Efficient Adaptive Scheduling of Multiprocessors with Stable Parallelism Feedback

Hongyang Sun, *Student Member*, *IEEE*, Yangjie Cao, and Wen-Jing Hsu, *Senior Member*, *IEEE*

Abstract—With proliferation of multicore computers and multiprocessor systems, an imminent challenge is to efficiently schedule parallel applications on these resources. In contrast to conventional static scheduling, adaptive schedulers that dynamically allocate processors to jobs possess good potential for improving processor utilization and speeding up job's execution. In this paper, we focus on adaptive scheduling of malleable jobs with periodic processor reallocations based on parallelism feedback of the jobs and allocation policy of the system. We present an efficient adaptive scheduler ACDEQ that provides parallelism feedback using an adaptive controller A-CONTROL and allocates processors based on the well-known Dynamic Equipartitioning algorithm (DEQ). Compared to A-GREEDY, an existing adaptive scheduler that experiences feedback instability thus incurs unnecessary scheduling overheads, we show that A-CONTROL achieves much more stable feedback among other desirable control-theoretic properties. Furthermore, we analyze algorithmically the performances of ACDEQ in terms of its response time and processor waste for an individual job as well as makespan and total response time for a set of jobs. To the best of our knowledge, ACDEQ is the first multiprocessor scheduling algorithm that offers both control-theoretic and algorithmic guarantees. We further evaluate ACDEQ via simulations by using Downey's parallel job model augmented with internal parallelism variations. The results confirm its improved performances over AGDEQ, and they show that ACDEQ excels especially when the scheduling overhead becomes high.

Index Terms—Adaptive scheduling, competitive analysis, control-theoretic analysis, malleable parallel jobs, multiprocessors, nonclairvoyant scheduling, parallelism feedback, stability, two-level scheduling.

1 INTRODUCTION

With proliferation of multicore computers and increasing use of multiprocessor systems, more software applications are designed to execute on multiple processors. How to efficiently schedule these parallel applications to fully exploit the multiprocessor resources has therefore become an imminent challenge. Conventional approaches to scheduling parallel jobs with a fixed number of processors (called *static scheduling*) can often cause processor wastes and job execution delays if the job's parallelism varies with time. In fact, many parallel jobs can be executed with a variable number of processors during their execution lifetime, and they are referred to as *malleable jobs* [20]. *Adaptive scheduling* that periodically adjusts processor allocations to malleable jobs hence has the apparent benefits of improving processor utilization and speeding up job's execution.

The two-level framework [1], [20], [24] has provided a convenient approach for adaptive scheduling on multiprocessors. Specifically, the scheduling of malleable jobs by a two-level adaptive scheduler is divided into two distinct levels and the processors are periodically reallocated after each *scheduling quantum* based on the interaction of the two levels. Within each quantum, a *task scheduler* at the job level schedules the tasks of each job, and based on the job's execution provides feedback to the system indicating its *processor desire*, an estimated number of processors the job requires, for the next quantum. At the system level, an *OS allocator* decides the processor allocation for each job in the next quantum according to the desires of all jobs and the system allocation policy. Since information about jobs' characteristics is generally unavailable, the challenge is for the task scheduler and the OS allocator to make feedback and allocation decisions in an *online nonclairvoyant* fashion [16], [24], [38], i.e., without assuming jobs' release time, work requirement, future parallelism, etc.

Under this two-level framework, Agrawal et al. [1] proposed a task scheduler A-GREEDY that provides parallelism feedback to the system for each job in a scheduling quantum using a simple multiplicative-increase multiplicative-decrease strategy. They analyzed the performance of A-GREEDY for an individual job and showed that it is efficient in terms of the job's response time and processor utilization. He et al. [24] later showed that A-GREEDY can be coupled with DEQ (i.e., Dynamic Equipartitioning) OS allocator [37], [48] to achieve competitive performances in terms of the makespan and total response time for a set of jobs.

Despite its good theoretical performances, simple analysis of A-GREEDY also reveals certain problems in terms of the transient response of its processor desires. In particular, due to the multiplicative-increase multiplicative-decrease strategy, A-GREEDY suffers from desire instability and hence possible instability in allocation, even when the job's parallelism stays constant. (See Sections 3 and 5 for elaboration.) The unstable problem can cause potential difficulties in managing the processor resources, such as

- H. Sun and W.-J. Hsu are with the School of Computer Engineering, Nanyang Technological University, Block N4, Nanyang Avenue, Singapore 639798. E-mail: {sunh0007, hsu}@ntu.edu.sg.
- Y. Cao is with the School of Electronic and Information Engineering, Xi'an Jiaotong University, No. 28, Xianning West Road, Xi'an, Shaanxi 710049, P.R. China. E-mail: caoyj@stu.xjtu.edu.cn.

Manuscript received 11 July 2009; revised 4 Nov. 2009; accepted 6 Nov. 2009; published online 1 June 2010.
Recommended for acceptance by J. Weissman.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-07-0312.
Digital Object Identifier no. 10.1109/TPDS.2010.121.

1045-9219/11/$26.00 © 2011 IEEE Published by the IEEE Computer Society

unnecessary processor waste and job execution delays as well as excessive reallocation overheads, loss of localities, etc., which tend to become worse with increased job parallelism.

In this paper, we present an efficient yet stable task scheduler called A-CONTROL under the same two-level framework. Unlike A-GREEDY, which responds to the job's parallelism variations in a discrete manner by a multiplicative factor each time, A-CONTROL calculates its processor desires continuously based on principles from *control theory*, which has been previously applied to scheduling real-time systems [35], [40] and designing computing applications [28]. In particular, we show using control-theoretic analysis that the processor desires calculated by A-CONTROL are able to achieve much improved transient and steady-state performances that A-GREEDY fails to attain.

We further prove from an algorithmic perspective the performances of the two-level adaptive scheduler ACDEQ, which combines the task scheduler A-CONTROL with OS allocator DEQ, in terms of response time and processor waste for a single job as well as makespan and total response time for a set of jobs. While the results in [1], [24] are obtained by modeling a malleable job as a directed acyclic graph (dag) and restricting A-GREEDY to execute the tasks of the job in a greedy manner [6], [23], in this paper, we choose the parallelism profile model [8], [14], [31] for malleable jobs. As shown in [13], [14], [26], the two models have no fundamental difference in demonstrating the algorithmic performances of an adaptive scheduler; the profile model, however, allows us to focus on processor desire calculation, which is the key aspect of task scheduling, without restriction on the specific execution strategy. In the preliminary version [47] of this paper, we nevertheless showed that similar results can be derived by modeling a job as a dag.

In our analysis, we make use of two intrinsic job characteristics, namely the *work* and the *span*, which represent the total amount of time to execute a job with one processor and an infinite number of processors of unit speed, respectively [1], [6], [24]. Moreover, we identify another job characteristic, which we call the *transition factor*, to characterize the variation of a job's average parallelism between two consecutive quanta. We will formally define the transition factor in Section 6. Intuitively, it indicates the inherent difficulty to adaptively schedule a malleable job in a nonclairvoyant fashion. We argue that this factor better reflects a scheduler's performance; hence, it should be incorporated into the analysis. The following summarizes our main results:

- A-CONTROL achieves good transient and steady-state performances in terms of its processor desire calculation. Specifically, the processor desires satisfy bounded-input bounded-output (BIBO) stability, zero steady-state error, zero overshoot, and user-configurable convergence rate v, where $0 \le v < 1$.
- ACDEQ completes any job J_i in a set $\mathcal{J}$ of jobs with work w_i, span l_i, and transition factor C_i on P processors of speed s in $R_s(J_i) \le \frac{2w_i}{s\overline{P}} + \frac{C_i+1-2v}{s(1-v)} l_i$ time, and wastes no more than $X_s(J_i) \le \frac{C_i(1-v)}{1-C_i v} w_i$ processor cycles, where $\overline{P} = P/|\mathcal{J}|$ denotes the equal share of processors for each job in the job set and $v < 1/C_i$ denotes the convergence rate.
- For any job set $\mathcal{J}$, ACDEQ achieves makespan $M_s(\mathcal{J}) \le (\frac{C+1-2Cv}{1-Cv} + \frac{C+1-2v}{1-v}) M_s^*(\mathcal{J})$ on speed s processors, where $M_s^*(\mathcal{J})$ denotes the makespan of the optimal scheduler on processors of the same speed, C denotes the maximum transition factor of all jobs in $\mathcal{J}$, and $v < 1/C$ denotes the convergence rate. Moreover, using resource augmentation [30], [41], we show that on processors of speed s, where $s = 4 + \epsilon$ for any $\epsilon > 0$, ACDEQ achieves total response time $R_s(\mathcal{J}) \le (2 + \frac{10+2C-12v}{\epsilon(1-v)}) R_1^*(\mathcal{J})$, where $R_1^*(\mathcal{J})$ denotes the total response time of the optimal scheduler for job set $\mathcal{J}$ on unit-speed processors.

Comparing with AGDEQ [1], [24], [46], these algorithmic bounds of ACDEQ suggest that it tends to perform better when jobs have slow parallelism variations, and hence small transition factors. We confirm our analysis by further conducting simulations using malleable jobs generated from Downey's model [15] and augmented with a set of internal parallelism variations. The results show that the task scheduler A-CONTROL indeed possesses much improved transient and steady-state performances than A-GREEDY on these parallelism variations. Moreover, ACDEQ also demonstrates its superior performances in terms of the algorithmic metrics over a wide range of workloads, especially when the scheduling overhead becomes high.

The rest of this paper is organized as follows: Section 2 formally introduces the scheduling model and the objective functions. Section 3 briefly reviews the A-GREEDY task scheduler. Section 4 introduces the ACDEQ two-level adaptive scheduler. Section 5 provides the control-theoretic analysis of A-CONTROL, followed by the algorithmic analysis of ACDEQ in Section 6. Our simulation results are presented in Section 7. Section 8 reviews some related work, and Section 9 concludes the paper.

2 MODELS AND OBJECTIVES

We model a malleable job using a parallelism profile [8], [13], [31], which specifies the number of processors the job can effectively utilize at any time during its execution. Adopting the notations in [16], [17], a job can have multiple phases of speedup functions. In this paper, we assume that each phase has a linear speedup function up to a certain degree of parallelism, beyond which no further speedup can be gained. Specifically, we consider a set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of n jobs to be scheduled on P processors. Each job J_i contains k_i phases $\langle J_i^1, J_i^2, \ldots, J_i^{k_i} \rangle$, and each phase J_i^k has an amount of *work* w_i^k, which represents the amount of time to execute the phase with a single unit-speed processor as well as a linear *speedup function* Γ_i^k up to parallelism h_i^k, where $h_i^k \ge 1$. The *span* l_i^k of phase J_i^k, which represents the amount of time to execute the phase with h_i^k or more processors of unit speed, is therefore $l_i^k = w_i^k / h_i^k$. The *total work* w_i of job J_i is given by $w_i = \sum_{k=1}^{k_i} w_i^k$, and the *total span* l_i of the job is $l_i = \sum_{k=1}^{k_i} l_i^k$. Suppose that at time t job J_i is in its kth phase and is allocated $a_i(t)$ processors of speed s, then its effective speedup or execution rate is given by $\Gamma_i^k(a_i(t)) = a_i(t)s$ if $a_i(t) \le h_i^k$ and $\Gamma_i^k(a_i(t)) = h_i^k s$ if $a_i(t) > h_i^k$. While this model of execution applies to any algorithm, the only difference between two task schedulers thus lies in their strategies to calculate processor desires which are the key aspects that essentially differentiate their performances.

A schedule for any set $\mathcal{J}$ of jobs specifies the number $a_i(t)$ of processors allocated to each job J_i at any time t. In order for the schedule to be valid, we require that at any time t the total processor allocation is not more than the total number of processors, i.e., $\sum_{i=1}^{n} a_i(t) \leq P$. Let r_i denote the *release time* of job J_i. Without loss of generality, we assume that the first job in the job set is released at time 0. Let c_i^k denote the *completion time* of the kth phase of job J_i, and let $c_i = c_i^{k_i}$ denote the *completion time* of job J_i. We also require that a valid schedule must complete all jobs in finite amount of time and cannot begin to execute a phase of a job unless it has completed all its previous phases, i.e., $r_i = c_i^0 < c_i^1 < \cdots < c_i^{k_i} < \infty$, and $\int_{c_i^{k-1}}^{c_i^k} \Gamma_i^k(a_i(t))dt = w_i^k$ for all $1 \leq k \leq k_i$.

The objectives include both individual job performances and job set performances. In particular, for each job J_i, we bound its *response time* $R_s(J_i)$ and *processor waste* $X_s(J_i)$ on speed s processors, where the response time is the duration between the job's release and completion, i.e., $R_s(J_i) = c_i - r_i$, and the processor waste is the total processor allocations to the job that are not utilized to do useful work, i.e., $X_s(J_i) = \int_0^\infty (a_i(t)s)dt - w_i$. For a job set $\mathcal{J}$, we analyze its *makespan* $M_s(\mathcal{J})$ and *total response time* $R_s(\mathcal{J})$ on speed s processors, where the makespan is the completion time of the last completed job in the job set, i.e., $M_s(\mathcal{J}) = \max_{J_i \in \mathcal{J}} c_i$, and the total response time is the sum of response time from all jobs in the job set, i.e., $R_s(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} R_s(J_i)$.

While response time and processor waste are bounded in terms of job characteristics such as work and span, makespan is bounded in terms of performance of the optimal scheduler using *competitive analysis* [7]. An online scheduler is said to be *c-competitive* in terms of makespan if $M_s(\mathcal{J}) \leq c \cdot M_s^*(\mathcal{J})$ holds for any job set $\mathcal{J}$, where $M_s(\mathcal{J})$ and $M_s^*(\mathcal{J})$ denote the makespan of a job set $\mathcal{J}$ scheduled by the online scheduler and the optimal, respectively, both on speed s processors. For the total response time, we employ *resource augmentation analysis* [30], [41], which equips the online scheduler with extraspeed processors than the optimal. Specifically, let $R_s(\mathcal{J})$ denote the total response time of job set $\mathcal{J}$ scheduled by the online scheduler using speed s processors, where $s > 1$, and let $R_1^*(\mathcal{J})$ denote the total response time of job set $\mathcal{J}$ scheduled by the optimal using unit-speed processors. Then, the online scheduler is said to be *s-speed c-competitive* in terms of the total response time if $R_s(\mathcal{J}) \leq c \cdot R_1^*(\mathcal{J})$ holds for any job set $\mathcal{J}$.

In addition, for a two-level adaptive scheduler, we also study the *transient* and *steady-state* performances in terms of certain control-theoretic properties of its processor desire calculation, which include stability, steady-state error, overshoot, and convergence. We will formally introduce these properties when we present the control-theoretic analysis in Section 5. Basically, they capture the quality of feedback generated by the adaptive scheduler in response to the job's parallelism variations, which will affect the practical performance of the algorithm.

3 Revisit Task Scheduler: A-Greedy

We briefly review the A-GREEDY task scheduler proposed by Agrawal et al. [1] to schedule individual malleable jobs. In particular, the processor desires of A-GREEDY are calculated using a multiplicative-increase multiplicative-decrease strategy based on the job's execution characteristics in the previous quantum.

Let t_q denote the time when scheduling quantum q starts. The work $w_i(q)$ completed for job J_i in quantum q is given by $w_i(q) = \int_{t_q}^{t_q+L} \Gamma_i^{k_t}(a_i(q))dt$, where k_t is the phase job J_i is executing at time t, $a_i(q)$ is the processor allocation of the job in quantum q, and L is the quantum length. Job J_i is said to be *efficient* in quantum q if work $w_i(q)$ completed is at least δ fraction of the maximum amount of work that can be done in the quantum, i.e., $w_i(q) \geq \delta a_i(q)sL$, where $0 < \delta < 1$ is called the *utilization parameter*; otherwise, it is *inefficient* if $w_i(q) < \delta a_i(q)sL$. Furthermore, the job is said to be *satisfied* in quantum q if its processor allocation $a_i(q)$ is at least the processor desire $d_i(q)$, i.e., $a_i(q) \geq d_i(q)$; otherwise, it is *deprived* if $a_i(q) < d_i(q)$. Based on the efficient-inefficient classification and the satisfied-deprived classification, the processor desire $d_i(q+1)$ of job J_i in the next quantum $q+1$ is calculated as follows:

$$d_i(q+1) = \begin{cases} d_i(q) \cdot \rho, & \text{if efficient and satisfied,} \\ d_i(q)/\rho, & \text{if inefficient,} \\ d_i(q), & \text{if efficient and deprived,} \end{cases}$$

where $\rho > 1$ is the *responsiveness parameter*.

The rationale of A-GREEDY is that it attempts to exploit the correlation between the parallelism of a job in two adjacent quanta, though the existence of such correlation is not explicitly assumed. Specifically, if the allocated processors in quantum q are not utilized efficiently, then the parallelism of the job may not be as high. Therefore, the processor desire will be reduced by a factor of ρ for the next quantum $q+1$. If the allocated processors are utilized efficiently and the processor desire is satisfied, then the parallelism of the job could be even higher. Thus, the processor desire will be increased by a factor of ρ. Last, if the allocated processors are utilized efficiently but the desire is deprived, then it is not known whether the processors could still be efficiently utilized had the desire been satisfied. Therefore, the processor desire is not changed for the next quantum.

It is not difficult to see, however, that if the parallelism of the job is indeed correlated, e.g., if the parallelism stays constantly at a certain level for sufficiently long time, then the processor desires of A-GREEDY tend to become unstable as it oscillates around the target parallelism. The instability may cause much degradation in the actual performance of A-GREEDY due to the extra context switching overheads, loss of locality, etc. In this paper, we aim at an improved task scheduling algorithm that will mitigate the problem.

4 Stable Two-Level Scheduler: ACDEQ

We propose a two-level adaptive scheduler ACDEQ, which combines a task scheduler A-CONTROL with the OS allocator DEQ [37], [48]. In particular, A-CONTROL calculates the processor desires for a job using an adaptive controller that maintains the job's desire stability among other control-theoretic properties. In this section, we describe ACDEQ in detail.

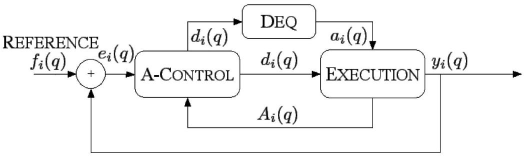


Fig. 1. Feedback control structure of ACDEQ.

Fig. 1 shows the feedback control structure of ACDEQ for an individual job J_i. In each quantum q, the output of A-CONTROL is the processor desire $d_i(q)$. The desire is sent to OS allocator DEQ, which gives job J_i a processor allotment $a_i(q)$ based on the desires of all jobs as well as its allocation policy. During quantum q, job J_i is executed with $a_i(q)$ processors while the work $w_i(q)$ completed and the span $l_i(q)$ reduced for the job are collected. Specifically, let t_q denote the time quantum q starts, then the *quantum work* $w_i(q)$ is given by $w_i(q) = \int_{t_q}^{t_q+L} \Gamma_i^{k_t}(a_i(q))dt$, and the *quantum span* $l_i(q)$ is given by $l_i(q) = \int_{t_q}^{t_q+L} \Gamma_i^{k_t}(a_i(q))/h_i^{k_t}dt$, where k_t is the phase job J_i is executing at time t and L is the quantum length. The *quantum average parallelism* $A_i(q)$ is therefore $A_i(q) = w_i(q)/l_i(q)$. The output $y_i(q)$ is defined as $y_i(q) = d_i(q)/A_i(q)$, and it is compared with the reference $f_i(q)$ to produce an error term $e_i(q) = f_i(q) - y_i(q)$, which is used by the adaptive controller to calculate the processor desire $d_i(q+1)$ for quantum $q+1$. The controller applies the following integral control law [28]:

$$d_i(q+1) = d_i(q) + K_i(q+1)e_i(q), \tag{1}$$

where $K_i(q+1)$ denotes the *controller gain* for quantum $q+1$ and it determines how aggressively the controller responds to the job's parallelism. Note that the controller adjusts the processor desire based on the desire and the error of the previous quantum, and it is *adaptive* because the gain $K_i(q+1)$ is reset for each quantum based on the measurement $A_i(q)$ and some performance specifications. In the next section, we will present these specifications and show how to set the controller gain from control-theoretic perspective. As with A-GREEDY [1], the processor desire of job J_i in its first quantum is set to 1. Moreover, to ease our analysis, we assume that the jobs are much larger compared to the quantum length, and each job is only released and completed at the beginning of a quantum.[1]

Following the terminologies in [1], we define some notions. First, a job J_i is said to be *satisfied* in quantum q if its processor allocation is at least its processor desire, i.e., $a_i(q) \geq d_i(q)$; otherwise, if $a_i(q) < d_i(q)$, job J_i is *deprived*. In addition, job J_i is said to be *overallocated* in quantum q if its processor allocation is more than its average parallelism in the quantum, i.e., $a_i(q) > A_i(q)$; otherwise, the job is *underallocated* if $a_i(q) \leq A_i(q)$. Finally, job J_i is said to be *accounted* in a quantum if it is both deprived and underallocated; otherwise, the job is *deductible* if it is either satisfied or overallocated. Furthermore, we extend the concepts of accounted, deductible, etc. from quantum to time. For instance, job J_i is said to be accounted at time t if the job is accounted in the quantum which contains t.

Now, we briefly describe the DEQ OS allocator [37], [48], which is well known for its efficiency and fairness to allocate processors to jobs [8], [14], [24]. DEQ is a variants of Equipartitioning (EQUI) that divides the total number of processors equally among all active jobs at any time. In DEQ, however, if a job desires fewer processors than the equal share, it will not be allocated more processors than its desire, and the remaining processors will instead be given to the other jobs with higher desires. Let $\mathcal{J}(t)$ denote the set of active jobs at time t when quantum q begins. Based on the processor desires from all jobs in $\mathcal{J}(t)$, DEQ allocates the processors as follows:

```
1) **if** $|\mathcal{J}(t)| = 0$, **then** return;
2) **if** $\forall J_i \in \mathcal{J}(t)$, $d_i(q) > P/|\mathcal{J}(t)|$, **then** $\forall J_i \in \mathcal{J}(t)$,
   $a_i(q) = P/|\mathcal{J}(t)|$, and return;
3) **else** let $\mathcal{J}'(t) = \{J_i \in \mathcal{J}(t) : d_i(q) \leq P/|\mathcal{J}(t)|\}$.
   $\forall J_i \in \mathcal{J}'(t)$, $a_i(q) = d_i(q)$. Update $\mathcal{J}(t) = \mathcal{J}(t) - \mathcal{J}'(t)$
   and $P = P - \sum_{J_i \in \mathcal{J}'(t)} a_i(q)$. **Goto** Step 1.
```

In this paper, as in [8], [14], [16], [17], [24], we assume that the number of processors allocated to a job can be nonintegral. The fractional allocation is considered as time-sharing a processor with other jobs. Moreover, from the pseudocode of DEQ, we can see that if a job is deprived, then its processor allocation is at least the initial equal share $P/|\mathcal{J}(t)|$.

5 Control-Theoretic Analysis

The adaptive controller shown in Fig. 1 dynamically adjusts its controller gain based on the time-varying parallelism of the job and is referred to as *self-tuning regulator* [5] in control theory. In this section, we determine how the controller gain can be set for each scheduling quantum via control-theoretic analysis. Basically, we transform the system into z-domain, and employ pole placement strategy [28] by considering a set of transient and steady-state performance specifications, which directly apply to the scheduler when the job's average parallelism is constant (e.g., the job does not experience phase transitions). When the job's average parallelism changes (e.g., by making a transition from one phase to the next), our analysis will apply anew to the scheduler with respect to the job's new average parallelism. In case that the job's parallelism changes continuously, these specifications are unfortunately no longer applicable. In Section 7, we will empirically study the transient response of A-CONTROL under such scenarios using a few generic forms of parallelism variations.

Now, we focus on the scenario with constant average parallelism for a job. Assume that job J_i's average parallelism is A_i at some time, and it will stay constantly at A_i for sufficiently long. The controller gain K_i, which depends on the value of A_i, will also stay constant in the mean time. Ideally, the processor desire should match the job's average parallelism for a quantum to be efficient. Hence, the reference $f_i(q)$ of our scheduler is set to 1 in all quanta, which corresponds to a unit-step function. Thus, we can represent the REFERENCE, A-CONTROL, and EXECUTION shown in Fig. 1 in z-domain as follows:

- REFERENCE: $F_i(z) = z/(z-1)$.

[1] If a job is released or completed in the middle of a quantum, the performance bounds will increase by at most a constant additive factor.

- A-CONTROL: $G_i(z) = D_i(z)/E_i(z) = K_i/(z-1)$.
- EXECUTION: $S_i(z) = Y_i(z)/D_i(z) = 1/A_i$.

The closed-loop transfer function of the system can be derived accordingly as follows:

$$T_i(z) = \frac{Y_i(z)}{F_i(z)} = \frac{G_i(z)S_i(z)}{1 + G_i(z)S_i(z)} = \frac{K_i/A_i}{z - (1 - K_i/A_i)}. \qquad (2)$$

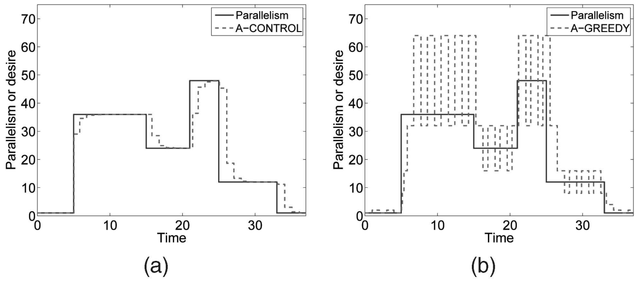
Clearly, the closed-loop is a first-order system with a single pole $p = 1 - K_i/A_i$. We adopt the following set of criteria [28] commonly used in control theory to place the pole by setting the value of controller gain K_i:

- *BIBO stability*. The system is BIBO stable, if given a bounded reference the processor desire is also bounded.
- *Steady-state error*. The steady-state error is given by the difference between the processor desire and the job's average parallelism after sufficiently long time, i.e., at steady state.
- *Maximum overshoot*. The maximum overshoot is the maximal difference between the transient processor desire and its steady-state value.
- *Convergence rate*. The convergence rate v indicates the speed at which the processor desire approaches the job's average parallelism, and is defined to be $v = \max_q(|d_i(q+1) - A_i|/|d_i(q) - A_i|)$.

Note that these four criteria directly apply to the output $y_i(q)$ of the closed-loop system. However, since $y_i(q)$ is defined to be $y_i(q) = d_i(q)/A_i(q)$ and the average parallelism $A_i(q)$ of the job is assumed to be constant for the period of consideration, the criteria also specify the transient and steady-state performances of the processor desire. We show that A-CONTROL has good performance in terms of these criteria in the following theorem:

Theorem 1. *Suppose that A-CONTROL schedules a job J_i whose average parallelism stays constantly at A_i for sufficiently long time. If the controller gain is set to $K_i = (1 - v)A_i$, where $v \in [0, 1)$, then the processor desires satisfy 1) BIBO stability, 2) zero steady-state error, 3) zero overshoot, and 4) convergence rate v.*

Proof. Using rules established in control theory [28], we prove the theorem based on the pole position in the z-plane, which is $p = 1 - K_i/A_i = 1 - (1 - v) = v$. For a first-order system to be BIBO stable, the pole needs to be within the unit circle, which is satisfied since $|p| = |v| < 1$. Applying Final Value Theorem [28] on the output $y_i(p)$, we get $\lim_{q\to\infty} y_i(q) = \lim_{z\to 1}(z-1)Y_i(z) = 1$. Hence, it has zero steady-state error. For a first-order system to have zero overshoot, the pole needs to be nonnegative. Finally, the convergence rate of a first-order system is exactly given by the value of its pole, which is independent of quantum q. □



(a)

(b)

Fig. 2. Transient and steady-state behaviors of (a) A-CONTROL, (b) A-GREEDY.

Since the average parallelism $A_i(q)$ is measured for each quantum $q \geq q_i$, where q_i is the first quantum for job J_i, the controller gain based on Theorem 1 is set to $K_i(q) = (1 - v)A_i(q - 1)$ for $q > q_i$. Substituting it into (1), we get the following relation on the processor desire:

$$d_i(q) = \begin{cases} vd_i(q-1) + (1-v)A_i(q-1), & \text{if } q > q_i, \\ 1, & \text{if } q = q_i, \end{cases} \qquad (3)$$

which is essentially the strategy for calculating the processor desire for each quantum. Note that v is a design parameter of A-CONTROL and can be configured differently for each job, but in this paper, we assume that it is set uniformly for all jobs. Of course, a special case is when the convergence rate is set to $v = 0$. This gives the fastest rate of convergence, or one-quantum convergence. The resulting processor desire for each quantum q is then equal to the job's average parallelism in the previous quantum, i.e., $d_i(q) = A_i(q - 1)$.

Fig. 2 compares the behaviors of A-CONTROL and A-GREEDY over several phases of a synthetic job. The quantum length in this case is set to 1, and is scaled in the figure to restore the original parallelism of the job. The convergence rate v of A-CONTROL is set to 0.2. The responsiveness parameter ρ of A-GREEDY is set to 2, and its utilization parameter δ is set to 0.8. As we can see, A-CONTROL satisfies good transient and steady-state performances in each phase of the job while A-GREEDY suffers from apparent desire oscillation, nonzero steady-state error, and overshoot. Other choices of the parameter values will result in similar behaviors of the two algorithms. For example, varying the value of v will only change the convergence rate of A-CONTROL without affecting its other control-theoretic properties, such as stability. For A-GREEDY, different combinations of ρ and δ, such as reducing ρ and increasing δ, will alleviate its desire oscillation but at the cost of slower convergence. In either case, the desire instability of A-GREEDY cannot be completely eliminated, and from practical point of view, this problem may cause potential oscillation in its processor allocations, especially when the load of the system is small and hence the desires tend to be satisfied. Such unnecessary processor reallocations will introduce extra overheads due to the context switching of some processors from one job to another as well as other associated issues, such as loss of localities, etc. In fact, the processor desire of A-GREEDY does satisfy BIBO stability; its oscillation (referred to as *limit cycle* [28] in control theory) is a direct result of the nonlinear behavior of the A-GREEDY algorithm.

6 Algorithmic Analysis

We now analyze the algorithmic performances of ACDEQ by bounding its response time and processor waste for any individual job as well as makespan and total response time for a set of jobs. We then compare ACDEQ with AGDEQ in terms of these performance metrics.

6.1 Preliminaries

For a job J_i scheduled by ACDEQ, given that its work completed in a quantum q is $w_i(q)$ and its span reduced in the quantum is $l_i(q)$, we define the *quantum work efficiency* to be $\alpha_i(q) = \frac{w_i(q)}{a_i(q)sL}$, and the *quantum span efficiency* to be $\beta_i(q) = \frac{l_i(q)}{sL}$, where $a_i(q)$ is the processor allocation to the job in quantum q, s is the processor speed, and L is the quantum length. Obviously, we have $0 \leq \alpha_i(q), \beta_i(q) \leq 1$. The quantum average parallelism $A_i(q)$ is therefore

$$A_i(q) = \frac{w_i(q)}{l_i(q)} = a_i(q)\frac{\alpha_i(q)}{\beta_i(q)}. \tag{4}$$

Let t_q denote the time quantum q starts; then quantum q can be split into two portions depending on whether the processor allocation $a_i(q)$ is more than the parallelism $h_i^{k_t}$ of the job at any time $t \in [t_q, t_q + L]$. Specifically, let $L_1 = \int_{t_q}^{t_q+L}[a_i(q) \leq h_i^{k_t}]$ and $L_2 = \int_{t_q}^{t_q+L}[a_i(q) > h_i^{k_t}]$, where $[x]$ is 1 if proposition x is true and 0 otherwise. Hence, the quantum work $w_i(q)$ is at least $a_i(q)sL_1$ and the quantum span $l_i(q)$ is at least sL_2. Thus, we have $L = L_1 + L_2 \leq \frac{w_i(q)}{a_i(q)s} + \frac{l_i(q)}{s} = (\frac{\alpha_i(q)}{\beta_i(q)} + 1)\frac{l_i(q)}{s}$. Substituting into $l_i(q) = \beta_i(q)sL$, we get

$$\alpha_i(q) + \beta_i(q) \geq 1, \tag{5}$$

which is a lower bound on the sum of quantum work efficiency and quantum span efficiency. We will use this relationship as well as (4) later in our analysis to bound response time and processor waste of a job.

We now define the concept of *transition factor* for job J_i. The transition factor, denoted by C_i, where $C_i \geq 1$, is the maximal ratio on the average parallelism of the job in any two consecutive quanta. Formally, the average quantum parallelism of the job satisfies

$$\frac{1}{C_i} \leq \frac{A_i(q)}{A_i(q-1)} \leq C_i, \tag{6}$$

for all $q \geq q_i$, and $A_i(q_i - 1)$ is defined to be 1, where q_i is the first quantum job J_i is scheduled. The transition factor, like the work and span, can be considered as an intrinsic job characteristic, and is independent of the task scheduler.[2] Since two malleable jobs with the same work and span can have very different parallelism variations, the transition factor indicates how fast the job's parallelism changes with time and thus suggests the level of difficulty to adaptively schedule it in a nonclairvoyant fashion. In contrast to [1], which does not consider the parallelism transitions of a job, we argue that the incorporation of transition factor better reflects the performance of a scheduling algorithm.

In particular, given the transition factor of a job, we can show that the processor desire generated by A-CONTROL in any quantum is bounded from both above and below in terms of the average parallelism of the job in the same quantum.

Lemma 2. *Suppose that A-CONTROL schedules a job J_i with transition factor C_i. Then the processor desire $d_i(q)$ for each quantum q satisfies*

$$\frac{1-v}{C_i - v}A_i(q) \leq d_i(q) \leq \frac{C_i(1-v)}{1-C_iv}A_i(q), \tag{7}$$

where $A_i(q)$ is the job's average parallelism in quantum q and v denotes the convergence rate of A-CONTROL. The inequality on the right only holds when $v < 1/C_i$.

Proof. We will prove the upper bound of $d_i(q)$ by induction on the scheduling quantum. The lower bound can be proven similarly, and is omitted.

Base case: For $q = q_i$, we have $d(q_i) = 1$. Because $A_i(q_i - 1) = 1$ by definition, according to (6), the quantum average parallelism satisfies $A_i(q_i) \geq 1/C_i$. Thus, we have $\frac{d_i(q_i)}{A_i(q_i)} \leq C_i \leq \frac{C_i - C_iv}{1-C_iv}$, since $C_i \geq 1$ and $C_iv < 1$. Therefore, we get

$$d_i(q_i) \leq \frac{C_i(1-v)}{1-C_iv}A_i(q_i).$$

Induction: For $q \geq q_i + 1$, suppose that we have $d_i(q-1) \leq \frac{C_i(1-v)}{1-C_iv}A_i(q-1)$. Because $A_i(q-1) \leq C_iA_i(q)$ from (6), then according to (3), we have for quantum q,

$$\begin{aligned} d_i(q) &= vd_i(q-1) + (1-v)A_i(q-1) \\ &\leq \frac{C_iv(1-v)}{1-C_iv}A_i(q-1) + (1-v)A_i(q-1) \\ &= \frac{1-v}{1-C_iv}A_i(q-1) \leq \frac{C_i(1-v)}{1-C_iv}A_i(q). \end{aligned}$$

□

Remarks. The assumption $v < 1/C_i$ is required for the upper bound of the processor desire to hold. Without this assumption, however, the ratio of processor desire and quantum average parallelism cannot be bounded. The worst case happens when the parallelism of the job reduces much faster than the responsiveness of A-CONTROL with the chosen convergence rate, resulting in the processor desire not reduced as quickly.

6.2 Response Time

We analyze the response time of an individual job scheduled by ACDEQ on arbitrary speed processors. Recall that the response time is the duration between the job's release and completion. Agrawal et al. [1] studied the response time of a job scheduled by the task scheduler A-GREEDY using *trim analysis*, which is a technique to limit the power of the OS allocator assuming that it can behave like an adversary to the task scheduler. However, most practical OS allocators such as DEQ do work cooperatively with the task schedulers. Hence, in the following theorem, we show that the response time of any individual job scheduled by ACDEQ can be bounded in terms of the job's deserved equal share of processors and their speed.

[2] The transition factor, however, does depend on the quantum length and the processor speed, whose variation may yield different transition factors for the same job. For a given quantum length, processor speed, as well as the parallelism profile of a job, the transition factor can usually be derived based on the worst-case schedule. We will not be concerned about how the transition factor may be derived, much like the work and span of a job. We will just make use of these job characteristics to quantify the behavior of our scheduler in terms of performance bounds.

Theorem 3. *Suppose that ACDEQ schedules a set $\mathcal{J}$ of jobs on P processors of speed s. Then, the response time $R_s(J_i)$ for any individual job $J_i \in \mathcal{J}$ with work w_i, span l_i, and transition factor C_i is bounded by*

$$R_s(J_i) \le \frac{2w_i}{s\overline{P}} + \frac{C_i + 1 - 2v}{s(1-v)} l_i, \tag{8}$$

where $\overline{P} = P/|\mathcal{J}|$ is the equal share of processors for each job and v is the convergence rate of A-CONTROL.

Proof. Let AC and DE denote the set of accounted quanta and the set of deductible quanta for job J_i, respectively. To bound the response time $R_s(J_i)$ of job J_i, we will bound separately its total accounted time and total deductible time, denoted by $R_s^{AC}(J_i)$ and $R_s^{DE}(J_i)$.

The total accounted time can be bounded as follows: since an accounted quantum q for job J_i is underallocated by definition, we have $a_i(q) \le A_i(q)$. According to (4) and (5), we have $\alpha_i(q) \ge 1/2$ and therefore $w_i(q) \ge a_i(q)sL/2$. Since an accounted quantum is also deprived, based on DEQ policy, we have $a_i(q) \ge P/|\mathcal{J}(t)| \ge \overline{P}$. The total work done on accounted time thus satisfies $w_i \ge \sum_{q \in AC} w_i(q) \ge \sum_{q \in AC} a_i(q)sL/2 \ge \overline{P}sR_s^{AC}(J_i)/2$. The total accounted time is then $R_s^{AC}(J_i) \le \frac{2w_i}{s\overline{P}}$.

To bound the total deductible time, we observe from definition of deductible quantum and Lemma 2 that $a_i(q) \ge \frac{1-v}{C_i - v} A_i(q)$ for job J_i in deductible quantum q. Substituting (4) and (5) into it, we obtain $\beta_i(q) \ge \frac{1-v}{C_i+1-2v}$ and therefore $l_i(q) \ge \frac{1-v}{C_i+1-2v} sL$. The sum of the quantum span in deductible quanta thus satisfies $l_i \ge \sum_{q \in DE} l_i(q) \ge \sum_{q \in DE} \frac{1-v}{C_i+1-2v} sL = \frac{1-v}{C_i+1-2v} sR_s^{DE}(J_i)$. The total deductible time is $R_s^{DE}(J_i) \le \frac{C_i+1-2v}{s(1-v)} l_i$. □

Remarks. The trim analysis used by Agrawal et al. [1] assumes an adversarial OS allocator, which can provide a large number of processors to a job when the job has low parallelism, and vice versa, thus preventing any task scheduler from achieving good speedup. By trimming off the available processors on deductible quanta effectively allows a task scheduler to achieves nearly linear speedup on accounted quanta. Adopting trim analysis, we can show a similar response time for A-CONTROL that works with any OS allocator. Readers can refer to [1] for more details on this analysis technique.

6.3 Processor Waste

As pointed out in remarks of Lemma 2, bounding processor waste relies on the convergence rate of A-CONTROL to satisfy $v < 1/C_i$ for a job J_i with transition factor C_i. Since the job characteristics are usually unknown prior to its execution, we assume that the convergence rate is chosen based on some historical workload characterization, which ensures that it satisfies the requirement. The processor waste can then be bounded as stated in the following theorem:

Theorem 4. *Suppose that ACDEQ schedules a set $\mathcal{J}$ of jobs on processors of speed s. Then, the processor waste $X_s(J_i)$ for any individual job $J_i \in \mathcal{J}$ with work w_i and transition factor C_i is bounded by*

$$X_s(J_i) \le \frac{C_i(1-v)}{1 - C_i v} w_i, \tag{9}$$

where $v < 1/C_i$ is the convergence rate of A-CONTROL.

Proof. Since ACDEQ never allocates more processors than job J_i desires, that is, $a_i(q) \le d_i(q)$ for any quantum q, from Lemma 2, we have $a_i(q) \le \frac{C_i(1-v)}{1-C_i v} A_i(q)$. Substituting (4) and (5) into it, we obtain $\alpha_i(q) \ge \frac{1-C_i v}{1+C_i-2C_i v}$, and therefore $w_i(q) \ge \frac{1-C_i v}{1+C_i-2C_i v} a_i(q)sL$. Let $X_i(q)$ denote the processor waste of job J_i in quantum q; then we have

$$\begin{aligned} X_i(q) &= a_i(q)sL - w_i(q) \le a_i(q)sL - \frac{1 - C_i v}{1 + C_i - 2C_i v} a_i(q)sL \\ &= \frac{C_i(1-v)}{1 + C_i - 2C_i v} a_i(q)sL \le \frac{C_i(1-v)}{1 - C_i v} w_i(q). \end{aligned}$$

The total processor waste $X_s(J_i)$ is then bounded by $X_s(J_i) = \sum_q X_i(q) \le \sum_q \frac{C_i(1-v)}{1-C_i v} w_i(q) \le \frac{C_i(1-v)}{1-C_i v} w_i$. □

Remarks. Let $X_s^{AC}(J_i)$ denote the processor waste of job J_i on the set of accounted quanta, and we can obtain a separate bound for $X_s^{AC}(J_i)$. Since an accounted quantum q for job J_i is underallocated by definition, we have $a_i(q) \le A_i(q)$. Following the proof of Theorem 4, we can get $X_s^{AC}(J_i) \le w_i$, which gives a tighter waste bound for the job on accounted time. We will make use of this bound later in the proof of Theorem 6.

6.4 Makespan

We analyze the makespan for a set of jobs scheduled by ACDEQ. Recall that makespan is the completion time of the last completed job in the job set, and together with the total response time, they provide a common set of performance indications for an adaptive scheduler in multiprogrammed environment. For ACDEQ, we show the makespan using *competitive analysis* [7], which compares its performance with that of the optimal on processors of the same speed. Specifically, we show that the makespan depends on both the response time and the processor waste for an individual job, and can be obtained by combining both bounds. The following theorem gives the makespan performance:

Theorem 5. *Suppose that ACDEQ schedules a set $\mathcal{J}$ of jobs on processors of speed s. Then the makespan $M_s(\mathcal{J})$ of the job set is bounded by*

$$M_s(\mathcal{J}) \le \left(\frac{C + 1 - 2Cv}{1 - Cv} + \frac{C + 1 - 2v}{1 - v} \right) M_s^*(\mathcal{J}), \tag{10}$$

where $M_s^*(\mathcal{J})$ is the makespan of the optimal scheduler on processors of speed s, C is the maximum transition factor of all jobs in the job set, and $v < 1/C$ is the convergence rate of A-CONTROL.

Proof. Suppose that J_k is the last completed job in $\mathcal{J}$. Then, the makespan $M_s(\mathcal{J})$ of job set $\mathcal{J}$ is given by the completion time c_k of job J_k. Let $R_s^{SA}(J_k)$ denote the total satisfied time for J_k, and let $R_s^{DP}(J_k)$ denote the total deprived time for J_k. Then, the completion time of the job is given by $c_k = r_k + R_s^{SA}(J_k) + R_s^{DP}(J_k)$.

We bound the total satisfied time and the total deprived time for job J_k, separately. Since a satisfied job is also deductible by definition, from Theorem 3, the total satisfied time of J_k is bounded by $R_s^{SA}(J_k) \le \frac{C+1-2v}{s(1-v)} l_k$, where C is the maximum transition factor from all jobs in $\mathcal{J}$. Since OS allocator DEQ allocates all P processors when job J_k is deprived, the amount of processing power

$R_s^{DP}(J_k) \cdot sP$ is either spent on work or wasted. According to Theorem 4, we have

$$\begin{aligned} R_s^{DP}(J_k) \cdot sP &\leq \sum_{J_i \in \mathcal{J}} (w_i + X_s(J_i)) \\ &\leq \sum_{J_i \in \mathcal{J}} \left(w_i + \frac{C_i(1-v)}{1-C_i v} w_i \right) \\ &= \sum_{J_i \in \mathcal{J}} \frac{C_i + 1 - 2C_i v}{1 - C_i v} w_i \\ &\leq \frac{C + 1 - 2Cv}{1 - Cv} \sum_{J_i \in \mathcal{J}} w_i. \end{aligned}$$

The total deprived time of J_k is bounded by

$$R_s^{DP}(J_k) \leq \left(\frac{C+1-2Cv}{1-Cv} \right) \frac{\sum_{J_i \in \mathcal{J}} w_i}{sP}.$$

The makespan $M_s(\mathcal{J})$ of job set $\mathcal{J}$, which is equal to the completion time c_k of J_k, is then

$$\begin{aligned} M_s(\mathcal{J}) \leq &\left(\frac{C+1-2Cv}{1-Cv} \right) \frac{\sum_{J_i \in \mathcal{J}} w_i}{sP} \\ &+ \left(\frac{C+1-2v}{1-v} \right) \max_{J_i \in \mathcal{J}} \left(\frac{l_i}{s} + r_i \right). \end{aligned}$$

This directly implies Theorem 5 as both $\frac{\sum_{J_i \in \mathcal{J}} w_i}{sP}$ and $\max_{J_i \in \mathcal{J}} (\frac{l_i}{s} + r_i)$ are lower bounds on the makespan of the job set [8], [24] with speed s processors. □

6.5 Total Response Time

It is known that any deterministic nonclairvoyant scheduler is $\Omega(n^{1/3})$-competitive in terms of the total response time even for sequential jobs on a single processor [38], where n is the total number of jobs in the job set. Hence, it is unlikely that ACDEQ will achieve good competitive result if it shares the same amount of resources with the optimal scheduler. In this case, we apply *resource augmentation analysis* [30], [41], which equips the online algorithm with extraspeed processors compared to the optimal. We show that ACDEQ is $O(C/\epsilon)$-competitive in terms of the total response time when equipped with $O(1)$ times, or specifically $(4+\epsilon)$ times faster processors for any $\epsilon > 0$. The proof is included in the appendix.

Theorem 6. *Suppose that ACDEQ schedules a set $\mathcal{J}$ of jobs on processors of speed s, where $s = 4 + \epsilon$ for any $\epsilon > 0$. Then the total response time $R_s(\mathcal{J})$ of the job set is bounded by*

$$R_s(\mathcal{J}) \leq \left(2 + \frac{10 + 2C - 12v}{\epsilon(1-v)} \right) R_1^*(\mathcal{J}), \tag{11}$$

where $R_1^*(\mathcal{J})$ is the total response time of the optimal scheduler on unit-speed processors, C is the maximum transition factor of all jobs in the job set, and v is the convergence rate of A-CONTROL.

Remarks. Without resource augmentation, He et al. [24] proved the total response time of two-level scheduler AGDEQ under a special case, where all jobs are batch released at time 0. In [47], we have shown a similar result for ACDEQ, which is $O(C)$-competitive in terms of the total response time for any batched job set.

6.6 Comparison with AGDEQ

Now, we compare ACDEQ with AGDEQ in terms of their performance bounds for both an individual job and a job set. In addition, we briefly comment on the performances of both algorithms in practice.

For AGDEQ, the performance bounds are derived similarly as those of ACDEQ. Specifically, given a responsiveness parameter ρ and a utilization parameter δ of A-GREEDY, the performances of AGDEQ on P processors of speed s are bounded by the following [1], [24], [46]:

$$\begin{aligned} R_s(J_i) &\leq \frac{w_i}{\delta s \overline{P}} + \frac{2}{s(1-\delta)} l_i + o(1), \\ X_s(J_i) &\leq \frac{1+\rho-\delta}{\delta} w_i, \\ M_s(\mathcal{J}) &\leq \left(\frac{\rho+1}{\delta} + \frac{2}{1-\delta} \right) M_s^*(\mathcal{J}) + o(1), \\ R_s(\mathcal{J}) &\leq \left(2 + \frac{4}{\delta(1-\delta)\epsilon} \right) R_1^*(\mathcal{J}) + o(1), \end{aligned}$$

where $\overline{P} = P/|\mathcal{J}|$, and the total response time bound holds when $s = 2/\delta + \epsilon$ for any $\epsilon > 0$. In a direct comparison, ACDEQ tends to have better bounds when the jobs have small transition factors; otherwise, the bounds of AGDEQ become better with appropriately chosen parameters. Note that AGDEQ is oblivious of the transition factor in the analysis because the symmetric structure of its multiplicative-increase multiplicative-decrease strategy allows it to bypass this difficulty.

It should be noted that although both algorithms attempt to exploit the parallelism correlation of the jobs, their theoretical performance bounds are established pessimistically based on the scenario where the future parallelism of a job is not correlated to its past. (In the case of ACDEQ, a job's parallelism is only correlated by its maximum transition.) Their performances in practice should therefore be much better than predicted by these theoretical bounds, especially when the parallelism of the jobs does exhibit strong correlation. In the next section, we will further evaluate and compare the performances of ACDEQ and AGDEQ through simulations.

7 Simulations

We conduct simulations to study the practical performance of ACDEQ and compare it with AGDEQ using malleable parallel jobs. To better understand these two-level adaptive schedulers, we first study the transient responses of task schedulers A-CONTROL and A-GREEDY on a set of parallelism variations in terms of the processor desire calculations, which shed light on their performances in practice. We then augment Downey's job model [15] with these different parallelism variations to comprehensively evaluate the two adaptive schedulers in terms of both individual job performances and job set performances.

7.1 Malleable Parallel Jobs

To verify the effectiveness of two-level adaptive schedulers, it will be helpful to test them on malleable parallel jobs with changing parallelism characteristics. Unfortunately, there is little existing work in the literature that models such

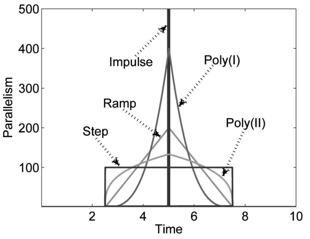


Fig. 3. Five different parallelism variation curves specified by Step, Impulse, Ramp, Poly(I), and Poly(II) profiles.

workloads. Hence, in this paper, we construct malleable jobs based on a traditional workload model, namely Downey's model [15] and augment it with a set of internal parallelism variations.

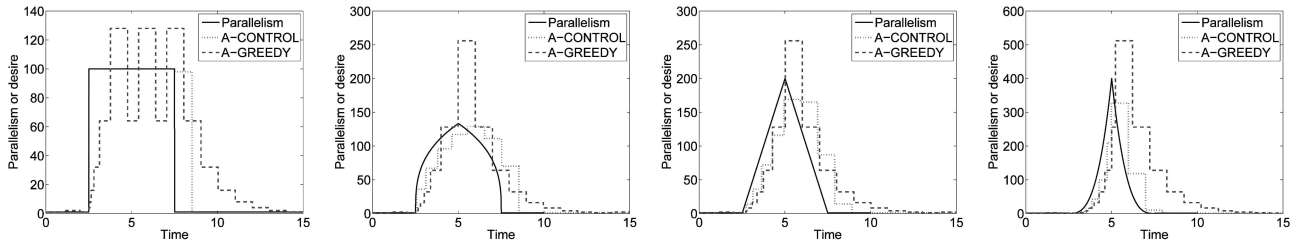
In particular, Downey's workload model, which generates nonmalleable parallel jobs, provides external information about the jobs such as their work requirements, arrival patterns, average parallelism, etc. Internally, we divide a parallel job into a series of segments and each segment is identified by a specific parallelism structure. Instead of using completely random parallelism structures, which does not allow clear account of the feedback-allocation process of a two-level adaptive scheduler, we identify five generic forms of parallelism variations, which are specified by *Step*, *Impulse*, *Ramp*, *Poly(I)*, and *Poly(II)* profiles, respectively, as shown in Fig. 3 and they describe precisely how the parallelism of a segment evolves with time. These profiles provide a comprehensive coverage of the changing parallelism dynamics and reflect a wide range of parallel programming patterns. For instance, the Step profile can describe constant and stable parallelism over a period of time from typical data-parallel sections of a job. The Impulse profile on the other hand can emulate a drastic one-off increase in parallelism typically encountered in, e.g., a short Do-Parallel loop. The Ramp and the two Poly profiles, which are constructed by polynomials of degree 1, 3, and 1/3 in our simulation, can model changes in the job's parallelism with different rates for spawning and joining parallel threads.

To maintain consistency with the original nonmalleable jobs, we ensure that the work and the average parallelism of a job adhere to those initially generated by Downey's model. In particular, the average parallelism of each segment is chosen uniformly according to that of the original model, and no more segment is added after the aggregate work reaches that initially generated. Also, for each segment, we construct the Step profile first, which is used as a blueprint to derive other profiles in case they have been selected. This ensures that all profiles are coherent with each other, as shown in Fig. 3, where the five different parallelism profiles have the same work and span, hence the same average parallelism. In our simulations in Sections 7.2 and 7.3, we only generate homogeneous jobs with a single type of profile interconnected by sequential phases, which is aimed at studying the impact of different parallelism variations on the task schedulers. In Section 7.4, we mix different profiles to create heterogeneous jobs and evaluate the performances of the two-level schedulers under this more diverse set of workloads.

7.2 Transient Response

To better understand the behavior of two-level schedulers, we first focus on task schedulers A-CONTROL and A-GREEDY by studying their transient responses to different parallelism variations, which can provide valuable insights on their performances in practice.

Fig. 4 shows the transient responses of A-CONTROL and A-GREEDY on four parallelism profiles. (The transient response of Impulse profile is similar to that of Step and hence is not shown.) In the figure, each profile has the same work, span, and average parallelism. The quantum length is set to 1/5 of the segment length, which is scaled to restore the original parallelism variation. In addition, sequential phases are added before and after each profile such that the processor desires of both schedulers start and end at a steady state with a value of 1, and are always satisfied by the OS allocator. The convergence rate of A-CONTROL is set to $v = 0$ in this case for faster response while the responsiveness parameter and the utilization parameter of A-GREEDY are set to $\rho = 2$ and $\delta = 0.8$, respectively. As can be seen, for Step profile, A-GREEDY is able to gradually catch up with the parallelism change but suffers from desire instability when the parallelism remains constant. In contrast, A-CONTROL rapidly approaches the parallelism within a quantum, and thereafter provides stable desires by directly utilizing the average parallelism of the job. For the other profiles, both A-CONTROL and A-GREEDY are able to respond gradually to the parallelism variations with A-CONTROL in general following more closely the changes of the parallelism and thus taking less processor reallocations. This is due to A-CONTROL's more effective processor desire calculation, which suggests that it probably performs better than A-GREEDY in practice. We will verify the claim in the following section through more comprehensive simulations.



(a) (b) (c) (d)

Fig. 4. Transient responses of A-CONTROL and A-GREEDY on (a) Step, (b) Poly(II), (c) Ramp, and (d) Poly(I) profiles.

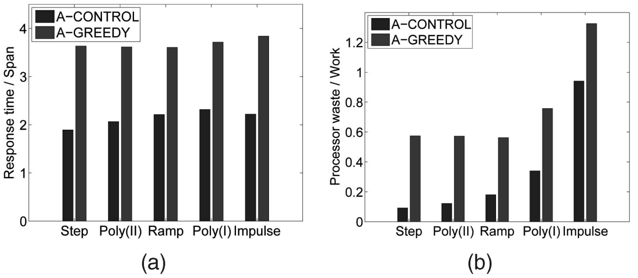


Fig. 5. Individual job performances of A-CONTROL and A-GREEDY on the five parallelism profiles in terms of (a) response time and (b) processor waste.

7.3 Individual Job Performances

To verify the quality of feedbacks observed in the transient responses, we design one set of simulations to measure the individual job performances, that is, the response time and the processor waste of A-CONTROL and A-GREEDY. In the simulation, only one job is run each time and the processor desires of both schedulers are always granted. This allows us to evaluate the performances of the task schedulers under a favorable circumstance; otherwise, the advantage of a more efficient processor desire calculation scheme cannot be reflected. Fig. 5 shows the effects of the five parallelism variations on the response time and the processor waste of both task schedulers. The response time is normalized in terms of the span of the job and the processor waste is normalized in terms of the job's total work.

From the figure, we can see clearly that A-CONTROL indeed outperforms A-GREEDY with respect to the response time and the processor waste for all parallelism variations. Moreover, smoother parallelism variations with smaller transitions also lead to better performances of A-CONTROL, matching the analysis in Section 6. For instance, A-CONTROL has better performance for the Step profile, which exhibits smaller parallelism transition (see Fig. 3). In contrast, the Ramp and Poly(I) profiles with steeper parallelism variations result in larger response time and processor waste for A-CONTROL. We should point out that although the Impulse profile has drastic parallelism variation, it occurs less frequently than the other profiles. Hence, A-CONTROL can easily capture its parallelism variation within one quantum under the unconstrained environment, thus has better response time. The performances of A-GREEDY, on the other hand, are relatively less sensitive to the different parallelism profiles, but are generally inferior to those of A-CONTROL even on jobs with larger parallelism transitions. In summary, the simulation results confirm our previous insight that a task scheduler with better transient responses should be able to achieve superior performances in terms of the response time and processor waste of an individual job.

7.4 Job Set Performances

We now evaluate the performances of ACDEQ and compare it with AGDEQ in terms of the makespan and total response time for a set of jobs. We set the system load proportionally to the number of jobs as well as their arrival rate based on Downey's model [15] for evaluating the total response time. For makespan, we use batched job set and define the load to be proportional to the number of jobs (more precisely, the load is defined to be the number of jobs divided by 100 in this case), since, otherwise, the makespan could be dominated by the release time of the last job from a large job stream. To simulate the performances of an adaptive scheduler in practice, we also capture the number of processor reallocations it incurs during running time, which are used to measure the overheads. The response time of a job is then assumed to increase by an additive factor $\gamma \cdot \chi$, where χ denotes its total number of processor reallocations under a particular scheduler and γ depends on the system's physical overhead for context switching.

Fig. 6 shows the number of processor reallocations, the makespan ratio, and the total response time ratio of ACDEQ and AGDEQ with different system loads. We can see from Fig. 6a that under light loads both schedulers produce a large number of processor reallocations on the last completed job, which contributes to the makespan of the job set. With increased loads, however, the processor desires of both schedulers cannot be easily granted, and this eventually leads to less number of reallocations. On the other hand, the total number of reallocations of all jobs is shown in Fig. 6c, and it is related to the total response time of the job set. As can be seen, the total number of reallocations first increases because of more jobs joining the system before it starts to decrease, which is also due to the depression of the processor desires under higher system loads. Moreover, the number of processor reallocations of ACDEQ is always less than that of AGDEQ under all system loads because of A-CONTROL's more effective processor desire calculation. Furthermore, we can see from Fig. 6b and

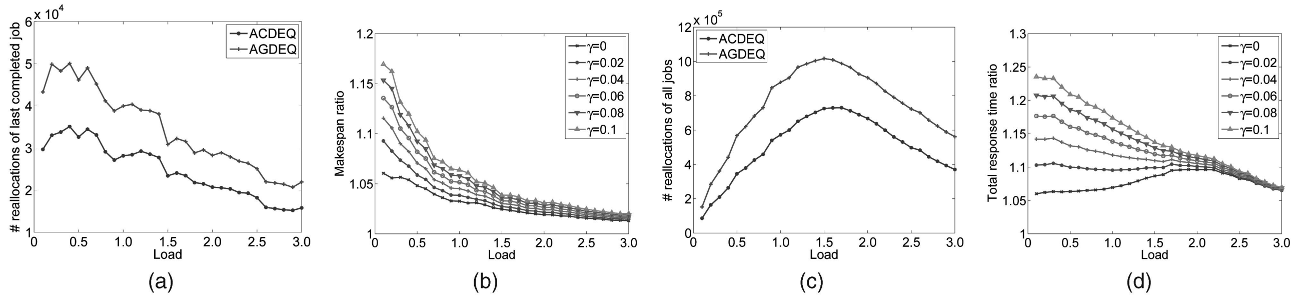


Fig. 6. Number of processor reallocations of AGDEQ and ACDEQ on (a) the last completed job and (c) all jobs in the job set. The performance ratio of AGDEQ over ACDEQ with different cost for reallocation overhead in terms of (b) the makespan and (d) total response time.

Fig. 6d that ACDEQ performs better than AGDEQ under light to medium loads in terms of both makespan and total response time, especially when the cost of processor reallocations becomes high. This again demonstrates that A-CONTROL is able to provide more effective processor desires. Under heavy system loads, however, the processor desires tend to be deprived and the advantage of A-CONTROL diminishes since neither schedulers have direct control over the processor allocations. Therefore, the performances of both schedulers are comparably similar in such a case.

8 Related Work

We review related work on adaptive scheduling, makespan and total response time minimization, and parallel workload modeling. For adaptive scheduling, many different strategies have been proposed under various perspectives ranging from algorithmic to control-theoretic to empirical.

From algorithmic perspective, Agrawal et al. [1], [3] studied adaptive task scheduling and proposed two algorithms, namely A-GREEDY and A-STEAL, based on centralized scheduling and distributed work stealing, respectively. Both schedulers employ a multiplicative-increase multiplicative-decrease strategy, and are shown to be efficient in terms of response time and processor waste for an individual job. He et al. [24] combined A-GREEDY and A-STEAL with DEQ [37], [48] and proved that the resulting two-level schedulers AGDEQ and ASDEQ are $O(1)$-competitive in terms of makespan, and when all jobs are released in a batch, $O(1)$-competitive in terms of the total response time. He et al. [25] also showed that in a heavily loaded system, the two-level schedulers can be coupled with a Round Robin (RR) strategy to achieve similar results. Sun et al. [46] later proved that when jobs have arbitrary release time, AGDEQ also achieves $O(1)$-competitiveness in terms of the total response time with $O(1)$ times faster processors than the optimal.

Adaptive scheduling has also been studied from control-theoretic perspective. Related work in this area tends to focus on transient and steady-state performances in terms of control-theoretic properties. Lu et al. [34], [35] presented a feedback control scheduling framework for adaptive real-time systems, and developed Feedback-Control Earliest-Deadline-First (FC-EDF) scheduler to control real-time CPU utilizations as well as an integral controller to control delays in web servers. Goel et al. [22] designed an adaptive controller that schedules real-rate applications by estimating the application's progress with time-stamps in a feedback loop. Using adaptive control, Padala et al. [40] developed a resource management system to optimize the resource utilization and at the same time to meet specific QoS goals for multitier applications. Similar approaches are also used in [33], [49] for dynamically adjusting the resource partitioning for enterprise servers.

Many empirical studies on adaptive scheduling are also known in the literature. Sen [45] presented experimental results on a dynamic desire estimation algorithm for the Cilk work-stealing scheduler [6]. Agrawal et al. [2] compared ASDEQ with a task scheduler ABP [4], which is also based on work-stealing but without parallelism feedback and hence coupled with EQUI. They showed empirically that the feedback-based scheduler ASDEQ indeed has superior performance than ABP. He et al. [25] evaluated the performance of AGDEQ under fork/join jobs, and revealed that it actually performs much better in practice than predicted by the theoretical bounds. In addition, Nguyen et al. [39], Weissman et al. [50], Corbalán et al. [12], and Sundarson et al. [43] have also implemented various adaptive scheduling strategies on different platforms based on measurements of certain job characteristics such as speedup, efficiency, execution time, etc. All of them reported success in improving the system performances with adaptive scheduling.

We now review other related work for scheduling a set of n parallel jobs on P processors with makespan and total response time as the performance metrics. Motwani et al. [38] showed that RR is $(2 - \frac{2}{n+1})$-competitive with respect to the total response time for fully parallelizable jobs that are batch released. When jobs can have arbitrary release time, however, they showed that every deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$-competitive. Kalyanasundaram and Pruhs [30] introduced resource augmentation analysis, and showed that the nonclairvoyant algorithm Shortest Elapsed Time First (SETF) is $(1+\epsilon)$-speed $(1+1/\epsilon)$-competitive in terms of the total response time. In addition, it is well known that, for fully parallelizable jobs, any scheduler that does not waste processors is optimal for makespan and the clairvoyant scheduler Shortest Remaining Processing Time (SRPT) is optimal for total response time [9].

For parallel jobs with changing degrees of parallelism, Edmonds [16] showed that EQUI is $(2+\epsilon)$-speed $O(1)$-competitive with respect to the total response time. Edmonds and Pruhs [18] showed that Latest Arrival Processor Sharing (LAPS), which in a sense combines EQUI and SETF, is $(1+\epsilon)$-speed $O(1)$-competitive for sufficiently large ϵ. When jobs are released in a batched fashion, Edmonds et al. [17] showed that EQUI is $(2+\sqrt{3})$-competitive for the total response time. Robert and Schabanel [44] showed that EQUI is $\Theta(\ln n / \ln \ln n)$-competitive for makespan. Deng et al. [14] proved that DEQ with jobs' instantaneous parallelism as feedback is 2-competitive in terms of the total response time for parallel jobs with a single phase and 4-competitive for multiphased jobs. The latter ratio was improved to 3 by He et al. [27]. In addition, Brecht et al. [8] showed that DEQ is $(2 - 1/P)$-competitive with respect to makespan.

Finally, we review some related work on parallel workload modeling. Some existing models such as the ones proposed by Feitelson [19], Jann et al. [29], and Lublin and Feitelson [36] are used to generate rigid jobs, which require a specific number of processors to execute. Others like those proposed by Downey [15] and Cirne and Berman [11] are used to model moldable jobs, which can execute with an arbitrary number of processors at launch time but cannot change allocations afterward. To model malleable jobs with internal structures, a flexible hierarchical model was proposed by Calzarossa et al. [10] and further extended by Feitelson and Rudolph [21]. Unfortunately, they only provided two basic types of internal parallelism structures based on fork-join and workpile models, but without detailed implementation. In the simulations conducted by Agrawal et al. [2] and He et al. [25], the workloads were generated by varying the parameters of fork-join jobs such as work and span according to various distributions. The authors concluded, however, that the simulation results are fairly insensitive to the chosen distributions.

9 Conclusion

Many researchers have applied control theory to design computing and resource management systems, and these control-inspired algorithms often demonstrate robust behaviors. However, as far as we know, there is little work in the literature that incorporates performance metrics from both control-theoretic and algorithmic perspectives. In this paper, we have presented ACDEQ, a control-based algorithm for scheduling malleable jobs on multiprocessors, and have analyzed its performance from both control-theoretic and algorithmic perspectives. We have also compared ACDEQ with an existing scheduler AGDEQ using synthetic malleable jobs with a set of different parallelism variations. The simulation results have confirmed ACDEQ's superior performances from both control-theoretic and algorithmic view points.

While both task schedulers A-Control and A-Greedy generate parallelism feedbacks based on the job's execution in a single quantum, it may be more effective to calculate processor desires using more than one historical quantum. This will require the task scheduler to treat the parallel job as a dynamic higher order system, and will be especially applicable to jobs with regular parallelism structure. Other future research in this area may include dynamically adjusting quantum length or other parameters to achieve better systemwide adaptivity, and identifying alternative job characteristics such as the frequency on its change of parallelism for tighter analysis of existing schedulers as well as for the better understanding of adaptive scheduling in general.

Appendix

Proof of Theorem 6

We first define some useful notions. Let $\mathcal{J}(t)$ denote the set of active jobs at time t, and let $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ denote the set of accounted jobs and the set of deductible jobs at time t, respectively. For convenience, let $n_t = |\mathcal{J}(t)|$, let $n_t^A = |\mathcal{J}_A(t)|$ and $n_t^B = |\mathcal{J}_B(t)|$. Hence, we have $n_t = n_t^A + n_t^B$. Throughout the execution of job J_i, let $a_A(J_i)$ denote its total accounted processing power, i.e., $a_A(J_i) = \int_0^\infty a_i(t)s \cdot [J_i(t) \in \mathcal{J}_A(t)]dt$, and let $t_B(J_i)$ denote its total deductible time, i.e., $t_B(J_i) = \int_0^\infty [J_i(t) \in \mathcal{J}_B(t)]dt$, where s denotes the processor speed of ACDEQ and $[x]$ is 1 if proposition x is true and 0 otherwise. From Theorems 3 and 4, the total accounted processing power $a_A(J_i)$ and the total deductible time $t_B(J_i)$ satisfy

$$a_A(J_i) \le 2w_i, \tag{12}$$

$$t_B(J_i) \le \frac{C_i + 1 - 2v}{s(1-v)} l_i. \tag{13}$$

Let $t_B(\mathcal{J}) = \sum_{i=1}^n t_B(J_i)$, and summing (13) over all jobs, we have $t_B(\mathcal{J}) \le \frac{C+1-2v}{s(1-v)} \sum_{i=1}^n l_i$, where n is the total number of jobs in $\mathcal{J}$. We also need to define the notion of *t-prefix* and *t-suffix* to ease analysis. For ACDEQ, define t-prefix $J_i(\overleftarrow{t})$ of job J_i to be the portion of the job executed on and before time t, and t-suffix $J_i(\overrightarrow{t})$ to be the portion executed after time t. In addition, we extend the definitions of t-prefix and t-suffix from a job to a job set such that $\mathcal{J}(\overleftarrow{t}) = \{J_i(\overleftarrow{t}) : J_i \in \mathcal{J} \text{ and } r_i \ge t\}$ and $\mathcal{J}(\overrightarrow{t}) = \{J_i(\overrightarrow{t}) : J_i \in \mathcal{J} \text{ and } r_i \ge t\}$. Similarly, we let $\mathcal{J}^*(\overleftarrow{t})$ and $\mathcal{J}^*(\overrightarrow{t})$ denote the t-prefix and the t-suffix of job set $\mathcal{J}$ executed by the optimal scheduler, respectively.

To prove the total response time of ACDEQ, we use *amortized local competitiveness argument* [18], [42], which bounds the amortized performance of an online algorithm at any time through a potential function. Specifically, we need to find a potential function $\Phi(t)$ such that on processors of speed $s = 4 + \epsilon$ for any $\epsilon > 0$, the execution of the job set satisfies the following:

- *Boundary condition*: $\Phi(0) = 0$ and $\Phi(\infty) \ge 0$;
- *Arrival condition*: $\Phi(t)$ does not increase when new jobs arrive;
- *Completion condition*: $\Phi(t)$ does not increase when jobs complete under either ACDEQ or the optimal;
- *Running condition*:

$$\frac{dR_s(\mathcal{J}(t))}{dt} + \frac{d\Phi(t)}{dt} \le \frac{2s}{\epsilon}\left(\frac{dR_1^*(\mathcal{J}^*(t))}{dt} + \frac{dt_B(\mathcal{J}(t))}{dt}\right), \tag{14}$$

where $\frac{dR_s(\mathcal{J}(t))}{dt} = \lim_{\Delta t \to 0} \frac{R_s(\mathcal{J}(\overleftarrow{t+\Delta t})) - R_s(\mathcal{J}(\overleftarrow{t}))}{\Delta t}$ denotes the rate of change for the job set in terms of the total response time under ACDEQ at time t, and we have $\frac{dR_s(\mathcal{J}(t))}{dt} = n_t$. Similarly, the rate of change in total response time under the optimal satisfies $\frac{dR_1^*(\mathcal{J}^*(t))}{dt} = n_t^*$, and the rate of change in total deductible time satisfies $\frac{dt_B(\mathcal{J}(t))}{dt} = n_t^B$. In addition, $\frac{d\Phi(t)}{dt} = \lim_{\Delta t \to 0} \frac{\Phi(t+\Delta t) - \Phi(t)}{\Delta t}$ denotes the rate of change in potential function at t. Note that we assume Δt is infinitesimally small such that no new job arrives and no job completes, or makes a transition between two phases, or experiences processors reallocation under both ACDEQ and the optimal. Since $\sum_{i=1}^n l_i$ is a lower bound on the total response time of job set $\mathcal{J}$ on unit-speed processors [14], [16], [24], integrating (14) over time, we have

$$\begin{aligned} R_s(\mathcal{J}) &\le \frac{2s}{\epsilon}\left(R_1^*(\mathcal{J}) + t_B(\mathcal{J})\right) \\ &\le \frac{2s}{\epsilon}\left(R_1^*(\mathcal{J}) + \frac{C+1-2v}{s(1-v)}\sum_{i=1}^n l_i\right) \\ &\le \frac{2s}{\epsilon}\left(1 + \frac{C+1-2v}{s(1-v)}\right)R_1^*(\mathcal{J}) \\ &= \left(2 + \frac{10 + 2C - 12v}{\epsilon(1-v)}\right)R_1^*(\mathcal{J}), \end{aligned}$$

which directly implies Theorem 6.

We adopt the potential function used by Lam et al. [32] for online speed scaling and tailor it to suit the total response time analysis of ACDEQ. Specifically, at any time t, let $n_t(z)$ denote the number of jobs whose remaining accounted processing power is at least $2z$ under ACDEQ, i.e., $n_t(z) = \sum_{i=1}^{n}[a_A(J_i(\overrightarrow{t})) \geq 2z]$, and let $n_t^*(z)$ denote the number of jobs whose remaining work is at least z under the optimal, i.e., $n_t^*(z) = \sum_{i=1}^{n}[w(J_i^*(\overrightarrow{t})) \geq z]$. The potential function is defined to be

$$\Phi(t) = \eta \int_0^\infty \left[\left(\sum_{i=1}^{n_t(z)} i \right) - n_t(z) n_t^*(z) \right] dz, \tag{15}$$

where $\eta = \frac{4}{\epsilon P}$. Define $\phi_t(z) = (\sum_{i=1}^{n_t(z)} i) - n_t(z)n_t^*(z)$ for convenience. We now need to check the conditions.

- *Boundary condition*: at time 0, no job exists. The terms $n_t(z)$ and $n_t^*(z)$ are both 0 for all z. Therefore, we have $\Phi(0) = 0$. At time ∞, no job remains, so again we have $\Phi(\infty) = 0$. Hence, the boundary condition holds.
- *Arrival condition*: suppose that a new job with work w' arrives at time t. Let t^- and t^+ denote the instances right before and after the job arrives. Hence, we have $n_{t^+}^*(z) = n_{t^-}^*(z) + 1$ for $z \leq w'$ and $n_{t^+}^*(z) = n_{t^-}^*(z)$ for $z > w'$. Similarly, $n_{t^+}(z) = n_{t^-}(z) + 1$ for $z \leq a'/2$ and $n_{t^+}(z) = n_{t^-}(z)$ for $z > a'/2$, where a' is the total accounted processing power to the job. Note that $a'/2 \leq w'$ from (12). Thus, it is obvious that for $z > w'$, we have $\phi_{t^+}(z) = \phi_{t^-}(z)$. For $z \leq w'$, we consider two cases.

  Case 1: for $z \leq a'/2$, we have

$$\begin{aligned} &\phi_{t^+}(z) - \phi_{t^-}(z) \\ &\quad = \left(\sum_{i=1}^{n_{t^-}(z)+1} i \right) - (n_{t^-}(z)+1)(n_{t^-}^*(z)+1) \\ &\qquad - \left(\sum_{i=1}^{n_{t^-}(z)} i \right) + n_{t^-}(z) n_{t^-}^*(z) = -n_{t^-}^*(z) \leq 0. \end{aligned}$$

  Case 2: for $a'/2 \leq z \leq w'$, we have

$$\begin{aligned} \phi_{t^+}(z) - \phi_{t^-}(z) &= \left(\sum_{i=1}^{n_{t^-}(z)} i \right) - n_{t^-}(z)(n_{t^-}^*(z)+1) \\ &\quad - \left(\sum_{i=1}^{n_{t^-}(z)} i \right) + n_{t^-}(z) n_{t^-}^*(z) \\ &= -n_{t^-}(z) \leq 0. \end{aligned}$$

  Hence, $\Phi(t^+) = \eta \int_0^\infty \phi_{t^+}(z)dz \leq \eta \int_0^\infty \phi_{t^-}(z)dz = \Phi(t^-)$, and the arrival condition holds.
- *Completion condition*: when a job completes under ACDEQ or the optimal, $\Phi(t)$ remains unchanged, because in such cases, $n_t(z)$ or $n_t^*(z)$ do not change for all z. Hence, the completion condition holds.
- *Running condition*: as mentioned in Section 4, DEQ ensures that accounted jobs, which are deprived by definition, get at least P/n_t processors at any time t. In the worst case, the n_t^A accounted jobs at time t have the most remaining accounted processing power, while the optimal executes the job with the least remaining work using all P processors. As a result, the change of potential function $\frac{d\Phi(t)}{dt}$ can be shown to satisfy [46]

$$\begin{aligned} \frac{d\Phi(t)}{dt} &\leq \frac{4}{\epsilon P} \left(-\frac{n_t^A(n_t^A+1)}{2} \cdot \frac{sP}{2n_t} + n_t P + n_t^* \frac{sPn_t^A}{2n_t} \right) \\ &\leq \frac{4}{\epsilon} \left(1 - \frac{x_t^2 s}{4} \right) n_t + \frac{2s}{\epsilon} n_t^* , \end{aligned} \tag{16}$$

  where $x_t = n_t^A / n_t$, and $0 \leq x_t \leq 1$. Since a job is either accounted or deductible, we have $n_t^B = (1 - x_t)n_t$. It can be easily verified that the running condition holds for all values of x_t by substituting (16) into (14).

References

[1] K. Agrawal, Y. He, W.J. Hsu, and C.E. Leiserson, "Adaptive Scheduling with Parallelism Feedback," *Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP),* pp. 100-109, 2006.

[2] K. Agrawal, Y. He, and C.E. Leiserson, "An Empirical Evaluation of Work Stealing with Parallelism Feedback," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 19-29, 2006.

[3] K. Agrawal, Y. He, and C.E. Leiserson, "Adaptive Work Stealing with Parallelism Feedback," *Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP),* pp. 112-120, 2007.

[4] N.S. Arora, R.D. Blumofe, and C.G. Plaxton, "Thread Scheduling for Multiprogrammed Multiprocessors," *Proc. Symp. Parallel Algorithms and Architectures (SPAA),* pp. 119-129, Sept. 1998.

[5] K.J. Åström and B. Wittenmark, *Adaptive Control.* Addison-Wesley, 1989.

[6] R.D. Blumofe and C.E. Leiserson, "Scheduling Multithreaded Computations by Work Stealing," *J. ACM,* vol. 46, no. 5, pp. 720-748, 1999.

[7] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis.* Cambridge Univ. Press, 1998.

[8] T. Brecht, X. Deng, and N. Gu, "Competitive Dynamic Multiprocessor Allocation for Parallel Applications," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 448-455, 1995.

[9] P. Brucker, *Scheduling Algorithms.* Springer-Verlag, 2001.

[10] M. Calzarossa, A.P. Merlo, D. Tessera, G. Haring, and G. Kotsis, "A Hierarchical Approach to Workload Characterization for Parallel Systems," *Proc. Int'l Conf. and Exhibition High-Performance Computing and Networking (HPCN),* pp. 102-109, 1995.

[11] W. Cirne and F. Berman, "A Model for Moldable Supercomputer Jobs," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* p. 59, 2001.

[12] J. Corbálan, X. Martorell, and J. Labarta, "Performance-Driven Processor Allocation," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 7, pp. 599-611, July 2005.

[13] X. Deng and P. Dymond, "On Multiprocessor System Scheduling," *Proc. Symp. Parallel Algorithms and Architectures (SPAA),* pp. 82-88, 1996.

[14] X. Deng, N. Gu, T. Brecht, and K. Lu, "Preemptive Scheduling of Parallel Jobs on Multiprocessors," *Proc. Symp. Discete Algorithms (SODA) ,* pp. 159-167, 1996.

[15] A.B. Downey, "A Parallel Workload Model and Its Implications for Processor Allocation," *Proc. Int'l Symp. High Performance Distributed Computing (HPDC),* p. 112, 1997.

[16] J. Edmonds, "Scheduling in the Dark," *Proc. Symp. Theory of Computing (STOC),* pp. 179-188, 1999.

[17] J. Edmonds, D.D. Chinn, T. Brecht, and X. Deng, "Non-Clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics," *Proc. Symp. Theory of Computing (STOC),* pp. 120-129, 1997.

[18] J. Edmonds and K. Pruhs, "Scalably Scheduling Processes with Arbitrary Speedup Curves," *Proc. Symp. Discrete Algorithms,* pp. 685-692, 2009.

[19] D.G. Feitelson, "Packing Schemes for Gang Scheduling," *Proc. Job Scheduling Strategies for Parallel Processing (JSSPP) Workshop,* pp. 89-110, 1996.

[20] D.G. Feitelson, "Job Scheduling in Multiprogrammed Parallel Systems (Extended Version)," Research Report RC19790(87657), IBM , Second Revision, 1997.

[21] D.G. Feitelson and L. Rudolph, "Metrics and Benchmarking for Parallel Job Scheduling," *Proc. Job Scheduling Strategies for Parallel Processing (JSSPP) Workshop,* pp. 1-24, 1998.

[22] A. Goel, J. Walpole, and M. Shor, "Real-Rate Scheduling," *Proc. Real-Time Application Symp. (RTAS),* pp. 434-441, 2004.

[23] R.L. Graham, "Bounds on Multiprocessing Anomalies," *SIAM J. Applied Math.,* vol. 17, no. 2, pp. 416-429, 1969.

[24] Y. He, W.J. Hsu, and C.E. Leiserson, "Provably Efficient Two-Level Adaptive Scheduling," *Proc. Job Scheduling Strategies for Parallel Processing (JSSPP) Symp.,* pp. 1-32, 2006.

[25] Y. He, W.J. Hsu, and C.E. Leiserson, "Provably Efficient Online Non-Clairvoyant Adaptive Scheduling," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 1-10, 2007.

[26] Y. He, H. Sun, and W.J. Hsu, "Improved Results for Scheduling Batched Parallel Jobs by Using a Generalized Analysis Framework," *J. Parallel and Distributed Computing,* vol. 70, 2009. pp. 173-182.

[27] Y. He, H. Sun, and W.J. Hsu, "Adaptive Scheduling of Parallel Jobs on Functionally Heterogeneous Resources," *Proc. Int'l Conf. Parallel Processing (ICPP),* p. 43, 2007.

[28] J.L. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems.* Wiley-Interscience, 2004.

[29] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riordan, "Modeling of Workload in MPPs," *Proc. Job Scheduling Strategies for Parallel Processing (JSSPP) Workshop,* pp. 95-116, 1997.

[30] B. Kalyanasundaram and K. Pruhs, "Speed is as Powerful as Clairvoyance," *Foundations of Computer Science,* pp. 214-221, Springer 1995.

[31] M. Kumar, "Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications," *IEEE Trans. Computers,* vol. 37, no. 9, pp. 1088-1098, Sept. 1988.

[32] T.W. Lam, L.K. Lee, I.K.K. To, and P.W.H. Wong, "Speed Scaling Functions for Flow Time Scheduling Based on Active Job Count," *Proc. European Symp. Algorithms (ESA),* pp. 647-659, 2008.

[33] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive Entitlement Control of Resource Containers on Shared Servers," *Proc. IFIP/IEEE Int'l Symp. Integrated Network Management,* pp. 163-176, 2005.

[34] C. Lu, T.F. Abdelzaher, J.A. Stankovic, and S.H. Son, "A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers," *Proc. Real-Time Technology Applications Symp. (RTAS),* pp. 51-62, 2001.

[35] C. Lu, J.A. Stankovic, S.H. Son, and G. Tao, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Real-Time Systems,* vol. 23, nos. 1/2, pp. 85-126, 2002.

[36] U. Lublin and D.G. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs," *J. Parallel and Distributed Computing,* vol. 63, no. 11, pp. 1105-1122, 2003.

[37] C. McCann, R. Vaswani, and J. Zahorjan, "A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors," *ACM Trans. Computer Systems,* vol. 11, no. 2, pp. 146-178, 1993.

[38] R. Motwani, S. Phillips, and E. Torng, "Non-Clairvoyant Scheduling," *Proc. Symp. Discete Algorithms (SODA),* pp. 422-431, 1993.

[39] T.D. Nguyen, R. Vaswani, and J. Zahorjan, "Maximizing Speedup through Self-Tuning of Processor Allocation," *Proc. Int'l Parallel Processing Symp. (IPPS),* pp. 463-468, 1996.

[40] P. Padala, K.G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive Control of Virtualized Resources in Utility Computing Environments," *Proc. European Conf. Computer Systems (EuroSys),* pp. 289-302, 2007.

[41] C.A. Phillips, C. Stein, E. Torng, and J. Wein, "Optimal Time-Critical Scheduling via Resource Augmentation (Extended Abstract)," *Proc. Symp. Theory of Computing (STOC),* pp. 140-149, 1997.

[42] K. Pruhs, "Competitive Online Scheduling for Server Systems," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 34, no. 4, pp. 52-58, 2007.

[43] C.J. Ribbens and R. Sudarsan, "ReSHAPE: A Framework for Dynamic Resizing and Scheduling of Homogeneous Applications in a Parallel Environment," *Proc. Int'l Conf. Parallel Processing (ICPP),* p. 44, 2007.

[44] J. Robert and N. Schabanel, "Non-Clairvoyant Batch Set Scheduling: Fairness Is Fair Enough," *Proc. European Symp. Algorithms (ESA),* pp. 741-753, 2007.

[45] S. Sen, "Dynamic Processor Allocation for Adaptively Parallel Jobs," master's thesis, Massachusetts Inst. of Technology, 2004.

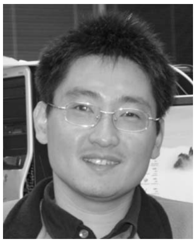
[46] H. Sun, Y. Cao, and W.J. Hsu, "Competitive Two-Level Adaptive Scheduling Using Resource Augmentation," *Proc. Job Scheduling Strategies Parallel Processing (JSSPP) Workshop,* pp. 1-25, 2009.

[47] H. Sun and W.J. Hsu, "Adaptive B-Greedy (ABG): A Simple Yet Efficient Scheduling Algorithm," *Proc. Int'l Workshop System Management Techniques, Processes, and Services (SMTPS) in Conjunction with Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 1-8, 2008.

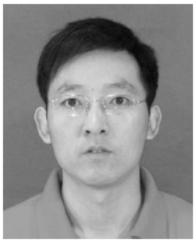
[48] A. Tucker and A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors," *Proc. Symp. Operating Systems Principles (SOSP),* pp. 159-166, 1989.

[49] Z. Wang, X. Zhu, and S. Singhal, "Utilization vs. SLO-Based Control for Dynamic Sizing of Resource Partitions," *Proc. Distributed Systems: Operations and Management (DSOM) Workshop,* pp. 133-144, 2005.

[50] J.B. Weissman, L.R. Abburi, and D. England, "Integrated Scheduling: The Best of Both Worlds," *J. Parallel and Distributed Computing,* vol. 63, no. 6, pp. 649-668, 2003.



Hongyang Sun received the BE degree in computer engineering from Nanyang Technological University, Singapore, in 2005, and the MSc degree in computer science from the National University of Singapore under the Singapore-MIT Alliance the following year. He is currently working toward the PhD degree from Nanyang Technological University. His main research interests include parallel processing and online scheduling. He is a student member of the IEEE.



Yangjie Cao received the BSc degree in mathematics from Henan Normal University, China, in 1999, and the MSc degree in computer science from Zhengzhou University, China, in 2006. He is currently working toward the PhD degree from Xi'an Jiaotong University, China. He worked in industry from 1999 to 2003. His main research interests include adaptive scheduling and high-performance computing.

Wen-Jing Hsu is currently with the School of Computer Engineering, Nanyang Technological University, Singapore (NTU). He has been a faculty fellow of the Singapore-MIT Alliance (SMA) Computer Science Program since 2001. At NTU, he has initiated and completed several large-scale externally funded research projects. He has authored or coauthored many papers in parallel and distributed processing, among them four papers were nominated for Best Paper Awards. He is currently serving as the deputy director of the Maritime Research Center, NTU and has published a book titled *Anatomy of HCTS—A High-Capacity Container Handling System for Mega Vessels*. Earlier, he was the director of the Center of Advanced Information Systems, and deputy director of financial engineering, NTU. He also held positions in the faculty of Michigan State University, and was a visiting scientist to the IBM T. J. Watson Research Center and the IBM Palo Alto Scientific Center. He is a senior member of the IEEE.