Adaptive B-Greedy (ABG): A Simple yet Efficient Scheduling Algorithm

Hongyang Sun Wen-Jing Hsu

School of Computer Engineering,
Nanyang Technological University, Singapore
{sunh0007, hsu}@ntu.edu.sg

Abstract

In order to improve processor utilizations on parallel systems, adaptive scheduling with parallelism feedback was recently proposed. A-Greedy, an existing adaptive scheduler, offers provably-good job execution time and processor utilization. Unfortunately, it suffers from unstable feedback and hence unnecessary processor reallocations even when the job has constant parallelism. This problem may cause difficulties in the management of system resources.

We propose a new adaptive scheduler called ABG (for Adaptive B-Greedy), which ensures both performance and stability. In a direct comparison with A-Greedy using simulated data-parallel jobs, ABG shows an average 50% reduction in wasted processor cycles and an average 20% improvement in running time. For a set of jobs, ABG also outperforms A-Greedy by 10% to 15% on average in terms of both makespan and mean response time, provided the system is not heavily loaded. Our detailed analysis shows that ABG indeed offers improved transient and steady-state behaviors in terms of control-theoretic metrics. Using trim analysis, we show that ABG provides nearly linear speedup for individual jobs and good processor utilizations. Using competitive analysis, we also show that ABG offers good makespan and mean response time bounds.

1 Introduction

Conventional approaches to scheduling a parallel job with a fixed number of processors can often cause processor waste and job execution delays if the job's parallelism varies with time [1]. In fact, many parallel jobs can be designed to run with a variable number of processors during their execution, and such jobs are called malleable jobs [8]. *Adaptive scheduling* that periodically adjusts the processor allocations to malleable jobs can thus improve processor utilization and speed up job execution.

A two-level architecture has provided a convenient framework for adaptive scheduling of malleable jobs [1, 8], where a system-level *OS allocator* allots processors to jobs and a user-level *task scheduler* schedules a job's tasks onto the allotted processors. The execution of the job is divided into *scheduling quanta*. The task scheduler periodically provides feedback in terms of processor requests to the OS allocator between scheduling quanta, and based on the system policy and the task scheduler's request, the OS allocator decides on an appropriate number of processors to allot to the job for the next quantum. Since the future parallelism of the job is usually unknown prior to the job's execution, the challenge is for the task scheduler to make processor requests in a *non-clairvoyant* fashion.

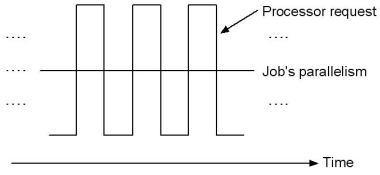

Figure 1. Request instability of A-Greedy.

Agrawal, He, Hsu, and Leiserson [1] proposed an adaptive task scheduler A-Greedy that provides feedback to the OS allocator based on the statistics of the job in the previous quantum. It calculates the processor requests using a multiplicative-increase multiplicative-decrease strategy and schedules the tasks of the job greedily with the allotted processors. A-Greedy is shown to be efficient in terms of running time and processor utilizations when scheduling a malleable job. However, simple analysis also reveals certain problems of A-Greedy in terms of its transient behaviors of processor requests. As shown in Figure 1, it suffers from request instability even when the parallelism of the job stays constant. The fluctuating processor requests can cause potential difficulties for the management of processor resources, such as unnecessary processor waste and job execution delays as well as unnecessary reallocation overheads and loss of localities, etc., which tend to worsen with increased job parallelism.

We present a new adaptive scheduler called Adaptive B-Greedy (ABG for short) under the same two-level scheduling framework. ABG includes a scheduling algorithm B-Greedy and a processor request calculation algorithm A-Control. B-Greedy schedules the tasks of a job in a greedy manner, but with a breath-first strategy, which allows it to measure more precisely the job's average parallelism in each quantum. A-Control calculates the processor requests using an adaptive controller. The calculation is based on classical control theory, which has been routinely applied in scheduling real-time systems (e.g. [17, 20]) and designing computing applications (e.g. [14]). Ours appears to be its first application in scheduling multiprocessor systems. Using control-theoretic analysis, we show that the processor requests calculated by A-Control is able to achieve much improved transient and steady-state performances that A-Greedy fails to attain.

We also prove from algorithmic perspective the running time and the processor utilizations of scheduling a malleable job with ABG. As with [1, 6], we model a malleable job as a dynamically unfolding directed acyclic graph (dag). We use two intrinsic characteristics of the job in our analysis, namely the *work* T_1, which is the total number of vertices (each one represents a unit-size task) in the dag, and the *critical-path length* T_∞, which is the number of nodes on the longest dependency chain in the dag. Moreover, we identify a new job characteristic, which we call the *transition factor*, to characterize the variation of job's average parallelism between any two adjacent scheduling quanta. The transition factor is an important job characteristic because it fundamentally affects the effectiveness of an adaptive and non-clairvoyant scheduler. We will formally define transition factor in Section 5. Intuitively, it indicates the inherent difficulty to schedule a given job, and we argue that this factor should be reflected in the performance analysis.

In addition, to analyze the running time of a job when scheduled by ABG, we use a statistics-inspired *trim analysis* [1], which shows the limit of an OS allocator even if it behaves like an adversary that prevents any task scheduler from achieving linear speedup. We will describe more about trim analysis in Section 6. The readers can also refer to [1–3] for an excellent argument on the application of trim analysis in adaptive scheduling.

Our main contributions lie in analytically showing that ABG has the following properties:

- ABG achieves good transient and steady-state performances in terms of its processor requests. The processor requests satisfy bounded-input bounded-output (BIBO) stability, zero steady-state error, zero overshoot, and convergence at a user-configurable rate r.
- ABG finishes a job with work T_1, critical-path length T_∞, and transition factor C_L on a machine with P processors and quantum length L in $T \leq \frac{2T_1}{\widetilde{P}} + (C_L + 2)\,T_\infty + L = O(\frac{T_1}{\widetilde{P}} + C_L T_\infty + L)$ time, and wastes no more than $W = O(C_L T_1 + P \cdot L)$ processor cycles, where $\widetilde{P}$ denotes the $O(C_L T_\infty + L)$-trimmed processor availability and the convergence rate needs to satisfy $r < 1/C_L$.
- ABG can be coupled with a fair and non-reserving (defined in Section 5) OS allocator to schedule a set of jobs $\mathcal{J}$ on a machine with P processors and quantum length L. If $|\mathcal{J}| \leq P$, i.e., the number of jobs is no more than the number of processors, then the makespan is bounded by $M = O(C_L M^* + L \cdot |\mathcal{J}|)$ for any set of jobs with arbitrary release times, and the mean response time is bounded by $R = O(C_L R^* + L \cdot |\mathcal{J}|)$ for any set of jobs released in a batched fashion, where M^* and R^* denote the lower bounds of makespan and mean response time, respectively. Here, C_L denotes the maximum transition factor of all jobs and the convergence rate needs to satisfy $r < 1/C_L$.

We conduct simulations with data-parallel jobs to evaluate the algorithmic performances of ABG. Simulation results indicate that ABG performs much better on this type of jobs than what the theoretical bounds predict, and that ABG achieves an average 20% improvement in running time and 50% reduction in processor waste as compared to A-Greedy. In addition, when the system is not heavily loaded in a multiprogrammed environment, ABG also outperforms A-Greedy by 10% to 15% on average for both makespan and mean response time of the job set.

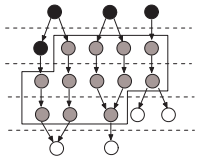

Figure 2. Tasks scheduled by B-Greedy.

The remaining parts of the paper are organized as follows. Section 2 introduces B-Greedy task scheduling algorithm. Section 3 presents A-Control processor request calculation scheme. Section 4 provides the control-theoretic analysis of ABG. Section 5 introduces some definitions and algorithmic preliminaries, followed by the algorithmic analysis in Section 6. Section 7 describes the simulation and comparison of ABG and A-Greedy in terms of the algorithmic performances. Section 8 reviews the related work of adaptive scheduling from both control and algorithm perspectives. Finally, Section 9 summarizes our findings and suggests some future work.

2 B-Greedy Task Scheduling

The B-Greedy task scheduling algorithm uses a variant of the well-known greedy scheduler [10] augmented with a breadth-first level-by-level strategy to schedule the tasks of a job. Suppose that a job gets $a(q)$ processors in quantum q, then on any time step of the quantum, a greedy scheduler schedules any $a(q)$ tasks of the job if there are more than $a(q)$ tasks that are ready, i.e., tasks whose parents are already executed. Otherwise, it schedules all ready tasks. B-Greedy works similarly, except that it gives priority to the ready task with the lowest level, which is the length of the longest chain from the source node(s) of the dag to the task. This breadth-first scheduling strategy ensures that no task at level l completes later than any task at level $l + 1$.

One rationale of B-Greedy is that it attempts to fully expand the parallelism of the job, and therefore tries to make faster progress on its critical path at all time. Furthermore, it may be adapted easily to measure the average parallelism of the job in a scheduling quantum. For each quantum q, B-Greedy collects the *quantum work* $T_1(q)$, which is the number of tasks the scheduler has completed in the quantum, and the *quantum critical-path length* $T_\infty(q)$, which is the number of levels the job has advanced in the quantum. The *quantum average parallelism* $A(q)$ is then computed as $A(q) = T_1(q)/T_\infty(q)$. Note that the quantum critical-path length $T_\infty(q)$ can be non-integer values if the scheduler has completed some but not all tasks in a level. The fractional part is defined to be the ratio of the number of tasks completed to the total number of tasks on that level. For example, Figure 2 shows in gray the tasks that are scheduled in quantum q by B-Greedy. The quantum work and quantum critical-path length in this case are $T_1(q) = 12$ and $T_\infty(q) = 0.8{+}1{+}0.6 = 2.4$, respectively. Therefore, the quantum average parallelism is $A(q) = T_1(q)/T_\infty(q) = 12/2.4 = 5$.

3 A-Control Processor Request Calculation

A-Control calculates the processor requests using an adaptive controller. As shown in Figure 3, the output of A-Control is the processor request $d(q)$ for quantum q. The request is sent

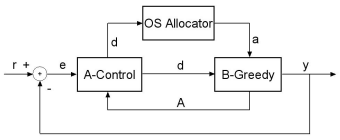

Figure 3. Feedback control structure of ABG.

to the OS allocator, which based on the system policy gives the job an allotment $a(q)$. As with [1], we assume that the OS allocator is *conservative*, that is, it never allots more processors than requested. Thus the allotment $a(q)$ for the job in any quantum q always satisfies $a(q) = \min\{d(q), p(q)\}$, where $p(q)$ is the number of available processors for the job in quantum q based on the OS allocator's system policy. After the job is scheduled by B-Greedy at the end of quantum q, the quantum average parallelism $A(q)$ is collected, and the output $y(q)$ is computed as $y(q) = d(q)/A(q)$. The output is compared with the reference $r(q)$ to produce an error term $e(q) = r(q) - y(q)$, which is used by the adaptive controller for calculation of the processor request $d(q+1)$ for the next quantum $q+1$. The controller applies the following integral control law [14]:

$$d(q+1) = d(q) + K(q+1)e(q), \tag{1}$$

where $K(q+1)$ denotes the controller gain for quantum $q+1$, and determines how aggressively the controller responds to the job's parallelism. It adjusts the processor request for each quantum based on the request and the error of the previous quantum. The controller is adaptive because the gain $K(q+1)$ is reset for each quantum based on the measurement $A(q)$ and some performance specifications. Shortly in Section 4, we will present these specifications and show how to set the controller gain for each quantum to satisfy them from control-theoretic perspective. The processor request for the first quantum is simply set to be $d(1) = 1$.

4 Control-Theoretic Analysis

The adaptive controller shown in Figure 3 dynamically adjusts its controller gain based on the time-varying parallelism of the job and is referred to as *self-tuning regulator* [5] in classical control theory. We now determine how the controller gain can be set for each scheduling quantum via control-theoretic analysis. We basically transform the system into z-domain, and employ pole placement strategy [14] by considering a set of transient and steady-state performance specifications. Our analysis directly applies to times when the job's average parallelism is constant. At other times when the average parallelism is changing, these specifications are not properly defined, however.

Assume that the job's average parallelism is A, and it will stay constantly at A for sufficiently long time. The controller gain K, which depends on the value of A, will also stay constant in the mean time. Ideally the processor request should match the job's average parallelism for a quantum to be efficient. Hence, the reference in our scheduler is set to be 1 for all quanta, which corresponds to a unit-step function. Thus, we can represent the reference as well as A-Control and B-Greedy in z-domain:

- Reference: $R(z) = z/(z-1)$.
- A-Control: $G(z) = D(z)/E(z) = K/(z-1)$.
- B-Greedy: $S(z) = Y(z)/D(z) = 1/A$.

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

$$T(z) = \frac{Y(z)}{R(z)} = \frac{G(z)S(z)}{1 + G(z)S(z)} = \frac{K/A}{z - (1 - K/A)}. \tag{2}$$

Clearly, the closed-loop is a first-order system with single pole $p_0 = 1 - K/A$. We adopt the following set of criteria [14] commonly used in control theory to place the position of the pole by setting the value of controller gain K.

- *BIBO-Stability*. The system is bounded-input bounded-output (BIBO) stable if given a bounded reference, the processor request is also bounded.
- *Steady-State Error*. The steady-state error is given by the difference between the processor request 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 request and its steady-state value.
- *Convergence Rate*. The convergence rate r is the speed at which the processor request approaches the job's average parallelism. Specifically, r is defined to be $r = (|d(q+1) - A|) \,/\, (|d(q) - A|)$.

These four criteria specify the transient and steady-state performance of the scheduler when the average parallelism of the job stays constant. We show that ABG has good performance in terms of these criteria in the following theorem.

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

Proof. The proof is straightforward using control theory. The details can be found in the full version of this paper [23]. □

Since the average parallelism $A(q)$ of the job is measured for each scheduling quantum $q \geq 1$, the controller gain according to Theorem 1 is set to be $K(q) = (1-r)A(q-1)$ for $q \geq 2$. Substituting this into Equation (1), we get the following relation on the processor request,

$$d(q) = \begin{cases} rd(q-1) + (1-r)A(q-1) & \text{if } q > 1, \\ 1 & \text{if } q = 1, \end{cases} \tag{3}$$

which is essentially the algorithm for calculating processor request for each quantum. A special case of this algorithm occurs when the convergence rate satisfies $r = 0$. This gives the fastest rate of convergence, or one-step convergence. The resulting processor request for each quantum q is then equal to the job's average parallelism of the previous quantum, i.e., $d(q) = A(q-1)$.

Figure 4 shows the behavior of ABG as compared to A-Greedy on a synthetic job with constant parallelism over 8 scheduling quanta. The convergence rate of the controller is set to be 0.2 in this case, and the multiplicative factor of A-Greedy is set to be 2. As can be readily seen, ABG satisfies good transient and steady-state performances as stated in Theorem 1 while

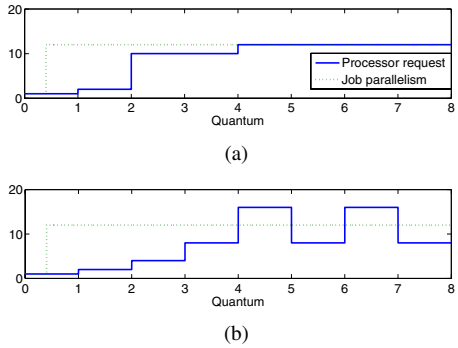

Figure 4. Transient and steady-state behaviors of (a) ABG (b) A-Greedy.

A-Greedy suffers from apparent oscillating instability, nonzero steady-state error and significant overshoot. These problems of A-Greedy can cause unnecessary processor waste and job execution delays, which worsens with increased job parallelism. Practically, because of its instability, A-Greedy may also suffer from unnecessary context switching overhead and loss of locality, etc. Thus, it is important to take control-theoretic performances into consideration when evaluating an adaptive scheduler.

5 Algorithmic Preliminaries

5.1 Quantum Efficiency

A scheduling quantum of a job is said to be a *full quantum* iff there is work done on each time step of the quantum. We assume that the OS allocator is *fair* and *non-reserving*[1], and that the number of jobs in the system is no more than the number of processors. Thus each job receives at least 1 processor at all times. Therefore all except the last quantum of a job are guaranteed to be full quanta. Suppose that the quantum length is L, then for each full quantum q with allotment $a(q)$, the quantum work satisfies $T_1(q) \leq a(q)L$ and the quantum critical-path length satisfies $T_\infty(q) \leq L$. We define the *quantum work efficiency* to be $\alpha(q) = \frac{T_1(q)}{a(q)L}$, and the *quantum critical-path length efficiency* to be $\beta(q) = \frac{T_\infty(q)}{L}$. Thus, both $\alpha(q)$ and $\beta(q)$ satisfy $0 < \alpha(q), \beta(q) \leq 1$. The quantum average parallelism $A(q)$ is therefore

$$A(q) = \frac{T_1(q)}{T_\infty(q)} = a(q)\frac{\alpha(q)}{\beta(q)}. \tag{4}$$

Since B-Greedy schedules a job in breadth-first manner, treating part of the job completed in each quantum as a separate job, we can apply a well-known result [6,10] from greedy scheduling, which is $L \leq \frac{T_1(q)}{a(q)} + T_\infty(q) = \left(\frac{\alpha(q)}{\beta(q)} + 1\right) T_\infty(q)$. Substituting in $T_\infty(q) = \beta(q)L$, we get

$$\alpha(q) + \beta(q) \geq 1. \tag{5}$$

Inequality (5) gives a lower bound on the sum of quantum work efficiency and quantum critical-path length efficiency of any full quantum. We will use this relationship as well as Equation (4) later in our analysis to derive the upper bounds on running time and processor waste.

[1] An OS allocator is *fair* if it gives all jobs an equal number of processors unless a job requests for less. An OS allocator is *non-reserving* if it never keeps any processor idle unless no job requests for more.

5.2 Transition Factor

The *transition factor* of a job, denoted by C_L, where $C_L \geq 1$, is defined to be the maximal ratio on the average parallelism of any two adjacent full quanta for a given quantum length L. Formally, for a job with transition factor C_L, the average quantum parallelism of the job satisfies

$$\frac{1}{C_L} \leq \frac{A(q)}{A(q-1)} \leq C_L, \tag{6}$$

for $q \geq 1$, and $A(0)$ is defined to be 1.

The transition factor of a job, like the work and critical-path length, is an intrinsic job characteristic, and is independent of the task scheduler used to schedule it. [2] Intuitively, it represents how fast the parallelism of the job changes with time and thus reflects the degree of difficulty to schedule the job in an adaptive and non-clairvoyant fashion.

5.3 Processor Request Bounds

We now show that the processor requests generated by ABG can be bounded from both above and below provided that certain conditions are satisfied.

Lemma 2 *Suppose that ABG schedules a job with transition factor C_L on a machine with quantum length L. Then the processor request $d(q)$ for each full quantum q satisfies*

$$\frac{1-r}{C_L - r}A(q) \leq d(q) \leq \frac{C_L(1-r)}{1-C_L r}A(q), \tag{7}$$

where $A(q)$ is the job's average parallelism in quantum q and r denotes the convergence rate. The inequality on the right only holds provided $r < 1/C_L$.

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

Base case: For $q = 1$, we have $d(1) = 1$. Because $A(0) = 1$, and according to Inequality (6), the average parallelism $A(1) \geq 1/C_L$. Thus we have $\frac{d(1)}{A(1)} \leq C_L \leq \frac{C_L - C_L r}{1 - C_L r}$, since $C_L \geq 1$ and $C_L r < 1$. Therefore, we get $d(1) \leq \frac{C_L(1-r)}{1-C_L r}A(1)$.

Induction: For $q \geq 2$, suppose that we have $d(q-1) \leq \frac{C_L(1-r)}{1-C_L r}A(q-1)$. Because $A(q-1) \leq C_L A(q)$ from Inequality (6), then according to the processor request algorithm in Equation (3), we have for quantum q, $d(q) = rd(q-1) + (1-r)A(q-1) \leq \frac{C_L r(1-r)}{1-C_L r}A(q-1) + (1-r)A(q-1) = \frac{1-r}{1-C_L r}A(q-1) \leq \frac{C_L(1-r)}{1-C_L r}A(q)$. □

Remarks. The assumption $r < 1/C_L$ is required for the upper bound of the processor request to hold. Without this assumption, unfortunately, the ratio of the processor request and the quantum average parallelism cannot be bounded from above. The worst case happens when the parallelism of the job reduces much faster than the responsiveness of the chosen convergence rate, hence the processor requests cannot be reduced as quickly.

[2] The transition factor, however, does depend on the length of the scheduling quantum. Different quantum length may yield different transition factor for the same job. For a given quantum length and the dag structure of the 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 the critical-path length of a job. We will just make use of these job characteristics to quantify the behavior of our scheduler in terms of performance bounds.

6 Algorithmic Analysis

6.1 Running Time

We use trim analysis [1] for analyzing the running time of ABG. We classify each full quantum q into "accounted", namely the quantum that is accounted for speedup, and "deductible", namely the quantum that is trimmed and not considered. Specifically, a full quantum q is *accounted* if the processor request is deprived, i.e., $a(q) < d(q)$ and the allotment is less than the average parallelism, i.e., $a(q) < A(q)$. Otherwise, the quantum is *deductible* if $a(q) = d(q)$ or $a(q) \geq A(q)$. We prove the running time of ABG by bounding the number of each type of quanta separately.

The trim analysis is used to bound the number of accounted quanta, and it is applied as a tool to limit the power of the OS allocator, which can behave like an adversary to the task scheduler. In particular, the OS allocator may provide a large number of available processors when the parallelism of the job is low, thus preventing any task scheduler from achieving good linear speedup in terms of the average processor availability for the job. Trim analysis trims off a few quanta with the highest number of available processors and shows that a job can achieve nearly linear speedup in terms of the average processor availability on the remaining quanta. The resulting average processor availability is called R-trimmed availability, where R is the number of time steps trimmed. The following theorem states the running time of ABG under trim analysis.

Theorem 3 *Suppose that ABG schedules a job with work T_1, critical-path length T_∞ and transition factor C_L on a machine with quantum length L. Then the running time of the job is bounded by*

$$T \leq \frac{2T_1}{\widetilde{P}} + \frac{C_L + 1 - 2r}{1 - r} T_\infty + L, \tag{8}$$

where $\widetilde{P}$ denotes the $\left(\frac{C_L+1-2r}{1-r}T_\infty + L\right)$-trimmed processor availability and r denotes the convergence rate.

Proof. Let A denote the set of accounted quanta. From the definition of accounted quantum, Equation (4) and Inequality (5), we have $\alpha(q) \geq 1/2$ for each $q \in A$, i.e., $T_1(q) \geq \frac{a(q)L}{2}$. Since $a(q) < d(q)$ on accounted quanta, and OS allocator allocates $a(q) = \min\{d(q), p(q)\}$ processors to the job, we have $a(q) = p(q)$. The total work done on accounted quanta satisfies $T_1 \geq \sum_{q\in A} T_1(q) \geq \sum_{q\in A} \frac{a(q)L}{2} = \frac{L}{2}\sum_{q\in A} p(q) = \frac{L}{2}\overline{P}|A|$, where $\overline{P}$ is the average processor availability on accounted quanta. The total number of accounted quanta is then $|A| \leq \frac{2T_1}{\overline{P}L}$.

Let D denote the set of deductible quanta. From the definition of deductible quantum and Lemma 2, we have $a(q) \geq \frac{1-r}{C_L-r}A(q)$ for each $q \in D$. Substituting Equation (4) and Inequality (5) into it, we obtain $\beta(q) \geq \frac{1-r}{C_L+1-2r}$, i.e., $T_\infty(q) \geq \frac{1-r}{C_L+1-2r}L$. The sum of the quantum critical-path length in deductible quanta thus satisfies $T_\infty \geq \sum_{q\in D} T_\infty(q) \geq \sum_{q\in D} \frac{1-r}{C_L+1-2r}L = \frac{1-r}{C_L+1-2r}|D|L$. Therefore, the total number of deductible quanta is $|D| \leq \frac{C_L+1-2r}{(1-r)L}T_\infty$.

Let N denote the set of non-full quanta. Since only the last quantum of the job may be a non-full quantum, we have $|N| \leq 1$. Combining these bounds, the running time of the job satisfies $T \leq \frac{2T_1}{\overline{P}} + \frac{C_L+1-2r}{1-r}T_\infty + L$. Since $\overline{P}$ is the average processor availability on all accounted quanta, that is, the processor availability after trimming at most $\frac{C_L+1-2r}{1-r}T_\infty + L$ non-accounted quanta, it is clearly no less than the $\left(\frac{C_L+1-2r}{1-r}T_\infty + L\right)$-trimmed processor availability $\widetilde{P}$. Thus, substituting $\overline{P} \geq \widetilde{P}$ into the running time bound, we proved the theorem. □

6.2 Processor Waste

We analyze the number of processor cycles wasted when scheduling a job using ABG. As pointed out in the remark of Lemma 2, bounding processor waste relies on the convergence rate to satisfy $r < 1/C_L$ for a job with transition factor C_L. However, since the job characteristics are usually unknown prior to its execution, we assume that the convergence rate is chosen based on some historical characterization of the workload, which ensures that it can satisfy the requirement. The processor waste can be bounded in terms of the total work as stated in the following theorem.

Theorem 4 *Suppose that ABG schedules a job with work T_1 and transition factor C_L on a machine with quantum length L. Then the total number of processor cycles wasted is bounded by*

$$W \leq \frac{C_L(1-r)}{1 - C_L r} T_1 + P \cdot L, \tag{9}$$

where r denotes the convergence rate of the scheduler.

Proof. Since we assume $a(q) \leq d(q)$, from Lemma 2, we have for each full quantum q, $a(q) \leq \frac{C_L(1-r)}{1-C_L r}A(q)$. Substituting Equation (4) and Inequality (5) into it, we obtain $\alpha(q) \geq \frac{1-C_L r}{1+C_L-2C_L r}$, i.e., $T_1(q) \geq \frac{1-C_L r}{1+C_L-2C_L r}a(q)L$. Let $w(q)$ denote the waste in the quantum, then $w(q)$ satisfies $w(q) = a(q)L - T_1(q) \leq a(q)L - \frac{1-C_L r}{1+C_L-2C_L r}a(q)L = \frac{C_L(1-r)}{1+C_L-2C_L r}a(q)L \leq \frac{C_L(1-r)}{1-C_L r}T_1(q)$. Let F denote the set of full quanta. The total processor waste in F is then $\sum_{q\in F} w(q) \leq \sum_{q\in F} \frac{C_L(1-r)}{1-C_L r}T_1(q) \leq \frac{C_L(1-r)}{1-C_L r}T_1$. The job's last quantum may not be a full quantum and wastes at most $P \cdot L$ processor cycles. The total waste is obtained by summing up wastes from both types of quanta. □

6.3 Makespan and Mean Response Time

Makespan or mean response time of a set of jobs often indicate the overall system performance in a multiprogrammed environment. Such global measures can depend on both running time and processor waste of a single job as well as the OS allocator's system policy. As was shown by He et al. [11, 12], when a particular task scheduler is coupled with a fair and non-reserving OS allocator such as dynamic equi-partitioning [18] in a two-level scheduling context, the makespan and mean response time of the job set can be bounded by combining the running time and processor waste bounds in a non-trivial manner. Suppose that we couple ABG with a fair and non-reserving OS allocator. We can apply the techniques in [11, 12] and the improved results in [13] to prove the makespan for any set of jobs with arbitrary release times and the mean response time for any set of jobs released in a batched fashion. The resulting performance bounds for the two-level scheduler are represented in terms of their *competitive ratio*, i.e., their performance ratio to the theoretical lower bounds.

The following theorem states the results for both makespan and mean response time. The proof is omitted and can be found in the full version of this paper [23].

Theorem 5 *Suppose that ABG is coupled with a fair and non-reserving OS allocator to schedule a set of jobs $\mathcal{J}$ on a machine with P processors and quantum length L. If the number of jobs is no more than the number of processors in the system, i.e., $|\mathcal{J}| \le P$, then the makespan is bounded by*

$$M \le \left(\frac{C_L + 1 - 2C_L r}{1 - C_L r} + \frac{C_L + 1 - 2r}{1 - r}\right) M^* + L \cdot (|\mathcal{J}| + 2) \tag{10}$$

for any set of jobs with arbitrary release times, and the mean response time is bounded by

$$R \le \left(\frac{2C_L + 2 - 4C_L r}{1 - C_L r} + \frac{C_L + 1 - 2r}{1 - r}\right) R^* + L \cdot (|\mathcal{J}| + 2) \tag{11}$$

for any set of jobs released in a batched fashion, where M^* and R^* denote the theoretical lower bounds for the makespan and mean response time, respectively. C_L denotes the maximum transition factor of all jobs in the system and r denotes the convergence rate, where $r < 1/C_L$. □

6.4 Comparison with A-Greedy

We briefly interpret the performance bounds of ABG and compare them with those of A-Greedy. Suppose that the convergence rate r of ABG can be chosen from the historical statistics of the workload such that $C_L r$ can be upper-bounded by a constant less than 1, the performance bounds can then be simplified to the following,

$$\begin{aligned} T &\le 2 \cdot \frac{T_1}{\widetilde{P}} + (C_L + 2)\, T_\infty + L = O(\frac{T_1}{\widetilde{P}} + C_L T_\infty + L), \\ W &= O(C_L T_1 + P \cdot L), \\ M &= O(C_L M^* + L \cdot |\mathcal{J}|), \\ R &= O(C_L R^* + L \cdot |\mathcal{J}|). \end{aligned}$$

where $\widetilde{P}$ is the $O(C_L T_\infty + L)$-trimmed processor availability. As can be seen, the transition factor C_L of the job plays a crucial role in reflecting the performances of ABG. Nearly linear speedup can be achieved if the average parallelism of the job dominates the average processor availability and the job has a relatively small transition factor. For a given job set $\mathcal{J}$ and quantum length L, the makespan and mean response time satisfy $M = O(C_L)M^*$ and $R = O(C_L)R^*$ respectively, and therefore are both $O(C_L)$-competitive. In a direct comparison with A-Greedy in terms of the bounds, ABG tends to perform better when the transition factor of the job is small. However, when the job has large swaying parallelism over different quanta and hence exhibits a large C_L, ABG may perform worse. A-Greedy is oblivious of the transition factor in its analysis because of its multiplicative-increase multiplicative-decrease strategy, whose symmetric structure allows it to bypass this difficulty. However, A-Greedy does possess large multiplicative constants inside the O-notations for its performance bounds [1, 11].

7 Simulations

7.1 Simulation Setup

Our simulations use similar setups as the one used to test A-Greedy [12]. In particular, we test the schedulers on data-parallel jobs that have fork-join structures, which alternate between serial and parallel phases. In order to examine the effect of a job's transition factor on the performance of the schedulers, we generate jobs with different transition factors by varying the level of parallelism in the parallel phases. For a specific transition factor, we also generate jobs with variable work and critical-path length by changing the length of each serial and parallel phase. In the simulation, 50 jobs are generated for each transition factor ranging from 2 to 100. The number of processors is set to be $P = 128$ and the quantum length is set to be $L = 1000$ steps. The scheduling overheads due to reallocation of processors and processor request calculations are ignored.

To set the parameters for the schedulers, we choose $r = 0.2$ for the convergence rate[3] of ABG, and maintain the same parameter settings for A-Greedy as in [12]. Note that the choice of the convergence rate in this case does not satisfy $r < 1/C_L$ for values of C_L that are greater than or equal to 5, and hence cannot guarantee the theoretical performance bounds on the processor waste, makespan and mean response time. Nevertheless, as will be seen shortly, the simulation results do not seem to be affected practically.

We conduct two sets of simulations. The first set features individual jobs with different transition factors running on the P processors alone, which is aimed at measuring running time and processor waste of the schedulers in an unconstrained environment. The second set of simulation features job sets with different loads space-sharing all processors in the system. We measure in this case the global performances, that is, the makespan and the mean response time to show how the schedulers perform in the multiprogrammed environment.

7.2 Simulation Results

In the first set of simulations where only a single job is run each time, according to our setup, all processor requests from both schedulers are granted. This allows us to evaluate the performances of both schedulers under a favorable circumstance, for otherwise the advantage of a more efficient processor request calculation scheme can not be reflected. Figure 5 shows the effects of the transition factor on the running time and the processor waste for both schedulers. In Figure 5(a), the running time is normalized in terms of the critical-path length of the job, which in the unconstrained environment is the optimal running time, and in Figure 5(c), the processor waste is normalized in terms of the job's total work. Each point in Figures 5(a) and 5(c) is averaged 50 different job runs with the same transition factor. Figures 5(b) and 5(d) show the running time and processor waste ratios of A-Greedy over ABG for each run.

As can be seen, except for some small values of the transition factor, where both task schedulers perform comparably, ABG performs consistently better than A-Greedy for most values of

[3] We have tried varying the value for the convergence rate. The results do not deviate too much for all values of convergence rate less than 0.6.

the transition factor. An average of 20% improvement in running time and 50% reduction in processor waste can be observed. In addition, unlike what the theoretical bounds in Section 6 predict, increasing the value of transition factor does not seem to have much effect on ABG while it continues to affect the performance of A-Greedy. This is probably because the theoretical analysis assumes the worst case scenario, which is unlikely to happen in a practical setting here. Also notice that A-Greedy responds to the increase of the transition factor for both running time and processor waste in an oscillatory and complementary fashion. This is due to the multiplicative-increase nature of its processor request calculation, which effectively doubles the processor request whenever the average parallelism of the job exceeds a certain threshold.

In the second set of simulations, we group together jobs with different transition factors to form job sets that have different loads on the system. The load is defined to be the average parallelism of the entire job set normalized by the total number of processors. For heavy system loads, therefore, some processor requests may not be granted. We couple both task schedulers with dynamic equi-partitioning OS allocator in the simulation. A total of 5000 job sets of different loads are run and Figure 6 shows the performance of ABG and A-Greedy in terms of their makespan and mean response time on these job sets. Figures 6(a) and 6(c) compare the average makespan and mean response time of both schedulers with the theoretical lower bounds at different loads.[4] Figures 6(b) and 6(d) show the makespan and the mean response time ratios of A-Greedy over ABG for each of the 5000 job sets.

We can see that under light system loads, where processor requests tend to be granted for both schedulers, ABG performs better than A-Greedy by 10% to 15% on average in both makespan and mean response time. This is because of ABG's more efficient processor request calculation. However, under heavy system loads, the processor requests tend to be deprived and the advantage of ABG diminishes since neither of the schedulers have direct control over the processor allocations. Therefore, the performance of both schedulers are comparably similar. Note that Figures 6(a) and 6(c) in particular resemble the performance shown for A-Greedy in [12], which verifies that our simulation provides a fair comparison between the two schedulers.

8 Related Work

Agrawal et al. [1–3] studied adaptive task scheduling both theoretically and empirically using parallelism feedback, and proposed two task schedulers based on the multiplicative-increase multiplicative-decrease strategy, namely a centralized A-Greedy and a distributed A-Steal. They also introduced trim analysis, and proved that both schedulers are efficient in terms of the running time and processor waste. He et al. [11, 12] combined the task schedulers with dynamic equi-partitioning [18] and round-robin OS allocators, and proved that the resulting two-level schedulers are $O(1)$-competitive in terms of the makespan and mean response time for a set of jobs. They also provided generalized proof techniques and improved results [13] for these global measures. In this paper, we adopted the trim analysis and proved the efficiency of ABG.

Other existing work in adaptive task scheduling includes an ABP scheduler [4] that is based on distributed and randomized work-stealing, but without using parallelism feedback. The empirical study in [2] showed that A-Steal gives much better performance than ABP. In addition, Sen [21], Nguyen et al. [19] and Corbalán et al. [7] measured certain job characteristics such as average parallelism, speedup, efficiency, etc. at runtime and used them as some kind of feedback in task scheduling empirically. Similar job characteristics are also considered by Sevcik [22] when designing static processor allocation policies. In this paper, we introduced another job characteristic, which is the transition factor, and incorporated it into our analysis.

Adaptive scheduling has also been studied from control-theoretic perspective. Related work in this area tends to focus on the transient performances in terms of control-theoretic properties. Lu et al. [16, 17] presented a feedback control real-time scheduling framework for adaptive real-time systems, and developed a 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 scheduling. Goel et al. [9] designed an adaptive controller that schedules real-rate applications by estimating the application's progress with timestamps in a feedback loop. Using adaptive control, Padala et al. [20] developed a resource management system to optimize the resource utilization and at the same time to meet specific QoS goals for multi-tier applications. Similar approaches are also used in [15, 24] for dynamically adjusting the resource partitioning for enterprize servers. In this paper, we applied control theory to scheduling malleable jobs in multiprocessor systems, and showed that our scheduler possesses good control-theoretic behaviors.

9 Conclusion and Future Work

Many researchers have applied control theory to design computing and resource management systems, and these control-inspired algorithms often demonstrate robust system behaviors. However, as far as we know, there is little existing work in the literature that incorporates performance metrics from both control and algorithmic perspectives. We have presented an adaptive control-based job scheduling algorithm ABG for scheduling malleable jobs in a multiprogrammed multiprocessor system. We have analyzed the scheduler from both control-theoretic and algorithmic perspectives, and demonstrated its good performance through control-theoretic specifications and algorithmic bounds. We have also compared the performance of ABG with A-Greedy via simulations and observed ABG's superior performances from both view points.

Possible future research in this area may include dynamically adjusting the quantum length and other parameters to achieve better system wide adaptivity. Also, beside the transition factor, alternative job characteristics such as the frequency on the change of parallelism, or the variance, etc. can be considered when analyzing adaptive schedulers. Finally, our analysis on

[4] Figures 6(a) and 6(c) indicate that both makespan and mean response time ratios increase with the system load initially and decrease after a peak value. Similar effects can be observed in the simulations by [12]. This results from the fact that we use two theoretical lower bounds for both makespan and mean response time, and the scheduler's performance ratio to one lower bound increases with the system load and tends to dominate when the load is small while the ratio to the other lower bound decreases with the system load and tends to dominate when the system load is high. The peaks represent the intersections of the ratios to the two lower bounds.

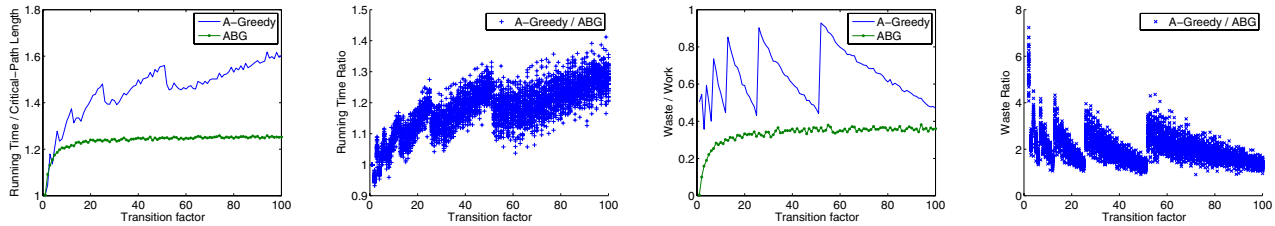

Figure 5. The running time and the processor waste of ABG and A-Greedy.

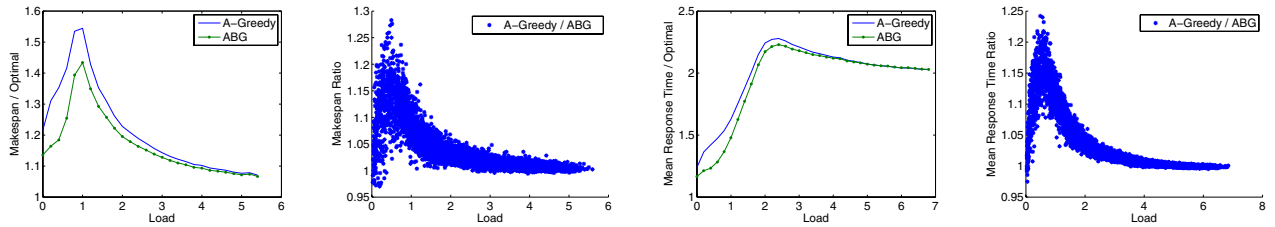

Figure 6. The makespan and the mean response time of ABG and A-Greedy.

ABG restricts the value of the convergence rate to attain the desirable performance bounds. To achieve tighter analysis without this constraint will be another interesting challenge.

Acknowledgements

The first author would like to thank Yuxiong He for pointing out the interpretation of simulation results in Figure 6(a) and Figure 6(c). He also thanks Yuxiong, Hui Fang, and Shin Yee Chung for many enjoyable discussions.

References

[1] K. Agrawal, Y. He, W.-J. Hsu, and C. E. Leiserson. Adaptive scheduling with parallelism feedback. In *PPoPP*, pages 100 – 109, New York City, NY, USA, 2006.

[2] K. Agrawal, Y. He, and C. E. Leiserson. An empirical evaluation of work stealing with parallelism feedback. In *ICDCS*, pages 19 – 29, Lisbon, Portugal, 2006.

[3] K. Agrawal, Y. He, and C. E. Leiserson. Adaptive work stealing with parallelism feedback. In *PPoPP*, pages 112–120, San Jose, California, USA, 2007.

[4] N. S. Arora, R. D. Blumofe, and C. G. Plaxton. Thread scheduling for multiprogrammed multiprocessors. In *SPAA*, pages 119–129, Puerto Vallarta, Mexico, 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. *Journal of the ACM*, 46(5):720–748, 1999.

[7] J. Corbalán, X. Martorell, and J. Labarta. Performance-driven processor allocation. *IEEE Transactions on Parallel and Distributed Systems*, 16(7):599–611, 2005.

[8] D. G. Feitelson. Job scheduling in multiprogrammed parallel systems (extended version). Technical report, IBM Research Report RC 19790 (87657) 2nd Revision, 1997.

[9] A. Goel, J. Walpole, and M. Shor. Real-rate scheduling. In *RTAS*, pages 434–441, Toronto, Canada, 2004.

[10] R. L. Graham. Bounds on multiprocessing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

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

[12] Y. He, W.-J. Hsu, and C. E. Leiserson. Provably efficient online nonclairvoyant adaptive scheduling. In *IPDPS*, pages 1–10, Long Beach, California, USA, 2007.

[13] Y. He, H. Sun, and W.-J. Hsu. Adaptive scheduling of parallel jobs on functionally heterogeneous resources. In *ICPP*, Xi'an, China, 2007.

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

[15] X. Liu, X. Zhu, S. Singhal, and M. Arlitt. Adaptive entitlement control of resource containers on shared servers. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 163–176, Nice, France, 2005.

[16] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. A feedback control approach for guaranteeing relative delays in web servers. In *RTAS*, pages 51–62, Taipei, Taiwan, 2001.

[17] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1-2):85–126, 2002.

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

[19] T. D. Nguyen, R. Vaswani, and J. Zahorjan. Maximizing speedup through self-tuning of processor allocation. In *IPPS*, pages 463–468, Honolulu, Hawaii, USA, 1996.

[20] 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. In *EuroSys*, pages 289–302, Lisbon, Portugal, 2007.

[21] S. Sen. Dynamic processor allocation for adaptively parallel jobs. Master's thesis, Massachusetts Institute of technology, 2004.

[22] K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In *SIGMETRICS*, pages 171–180, Oakland, California, USA, 1989.

[23] H. Sun and W.-J. Hsu. Adaptive B-Greedy (ABG): A simple yet efficient scheduling algorithm. Technical Report TR-07-03. School of Computer Engineering, Nanyang Technological University, Singapore. http://www.ntu.edu.sg/sunh0007/papers/ABG.pdf.

[24] Z. Wang, X. Zhu, and S. Singhal. Utilization vs. SLO-based control for dynamic sizing of resource partitions. In *DSOM*, pages 133–144, Barcelona, Spain, 2005.