\mathcal{J}\left(\overrightarrow{t}\right), \alpha\right)\right) + T_\infty\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) . \tag{18}$$

The following notations denote the changes in, respectively, the total response time, the squashed α-work area, and the aggregate span from time step t and $t+1$:

$$\begin{aligned}
&\Delta r = \mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) - \mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) , \\
&\Delta\,\mathrm{swa}\left(\alpha\right) = \mathrm{swa}\left(\mathcal{J}\left(\overrightarrow{t}\right), \alpha\right) - \mathrm{swa}\left(\mathcal{J}\left(\overrightarrow{t+1}\right), \alpha\right) , \\
&\Delta T_\infty = T_\infty\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) - T_\infty\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) .
\end{aligned}$$

Given induction hypothesis (Inequality (17)), we need only prove that the following inequality holds

$$\Delta r \le \left(2 - \frac{2}{n+1}\right)\left(\sum_{\alpha=1,\ldots,K} \Delta\,\mathrm{swa}\left(\alpha\right)\right) + \Delta T_\infty , \tag{19}$$

in order to prove our claim (Inequality (18)). We divide the proof of Inequality (19) into four steps.

(1) To bound Δr: At any time t, the total number of incomplete jobs is $\left|\mathcal{J}\left(\overrightarrow{t}\right)\right| = n$. Since each incomplete job in $\mathcal{J}\left(\overrightarrow{t}\right)$ adds one time step to the total response time during step t, we have

$$\Delta r = n . \tag{20}$$

(2) To bound ΔT_∞: At time step t, an incomplete job J_i is either $\forall$-satisfied or $\exists$-deprived. The incomplete jobs can be partitioned as $\mathcal{J}\left(\overrightarrow{t}\right) = \mathcal{JS}(t) \cup \mathcal{JD}(t)$, representing the set of $\forall$-satisfied

and $\exists$-deprived jobs at time t, respectively. If $J_i \in \mathcal{JS}(t)$, the span of J_i must reduce by 1 at time step t, i.e. $T_\infty\left(J_i\left(\overrightarrow{t}\right)\right) = T_\infty\left(J_i\left(\overrightarrow{t+1}\right)\right) + 1$. If $J_i \in \mathcal{JD}(t)$, the span of J_i never increases at any time step t, i.e. $T_\infty\left(J_i\left(\overrightarrow{t}\right)\right) \geq T_\infty\left(J_i\left(\overrightarrow{t+1}\right)\right)$. Therefore, the aggregate span of $\mathcal{J}$ must reduce by at least $|\mathcal{JS}(t)|$ at time step t, and we have

$$\Delta T_\infty \geq |\mathcal{JS}(t)| \ . \tag{21}$$

(3) To bound $\Delta\,\mathrm{swa}\,(\alpha)$: Consider the α-work of $J_i\left(\overrightarrow{t}\right)$ and $J_i\left(\overrightarrow{t+1}\right)$. For a job J_i that is α-deprived at time step t, J_i has α-allotment $a\,(J_i, \alpha, t) = \bar{p}\,(\alpha, t)$, where $\bar{p}\,(\alpha, t)$ denotes the mean deprived allotment at time step t for α-processors. Thus, we have

$$T_1\left(J_i\left(\overrightarrow{t}\right), \alpha\right) = T_1\left(J_i\left(\overrightarrow{t+1}\right), \alpha\right) + \bar{p}\,(\alpha, t) \ . \tag{22}$$

For a job J_i that is α-satisfied at time step t, J_i's α-allotment is equal to its α-desire, i.e., $a\,(J_i, \alpha, t) = d(J_i, \alpha, t)$. We have

$$T_1\left(J_i\left(\overrightarrow{t}\right), \alpha\right) = T_1\left(J_i\left(\overrightarrow{t+1}\right), \alpha\right) + d(J_i, \alpha, t) \ . \tag{23}$$

Let $\mathcal{JS}(\alpha, t)$ and $\mathcal{JD}(\alpha, t)$ denote the set of α-satisfied and α-deprived jobs at time step t respectively.

If there exist no α-deprived jobs at time t, i.e., $|\mathcal{JD}(\alpha, t)| = 0$. In this case, it is obvious that for $\alpha = 1, \ldots, K$, we have

$$\Delta\,\mathrm{swa}\,(\alpha) \geq 0. \tag{24}$$

If there exist α-deprived jobs at time t, i.e., $|\mathcal{JD}(\alpha, t)| > 0$. In this case, *all* α-processors must have been allotted to the jobs (otherwise, there would not be any jobs deprived of the α-processors). Recall that DEQ first allots jobs that request less processors (than the fair share of equal number of processors). Consequently, the jobs that get allotments later will have no less allotment than the jobs getting allotment earlier. Therefore, the deprived jobs actually receive no less number of processors than the satisfied jobs. With this scenario in mind, Lemma 4 bounds the change in the squashed α-work.

For $J_i \in \mathcal{J}$, let $a_i = T_1\left(J_i\left(\overrightarrow{t+1}\right), \alpha\right)$ and $b_i = T_1\left(J_i\left(\overrightarrow{t}\right), \alpha\right)$. By DEQ, all the α-processors have been allotted, i.e. $\sum_{J_i \in \mathcal{J}} a\left(J_i, \alpha, t\right) = P_\alpha$. Therefore, according to Lemma 4, we have

$$\text{sq-sum}\left(\langle b_i \rangle\right) - \text{sq-sum}\left(\langle a_i \rangle\right) \geq \frac{P_\alpha\left(\left|\mathcal{JD}(\alpha, t)\right| + 1\right)}{2} \ . \tag{25}$$

From Inequality (25) and Definition 5, we get for $\alpha = 1, \ldots, K$ and $\left|\mathcal{JD}(\alpha, t)\right| > 0$

$$\begin{aligned} \Delta\,\text{swa}\left(\alpha\right) &= \frac{\text{sq-sum}\left(\langle b_i \rangle\right) - \text{sq-sum}\left(\langle a_i \rangle\right)}{P_\alpha} \\ &\geq \frac{\left|\mathcal{JD}(\alpha, t)\right| + 1}{2} \ . \end{aligned} \tag{26}$$

(4) To derive Inequality (19): Since the incomplete jobs at time t can be partitioned as $\mathcal{J}\left(\overrightarrow{t}\right) = \mathcal{JS}(t) \cup \mathcal{JD}(t)$, we have $\left|\mathcal{JS}(t)\right| + \left|\mathcal{JD}(t)\right| = n$. Let $\mathcal{K} = \{1, 2, \ldots, K\}$, and let $\mathcal{K}' = \{\alpha \in \mathcal{K} : \left|\mathcal{JD}(\alpha, t)\right| > 0\}$. We can obtain the following inequality,

$$\begin{aligned} \sum_{\alpha \in \mathcal{K}'} \left|\mathcal{JD}(\alpha, t)\right| &= \sum_{\alpha \in \mathcal{K}} \left|\mathcal{JD}(\alpha, t)\right| \\ &\geq \left|\cup_{\alpha \in \mathcal{K}}\, \mathcal{JD}(\alpha, t)\right| \\ &= n - \left|\mathcal{JS}(t)\right| \ . \end{aligned} \tag{27}$$

Hence, from Inequalities (21), (24), (26) and (27), we get

$$\begin{aligned} &\left(2 - \frac{2}{n+1}\right)\left(\sum_{\alpha \in \mathcal{K}} \Delta\,\text{swa}\left(\alpha\right)\right) + \Delta T_\infty \\ &\geq \frac{2n}{n+1}\left(\sum_{\alpha \in \mathcal{K}'} \frac{\left|\mathcal{JD}(\alpha, t)\right| + 1}{2}\right) + \left|\mathcal{JS}(t)\right| \\ &\geq \frac{n}{n+1}\left(n - \left|\mathcal{JS}(t)\right| + \left|\mathcal{K}'\right|\right) + \left|\mathcal{JS}(t)\right| \ . \end{aligned} \tag{28}$$

If $\mathcal{K}' \neq \emptyset$, which implies $\left|\mathcal{K}'\right| \geq 1$. We have

$$\begin{aligned} &\frac{n}{n+1}\left(n - \left|\mathcal{JS}(t)\right| + \left|\mathcal{K}'\right|\right) + \left|\mathcal{JS}(t)\right| \\ &\geq n - \left|\mathcal{JS}(t)\right|\frac{n}{n+1} + \left|\mathcal{JS}(t)\right| \\ &\geq n \end{aligned} \tag{29}$$

If $\mathcal{K}' = \emptyset$, then we have $|\mathcal{JS}(t)| = n$. Inequality (29) holds trivially. Now, Equation (20) and Inequalities (28) and (29) indicate that Inequality (19) holds. The proof is complete. □

In the case where $K = 1$ for homogeneous resource scheduling, Theorem 5 indicates that DEQ algorithm is $(3 - 2/(|\mathcal{J}| + 1))$-competitive with respect to the mean response time. This result tightens the performance bound for DEQ algorithm analyzed in [12, 13]. However, DEQ is only applicable to the light workload case where the number of jobs is no more than the number of processors. By combining the current mean response time analysis with our previous work in [19], it is not hard to show that RAD is also $(3 - 2/(|\mathcal{J}| + 1))$-competitive for scheduling homogeneous resources on *any* batched job set. To the best of our knowledge, RAD is the first algorithm that offers 3-competitiveness with respect to mean response time for the scheduling of parallel jobs.

Analysis of K-RAD under heavy workload

Under heavy system workload, i.e., there exists some time t and some $\alpha = 1, \ldots, K$ for which $|\mathcal{J}(\alpha, t)| > P_\alpha$ and K-RAD will utilize both DEQ and RR algorithms. The following theorem gives the competitive ratio for the mean response time produced by K-RAD scheduler under this more general case.

Theorem 6 K-RAD *is* $(4K + 1 - 4K/(|\mathcal{J}| + 1))$*-competitive with respect to the mean response time for any batched job set* $\mathcal{J}$*.*

Proof. Similar to the proof of Theorem 5, we will show that to schedule a batch of n jobs, K-RAD achieves the following total response time

$$\mathrm{R}(\mathcal{J}) \le \left(4 - \frac{4}{n+1}\right) \left(\sum_{\alpha=1,\ldots,K} \mathrm{swa}\,(\mathcal{J}, \alpha)\right) + T_\infty\,(\mathcal{J})\,. \tag{30}$$

Then the main theorem follows directly. Also to prove it by induction, we need only show the following inequality holds on each time step t,

$$\Delta\, r \le \left(4 - \frac{4}{n+1}\right) \left(\sum_{\alpha=1,\ldots,K} \Delta\,\mathrm{swa}\,(\alpha)\right) + \Delta T_\infty. \tag{31}$$

Since the change of the total response time $\Delta\, r$ and the aggregate span ΔT_∞ are still given by Equation (20) and Inequality (21), the rest of the proof will focus on calculating the change of the

squashed α-work area $\Delta\,\mathrm{swa}\,(\alpha)$. By comparing Inequalities (19), (31) and (26), it suffices to show that when being scheduled by round robin (RR),

$$\Delta\,\mathrm{swa}\,(\alpha) \geq \frac{|\mathcal{JD}(\alpha,t)|+1}{4} \tag{32}$$

for $\alpha = 1,\ldots,K$ and $|\mathcal{JD}(\alpha,t)| > 0$. Recall from Section 3 that a RR cycle consists of more than one time steps. Specifically, during a RR cycle, all the jobs that have been α-active are scheduled at least once. Let l denote the total number of jobs scheduled in the RR cycle, and let τ denote the total number of time steps in the RR cycle. We have

$$\begin{aligned} \tau &= \lceil l/P_\alpha \rceil \\ &\leq l/P_\alpha + 1\ . \end{aligned}$$

Since each scheduled job has reduced its α-work by at least 1, according to Lemma 4 and Definition 5, the total change of the squashed α-work area $\Delta\,\mathrm{RR}\,(\alpha)$ during the cycle is given by

$$\Delta\,\mathrm{RR}\,(\alpha) \geq \frac{l(l+1)}{2P_\alpha} \tag{33}$$

Thus, given $P_\alpha < l$, the average change of the squashed α-work area $\Delta\,\mathrm{swa}\,(\alpha)$ during each time step of the cycle is

$$\begin{aligned} \Delta\,\mathrm{swa}\,(\alpha) &= \frac{\Delta\,\mathrm{RR}\,(\alpha)}{\tau} \\ &\geq \frac{l(l+1)/2P_\alpha}{(l+P_\alpha)/P_\alpha} \\ &\geq \frac{l+1}{4}\ . \end{aligned} \tag{34}$$

At any time step t in the cycle, the number of α-deprived jobs $|\mathcal{JD}(\alpha,t)|$ is at most the total number l of α-active jobs in the cycle, i.e., $|\mathcal{JD}(\alpha,t)| \leq l$. Inequality (34) then directly implies Inequality (32). The proof is complete. □

The results of Theorem 5 and Theorem 6 suggest that K-RAD doubles the competitive ratio under heavy workload by using round robin. One intuitive explanation is that, when round robin is used under heavy workload, if only a small number of jobs are active with limited desires, then

almost all the processor cycles in the last time step of a RR cycle can be wasted, which causes the bound to double in the worst case.

8 Concluding Remarks

We have proposed a new scheduling model — K-resource scheduling to incorporate the functional heterogeneity. We have also presented a provably efficient algorithm — K-RAD for scheduling parallel jobs under this model. Ultimately, efficient algorithms will be needed for scheduling large parallel machines with both general-purpose processors of different speed and special-purpose processors with different functionality. Therefore, one interesting challenge now is to extend the results to such scenarios.

References

[1] http://researchweb.watson.ibm.com/cell/.

[2] M. A. Bender and M. O. Rabin. Scheduling Cilk multithreaded computations on processors of different speeds. In *SPAA*, pages 13–21, July 2000.

[3] G. Blelloch, P. Gibbons, and Y. Matias. Provably efficient scheduling for languages with fine-grained parallelism. *Journal of the ACM*, 46(2):281–321, 1999.

[4] G. E. Blelloch and J. Greiner. A provable time and space efficient implementation of NESL. In *ICFP*, pages 213–225, Philadelphia, Pennsylvania, 1996.

[5] R. D. Blumofe. *Executing Multithreaded Programs Efficiently.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

[6] R. D. Blumofe and C. E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM Journal on Computing*, 27(1):202–229, 1998.

[7] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.

[8] T. Brecht, X. Deng, and N. Gu. Competitive dynamic multiprocessor allocation for parallel applications. In *Parallel and Distributed Processing*, pages 448 – 455, San Antonio, TX, 1995.

[9] C. Chekuri and M. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41(2):212–224, 2001.

[10] F. A. Chudak and D. B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *SODA*, pages 581–590, Philadelphia, PA, USA, 1997.

[11] E. Davis and J. M. Jaffe. Algorithms for scheduling tasks on unrelated processors. *Journal of ACM*, 28(4):721–736, 1981.

[12] X. Deng and P. Dymond. On multiprocessor system scheduling. In *SPAA*, pages 82–88, Padua, Italy, 1996.

[13] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *SODA*, pages 159–167, Philadelphia, PA, USA, 1996.

[14] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics (extended abstract). In *STOC*, pages 120–129, 1997.

[15] Z. Fang, P. Tang, P.-C. Yew, and C.-Q. Zhu. Dynamic processor self-scheduling for general parallel nested loops. *IEEE Transactions on Computers*, 39(7):919–929, 1990.

[16] Goldberg, Paterson, Srinivasan, and Sweedyk. Better approximation guarantees for job-shop scheduling. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.

[17] B. Hamidzadeh, D. J. Lilja, and Y. Atif. Dynamic scheduling techniques for heterogeneous computing systems. *Concurrency: Practice and Experience*, 7(7):633–652, 1995.

[18] Y. He, W. J. Hsu, and C. E. Leiserson. Provably efficient two-level adaptive scheduling. In *JSSPP*, Saint-Malo, France, 2006.

[19] Y. He, W. J. Hsu, and C. E. Leiserson. Provably efficient adaptive scheduling through equalized allotments. In *IPDPS*, Long Beach, California, USA, 2007.

[20] S. F. Hummel and E. Schonberg. Low-overhead scheduling of nested parallelism. *IBM Journal of Research and Development*, 35(5-6):743–765, 1991.

[21] A. Khokhar, V. K. Prasanna, M. Shaaban, and C.-L. Wang. Heterogeneous supercomputing: Problems and issues. In *Workshop on Heterogeneous Processing*, 1992.

[22] T. T. Kwan, R. E. McGrath, and D. A. Reed. Em3: A taxonomy of heterogeneous computing systems. *Computer*, 28(12):68–70, 1995.

[23] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. Technical Report BS–R89xx, Centre for Mathematics and Computer Science, The Netherlands, 1989.

[24] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-shop scheduling in o (congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[25] S. T. Leutenegger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *SIGMETRICS*, pages 226–236, Boulder, Colorado, United States, 1990.

[26] R. L.Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. RinnooyKan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[27] C. McCann, R. Vaswani, and J. Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, 1993.

[28] D. Menasce and V. Almeida. Heterogeneous supercomputing: Is it cost-effective? In *Supercomputing*, pages 169–177, 1990.

[29] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. In *SODA*, pages 422–431, Austin, Texas, United States, 1993.

[30] G. J. Narlikar and G. E. Blelloch. Space-efficient scheduling of nested parallelism. *ACM Transactions on Programming Languages and Systems*, 21(1):138–173, 1999.

[31] Shmoys, Stein, and Wein. Improved approximation algorithms for shop scheduling problems. In *SODA*, 1991.

[32] D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines online. In *FOCS*, pages 131–140, San Juan, Puerto Rico, 1991.

[33] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *SOSP*, pages 159–166, New York, NY, USA, 1989.

[34] J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. S. Yu. Scheduling parallelizable tasks to minimize average response time. In *SPAA*, pages 200–209, Cape May, New Jersey, United States, 1994.

[35] J. Turek, U. Schwiegelshohn, J. L. Wolf, and P. S. Yu. Scheduling parallel tasks to minimize average response time. In *SODA*, pages 112–121, Philadelphia, PA, USA, 1994.

[36] K. K. Yue and D. J. Lilja. Implementing a dynamic processor allocation policy for multiprogrammed parallel applications in the Solaris^{TM}operating system. *Concurrency and Computation-Practice and Experience*, 13(6):449–464, 2001.

[37] J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In *SIGMETRICS*, pages 214–225, Boulder, Colorado, United States, 1990.