Adaptive Scheduling of Parallel Jobs on Functionally Heterogeneous Resources

Yuxiong He[†‡] Hongyang Sun[†] Wen-Jing Hsu[†‡]

[†]*Nanyang Technological University*
[‡]*Singapore-MIT Alliance*

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.*

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 [21] define two distinct forms of heterogeneity in high-performance computing systems. *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. *Functional 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 et al. [11] propose a generic model that incorporates performance heterogeneity by assigning an infinite computation time to a task on the unmatched functional units. They study a dynamic self-adjusting scheduling algorithm for this type of heterogeneous systems empirically. Other forms of functional heterogeneity are also defined on a coarse level for machines with different types of parallel architectures such as SIMD, MIMD and vectors, etc. in [15,16] and more recently for the cell processors [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.

We propose a ***K-resource*** scheduling model for functionally heterogeneous resources and present a provably efficient scheduling algorithm. In the K-resource model, 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. Any two tasks of a job can be executed concurrently as long as they do not violate the 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.

We present a non-clairvoyant scheduling algorithm K-RAD, which extends the homogeneous resource scheduling algorithm RAD [12, 13] to the heterogeneous resources. RAD is a scheduling algorithm that combines the dynamic equi-partitioning (DEQ) algorithm [20, 26, 30] with the round robin (RR) algorithm. In [12, 13], we show that RAD is provably efficient in terms of both makespan and mean response time for the scheduling of heterogeneous resources. The good performance of RAD motivates us to extend it to heterogenous resource scheduling.

Related Work

Parallel job scheduling on homogeneous resources has been studied both empirically [19,20,26,29] and theoretically [5, 9, 10, 12, 13, 22, 25]. 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 et al. [25] prove lower bounds of online non-clairvoyant scheduling. They show 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 et al. [5] show that DEQ using instantaneous parallelism is $(2 - 1/P)$-competitive, and therefore is optimal.

For mean response time of batched jobs, Motwani et al. [22] show that round robin is 2-competitive, which is the best online competitive ratio that matches the lower bound. Deng and Dymond [9] prove that DEQ with instantaneous parallelism is $(4 - 4/n)$-competitive. Edmonds et al. [10] show that equi-partitioning is $(2 + \sqrt{3})$-competitive. In our previous work [12, 13], we introduce RAD and show its $O(1)$ competitiveness by using history-based feedback.

Scheduling with performance heterogeneity has been studied by many researchers [6–8, 25]. Shmoys et al. [25] present an $O(\log P)$-competitive algorithm in terms of makespan for uniformly related machines, which matches the lower bound. Davis and Jaffe [8] show a number of $O(\sqrt{m})$-competitive algorithms for unrelated machines.

Scheduling functionally heterogenous resources is studied mostly under the job-shop scheduling [17, 18, 24]. In the job-shop model, each job consists of a chain of heterogeneous tasks and there is only one machine from each type of resources. Shmoys et al. [24] generalize 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 are known in advance.

Our Results

We propose a K-resource scheduling model that captures the functional heterogeneity of resources on multiprocessors. We extend the RAD algorithm for homogeneous resources to K-RAD for the K-resource scheduling and 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 optimal for deterministic 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 best competitive ratio to date.

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 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 Models and Objective Functions

Our scheduling problem consists of a collection of independent jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_{|\mathcal{J}|}\}$ 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.

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 edges respectively. As a natural extension to the conventional dag [2–4, 23], a parallel job with heterogenous tasks is represented as a K-color dag, or **K-DAG**. A K-DAG has up to K types of vertices, and each ***α-vertex*** represents a unit-time α-task where $\alpha = 1, \ldots, K$. For a job $J_i \in \mathcal{J}$, $V(J_i, \alpha)$ represents the set of α-vertices of the job. Define $V(J_i) = \cup_{\alpha=1}^{K} V(J_i, \alpha)$. The ***$\alpha$-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 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 is first available for processing. Figure 1 shows an example of a 3-DAG job with three different types of tasks.

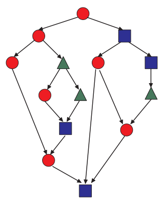

Figure 1: A 3-DAG job with 3 different types of tasks.

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 function $\tau : \cup_{J_i \in \mathcal{J}} V(J_i) \to \{1, 2, \ldots, \infty\}$ maps the vertices of the jobs in the job set $\mathcal{J}$ to the set of time steps. For each resource category $\alpha \in \{1, \ldots, K\}$, the function $\pi_\alpha : \cup_{J_i \in \mathcal{J}} V(J_i, \alpha) \to \{1, 2, \ldots, P_\alpha\}$ maps the set of α-vertices of the jobs in the job set $\mathcal{J}$ to the set of α-processors. 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 $\tau(u) < \tau(v)$. 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 metrics, which are defined as follows.

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

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

We will use ***competitive analysis***, which compares an online non-clairvoyant algorithm against an optimal offline algorithm. Let $\mathrm{T}^*(\mathcal{J})$ denote the makespan produced by the optimal algorithm $\mathcal{S}$ on a job set $\mathcal{J}$, and $\mathrm{T}(\mathcal{J})$ denote the makespan produced by a online deterministic algorithm $\mathcal{A}$ for the same job set. Algorithm $\mathcal{A}$ is said to be ***c-competitive*** in terms of the makespan if there exists a constant b such that $\mathrm{T}(\mathcal{J}) \le c \cdot \mathrm{T}^*(\mathcal{J}) + b$ holds for any job set $\mathcal{J}$.

3 K-RAD Algorithm

In this section, we present K-RAD algorithm, which is an extension of RAD algorithm [12,13] 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 different levels of workload. Specifically, when the number of jobs in the system exceeds 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 not more than the number of processors, RAD utilizes DEQ algorithm, 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 manage 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. Note that an uncompleted job J_i at any time t has total desire $\sum_{\alpha=1}^{K} d(J_i, \alpha, t) \ge 1$, although for some $\alpha \in \{1, \ldots, K\}$, its α-desire might be zero. At a given time step t, if a job J_i has non-zero α-desire, 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 $|\mathcal{J}(\alpha, t)| \le P_\alpha$, RAD uses DEQ to partition processors among the active jobs. Under DEQ, all jobs requesting less than fair share of processors tend to get what they request, while the other jobs requesting larger numbers 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 kept in a queue denoted as Q. Other α-active jobs that have not been scheduled since the beginning of the RR cycle is kept in another queue denoted as Q'. 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 $\min\left(\left|Q'\right|, P_\alpha - |Q|\right)$ jobs from queue Q' to Q, and partition the processors to the jobs in Q using DEQ. Such a time step also indicates the completion of the RR cycle, and all the jobs will be unmarked.

Figure 2 shows the pseudo-code of RAD algorithm. 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 ***$\alpha$-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 ***$\forall$-satisfied*** if it is α-satisfied for all $\alpha = 1, \ldots, K$, and the job is ***$\exists$-deprived*** otherwise.

4 Makespan Lower Bounds

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

```
DEQ $(\alpha, t, Q, P)$
1   **if** $Q = \emptyset$   **return**
2   $S \leftarrow \{J_i \in Q : d(J_i, \alpha, t) \leq P/|Q|\}$
3   **if** $S = \emptyset$
4       **for** each $J_i \in Q$
5           $a(J_i, \alpha, t) \leftarrow P/|Q|$
6       **return**
7   **else**
8       **for** each $J_i \in S$
9           $a(J_i, \alpha, t) \leftarrow d(J_i, \alpha, t)$
10      DEQ$(\alpha, t, Q - S, P - \sum_{J_i \in S} d(J_i, \alpha, t))$

ROUND-ROBIN $(\alpha, t, Q, P)$
1   $S \leftarrow$ the first $P$ jobs from $Q$
2   **for** each $J_i \in S$
3       $a(J_i, \alpha, t) \leftarrow 1$
4       mark $J_i$

RAD $(\alpha, t, \mathcal{J}, P)$
1   $Q \leftarrow \{J_i \in \mathcal{J} : J_i$ is unmarked and $\alpha$-active$\}$
2   $Q' \leftarrow \{J_i \in \mathcal{J} : J_i$ is marked and $\alpha$-active$\}$
3   **if** $|Q| > P$
4       ROUND-ROBIN$(\alpha, t, Q, P)$
5   **else**
6       Move $\min(|Q'|, P - |Q|)$ jobs from $Q'$ to $Q$
7       DEQ$(\alpha, t, Q, P)$
8       unmark all jobs
```

Figure 2: Pseudo-code for RAD algorithm, which includes the main procedure RAD, and two sub-procedures DEQ and ROUND-ROBIN.

deterministic online non-clairvoyant algorithm.

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 tasks in the K-DAG, based on the lower bounds [5] 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 times:

$$\begin{aligned} \mathrm{T}^*(\mathcal{J}) &\geq \max_{J_i \in \mathcal{J}} (r(J_i) + T_\infty(J_i)) \ , \\ \mathrm{T}^*(\mathcal{J}) &\geq \max_{\alpha=1,\ldots,K} (T_1(\mathcal{J}, \alpha)/P_\alpha) \ . \end{aligned}$$

We will now show a lower bound on the competitive ratio for any deterministic online algorithm.

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

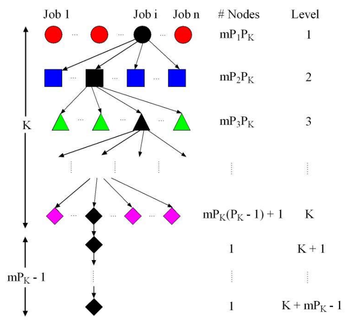

Figure 3: This example shows a set of n jobs that forces any deterministic online scheduler to have a competitive ratio of $K + 1 - K/P_{max}$ in terms of makespan in an adversary setting. The total number of nodes at each level is indicated on the right of the figure.

Proof. Without loss of generality, 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. Job J_i's critical path is highlighted by the darkened nodes in the figure with length $T_\infty(J_i) = K + mP_K - 1$. Level 1 of J_i has one unit of 1-task. Each subsequent level $\alpha \in \{2, \ldots, K - 1\}$ has exactly $mP_\alpha P_K$ units of α-task, all of which depend on a single task from the previous level. Level K 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 as possible by the 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 non-clairvoyant 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.

Let $m \gg K$ so that $(K - 1)/m$ approaches 0. Thus, the competitive ratio is

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

5 Makespan Analysis

This section shows that K-RAD is $(K + 1 - 1/P_{max})$-competitive with respect to the makespan. Suppose that $\mathrm{T}(\mathcal{J})$ is the makespan of job set $\mathcal{J}$ scheduled by K-RAD. Let $I = [1, 2, \ldots, \mathrm{T}(\mathcal{J})]$ denote the time interval in which K-RAD schedules the job set. Since $\mathcal{J}$ denotes a job set with arbitrary job release time, 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 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, 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.

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*

$$\mathrm{T}(\mathcal{J}) \le \sum_{\alpha=1}^{K} \frac{T_1(\mathcal{J}, \alpha)}{P_\alpha} + \left(1 - \frac{1}{P_{max}}\right) \max_{J_i \in \mathcal{J}} \left(T_\infty(J_i) + r(J_i)\right) \qquad (2)$$

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., $|R(J_k)| = r(J_k)$.

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 $\sum_{\alpha=1}^{K} T_1'(\mathcal{J}, \alpha) \ge |R(J_k)| = r(J_k)$.

(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., $|S(J_k)| \le T_\infty(J_k)$.

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

(3) To bound $|D(J_k)|$: Let $D(J_k, \alpha)$ denote the set of α-deprived steps for job J_k. According to K-RAD, on each deprived step $t \in D(J_k, \alpha)$, all α-processors must have been allotted 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. Since the total amount of α-work done on $D(J_k, \alpha)$ steps is $T_1(\mathcal{J}, \alpha) - T_1'(\mathcal{J}, \alpha) - T_1''(\mathcal{J}, \alpha)$ and $D(J_k) = \cup_{\alpha=1}^{K} D(J_k, \alpha)$, we have

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

To bound $\mathrm{T}(\mathcal{J})$: from the bounds for $|R(J_k)|$, $|S(J_k)|$ and $|D(J_k)|$, we obtain the makespan of the job set $\mathcal{J}$,

$$\begin{aligned} &\mathrm{T}(\mathcal{J}) \\ &= |R(J_k)| + |S(J_k)| + |D(J_k)| \\ &\le r(J_k) + |S(J_k)| + \sum_{\alpha=1}^{K} \frac{T_1(\mathcal{J}, \alpha)}{P_\alpha} - \frac{r(J_k) + |S(J_k)|}{P_{max}} \\ &\le \sum_{\alpha=1}^{K} \frac{T_1(\mathcal{J}, \alpha)}{P_\alpha} + \left(1 - \frac{1}{P_{max}}\right) \max_{J_i \in \mathcal{J}} \left(T_\infty(J_i) + r(J_i)\right) . \end{aligned}$$

□

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 any idle interval, the makespan can be bounded by Inequality (2). Since $\sum_{\alpha=1}^{K} T_1(\mathcal{J}, \alpha)/P_\alpha \le K \max_{\alpha=1,\ldots,K} T_1(\mathcal{J}, \alpha)/P_\alpha$ and both $\max_{\alpha=1,\ldots,K} T_1(\mathcal{J}, \alpha)/P_\alpha$ and $\max_{J_i \in \mathcal{J}} (T_\infty(J_i) + r(J_i))$ are lower bounds for makespan. Therefore, in this case we obtain $\mathrm{T}(\mathcal{J}) \le (K + 1 - 1/P_{max})\,\mathrm{T}^*(\mathcal{J})$.

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, an optimal scheduler $\mathcal{S}$ completes $\mathcal{J}_1$ 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 $\mathrm{T}(\mathcal{J}_2) \le (K+1-1/P_{max})\mathrm{T}^*(\mathcal{J}_2)$

time steps. The makespan of job set $\mathcal{J}$ scheduled by K-RAD is then

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

Hence, in both cases, K-RAD achieves $(K + 1 - 1/P_{max})$-competitive with respect to the makespan. □

Since the competitive ratio of K-RAD matches the lower bound given in Section 4, K-RAD is optimal for a deterministic online algorithm in terms of makespan.

6 Mean Response Time Lower Bounds

We present two lower bounds on the mean response time for scheduling any batched job set. We first introduce some useful definitions.

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

$$\text{sq-sum}(\langle a_i \rangle) = \sum_{i=1}^{m} (m - i + 1) a_{f(i)} . \qquad (3)$$

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 (3), we obtain the following equivalent definition for the squashed sum of list $\langle a_i \rangle$

$$\text{sq-sum}(\langle a_i \rangle) = \min_{g \in \Upsilon} \sum_{i=1}^{m} (m - i + 1) a_{g(i)} , \qquad (4)$$

where Υ is the set of all permutations on $\{1, 2, \ldots, m\}$.

Definition 5 The ***squashed α-work area*** of a job set $\mathcal{J}$ on P_α number of α-processors is $\mathrm{swa}(\mathcal{J}, \alpha) = \text{sq-sum}(\langle T_1(J_i, \alpha) \rangle)/P_\alpha$, where $T_1(J_i, \alpha)$ is the α-work of job $J_i \in \mathcal{J}$. The ***aggregate span*** of a job set $\mathcal{J}$ is $T_\infty(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} T_\infty(J_i)$ 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 [9, 27, 28] 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:

$$\begin{aligned}
\overline{\mathrm{R}^*}(\mathcal{J}) &\geq T_\infty(\mathcal{J}) / |\mathcal{J}| , \\
\overline{\mathrm{R}^*}(\mathcal{J}) &\geq \max_{\alpha=1,\ldots,K} \mathrm{swa}(\mathcal{J}, \alpha) / |\mathcal{J}| .
\end{aligned}$$

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 Mean Response Time Analysis

In this section, we will analyze the mean response time of K-RAD algorithm. We first present a supporting lemma and a notation to assist the analysis.

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*

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

Proof Sketch. Intuitively, when the algebraic sum of list $\langle s_i \rangle$ is given by a fixed amount P, the squashed sum of $\langle s_i \rangle$ is minimized if the values are distributed only on the largest elements of $\langle s_i \rangle$. Moreover, the squashed sum of $\langle s_i \rangle$ is always no more than the squashed sum difference between $\langle a_i \rangle$ and $\langle b_i \rangle$. Please refer to our technical report [14] for detailed derivation. □

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 of a job $J_i \in \mathcal{J}$, denoted as $J_i\left(\overrightarrow{t}\right)$, is defined to be the portion of the job that executes on or after time step t. The t-suffix of the job set $\mathcal{J}$ is then defined as $\mathcal{J}\left(\overrightarrow{t}\right) = \left\{J_i\left(\overrightarrow{t}\right) : J_i \in \mathcal{J}\right\}$.

Now we will consider two cases for the system workload and analyze the performance of K-RAD in each case 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 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.

Analysis of K-RAD under light workload

The following theorem gives the competitive ratio of K-RAD for the mean response time under light 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 $\alpha = 1, \ldots, K$.*

Proof. We will show that the total response time of $\mathcal{J}$ can be bounded by

$$\mathrm{R}(\mathcal{J}) \leq \left(2 - \frac{2}{|\mathcal{J}| + 1}\right) \sum_{\alpha=1}^{K} \mathrm{swa}(\mathcal{J}, \alpha) + T_\infty(\mathcal{J}) . \qquad (5)$$

Since $\sum_{\alpha=1}^{K} \mathrm{swa}(\mathcal{J}, \alpha) \leq K \max_{\alpha=1,\ldots,K} \mathrm{swa}(\mathcal{J}, \alpha)$ and both $\max_{\alpha=1,\ldots,K} \mathrm{swa}(\mathcal{J}, \alpha)$ and $T_\infty(\mathcal{J})$ are lower bounds for the total response time, Inequality (5) indicates that K-RAD is $(2K+1-2K/(|\mathcal{J}|+1))$-competitive with respect to the total response time, or the mean response time under light workload. We will prove Inequality (5) by induction on the t-suffix of the job set $\mathcal{J}\left(\overrightarrow{t}\right)$.

Basis: $t = \mathrm{T}(\mathcal{J}) + 1$. 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: $1 \leq t \leq \mathrm{T}(\mathcal{J})$. Let $n = \left|\mathcal{J}\left(\overrightarrow{t}\right)\right|$ denote the number of uncompleted jobs at time t and let $c = 2 - 2/(n+1)$. Since $n \geq \left|\mathcal{J}\left(\overrightarrow{t+1}\right)\right|$, we have $c \geq 2 - 2/\left(\left|\mathcal{J}\left(\overrightarrow{t+1}\right)\right| + 1\right)$. Suppose that Inequality (5) holds at $t+1$, which implies

$$\mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) \leq c \sum_{\alpha=1}^{K} \mathrm{swa}\left(\mathcal{J}\left(\overrightarrow{t+1}\right), \alpha\right) + T_\infty\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) . \tag{6}$$

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

$$\mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) \leq c \sum_{\alpha=1}^{K} \mathrm{swa}\left(\mathcal{J}\left(\overrightarrow{t}\right), \alpha\right) + T_\infty\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) . \tag{7}$$

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 $t+1$:

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

Given the induction hypothesis (Inequality (6)), we need only prove the following inequality in order to prove our claim (Inequality (7)),

$$\Delta r \leq c \sum_{\alpha=1}^{K} \Delta \mathrm{swa}(\alpha) + \Delta T_\infty . \tag{8}$$

We divide the proof of Inequality (8) into four steps.

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

(2) To bound ΔT_∞: At any time t, the uncompleted 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 t, respectively. If $J_i \in \mathcal{JS}(t)$, the span of J_i must reduce by 1. If $J_i \in \mathcal{JD}(t)$, the span of J_i never increases. Therefore, the aggregate span of $\mathcal{J}$ must reduce by at least $|\mathcal{JS}(t)|$ during time step t, so we have $\Delta T_\infty \geq |\mathcal{JS}(t)|$.

(3) To bound $\Delta \mathrm{swa}(\alpha)$: At any time t, if a job J_i is α-deprived, J_i has α-allotment $a(J_i, \alpha, t) = \bar{p}(\alpha, t)$, where $\bar{p}(\alpha, t)$ is the mean deprived allotment for α-processors. If a job J_i is α-satisfied, J_i's α-allotment is equal to its α-desire, i.e., $a(J_i, \alpha, t) = d(J_i, \alpha, t)$. Let $\mathcal{JS}(\alpha, t)$ and $\mathcal{JD}(\alpha, t)$ denote the set of α-satisfied and the set of α-deprived jobs at time step t respectively. We will consider two cases.

Case 1. 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 $\Delta \mathrm{swa}(\alpha) \geq 0$.

Case 2. If there exist α-deprived jobs at time t, i.e., $|\mathcal{JD}(\alpha, t)| > 0$. In this case, according to K-RAD, *all* α-processors must have been allotted to the jobs. The α-deprived jobs receive the same mean deprived allotment $\bar{p}(\alpha, t)$, which is no less than the allotment for the α-satisfied jobs. With this scenario in mind, Lemma 4 bounds the change in the squashed sum of α-work. For $J_i \in \mathcal{J}$, let $a_i = T_1\left(J_i\left(\overrightarrow{t+1}\right), \alpha\right)$ and $b_i = T_1\left(J_i\left(\overrightarrow{t}\right), \alpha\right)$. Since $\sum_{J_i \in \mathcal{J}} a(J_i, \alpha, t) = P_\alpha$, according to Lemma 4, we have

$$\text{sq-sum}(\langle b_i \rangle) - \text{sq-sum}(\langle a_i \rangle) \geq \frac{P_\alpha\left(|\mathcal{JD}(\alpha, t)| + 1\right)}{2} . \tag{9}$$

From Inequality (9) and Definition 5, we get for $\alpha = 1, \ldots, K$, $\Delta \mathrm{swa}(\alpha) \geq \left(|\mathcal{JD}(\alpha, t)| + 1\right)/2$

(4) To derive Inequality (8): Since the uncompleted jobs at time t can be partitioned as $\mathcal{J}\left(\overrightarrow{t}\right) = \mathcal{JS}(t) \cup \mathcal{JD}(t)$, we have $|\mathcal{JS}(t)| + |\mathcal{JD}(t)| = n$. Also because $\mathcal{JD}(t) = \cup_{\alpha=1}^{K} \mathcal{JD}(\alpha, t)$, we can obtain $\sum_{\alpha=1}^{K} |\mathcal{JD}(\alpha, t)| \geq |\mathcal{JD}(\alpha, t)| = n - |\mathcal{JS}(t)|$. Let $\mathcal{K} = \{1, \ldots, K\}$ and let $\mathcal{K}' = \{\alpha \in \mathcal{K} : |\mathcal{JD}(\alpha, t) > 0|\}$. Suppose $\left|\mathcal{K}'\right| \geq 1$, then from the bounds for Δr, ΔT_∞ and $\Delta \mathrm{swa}(\alpha)$, we get

$$\begin{aligned} & c \sum_{\alpha=1}^{K} \Delta \mathrm{swa}(\alpha) + \Delta T_\infty \\ \geq \quad & \frac{2n}{n+1} \sum_{\alpha \in \mathcal{K}'} \frac{|\mathcal{JD}(\alpha, t)| + 1}{2} + |\mathcal{JS}(t)| \\ \geq \quad & \frac{n}{n+1}\left(n - |\mathcal{JS}(t)| + \left|\mathcal{K}'\right|\right) + |\mathcal{JS}(t)| \\ \geq \quad & n - |\mathcal{JS}(t)| \frac{n}{n+1} + |\mathcal{JS}(t)| \geq n = \Delta r . \end{aligned}$$

Thus we have shown that Inequality (8) holds. If $\left|\mathcal{K}'\right| = 0$ on the other hand, we have $\Delta T_\infty = |\mathcal{JS}(t)| = n$. Inequality (8) again holds trivially. The proof is complete. □

Note that in the case where $K = 1$ for homogeneous resource scheduling, Theorem 5 indicates that RAD is $(3 - 2/(|\mathcal{J}| + 1))$-competitive with respect to the mean response time. Combining this result with our previous work in [13], it is not hard to show that RAD is $(3 - 2/(|\mathcal{J}| + 1))$-competitive under heavy workload as well. 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 under arbitrary workloads.

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 $|\mathcal{J}(\alpha, t)| > P_\alpha$, K-RAD will utilize both DEQ and RR algorithms. Theorem 6 gives the competitive ratio for the mean response time produced by K-RAD scheduler under this more general case. As Theorem 5, Theorem 6 can be shown using mathematical induction on the suffix of a job set $\mathcal{J}$. Please refer to our technical report [14] for detailed analysis.

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

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 is to develop scheduling models and algorithms that capture both functional and performance heterogeneity.

References

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

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

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

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

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

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

[7] Fabi A. Chudak and David 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.

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

[9] Xiaotie Deng and Patrick Dymond. On multiprocessor system scheduling. In *SPAA*, pages 82–88, Padua, Italy, 1996.

[10] Jeff Edmonds, Donald D. Chinn, Tim Brecht, and Xiaotie Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics (extended abstract). In *STOC*, pages 120–129, 1997.

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

[12] Yuxiong He, Wen Jing Hsu, and Charles E. Leiserson. Provably efficient two-level adaptive scheduling. In *JSSPP*, Saint-Malo, France, 2006.

[13] Yuxiong He, Wen Jing Hsu, and Charles E. Leiserson. Provably efficient adaptive scheduling through equalized allotments. In *IPDPS*, Long Beach, California, USA, 2007.

[14] Yuxiong He, Hongyang Sun, and Wen Jing Hsu. Adaptive scheduling of parallel jobs on functionally heterogeneous resources. Technical Report 07-01, NTU, http://people.csail.mit.edu/yxhe/paper/heterogeneous07.pdf.

[15] Ashfaq Khokhar, Viktor K. Prasanna, Muhammad Shaaban, and Cho-Li Wang. Heterogeneous supercomputing: Problems and issues. In *Workshop on Heterogeneous Processing*, 1992.

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

[17] 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.

[18] Frank Thomson Leighton, Bruce M. Maggs, and Satish B. Rao. Packet routing and job-shop scheduling in o (congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

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

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

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

[22] Rajeev Motwani, Steven Phillips, and Eric Torng. Non-clairvoyant scheduling. In *SODA*, pages 422–431, Austin, Texas, United States, 1993.

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

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

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

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

[27] John Turek, Walter Ludwig, Joel L. Wolf, Lisa Fleischer, Prasoon Tiwari, Jason Glasgow, Uwe Schwiegelshohn, and Philip S. Yu. Scheduling parallelizable tasks to minimize average response time. In *SPAA*, pages 200–209, Cape May, New Jersey, United States, 1994.

[28] John Turek, Uwe Schwiegelshohn, Joel L. Wolf, and Philip S. Yu. Scheduling parallel tasks to minimize average response time. In *SODA*, pages 112–121, Philadelphia, PA, USA, 1994.

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

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