Discrete Applied Mathematics 157 (2009) 364–376

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Idle regulation in non-clairvoyant scheduling of parallel jobs

Andrei Tchernykh [a,*], Denis Trystram [b], Carlos Brizuela [a], Isaac Scherson [c]

[a] Computer Sciences Department, CICESE Research Center, Ensenada 22860, BC, Mexico
[b] ID- IMAG, 38330 Montbonnot St. Martin, France
[c] Department of Computer Science, School of Information and Computer Sciences, University of California, Irvine, CA 92697, USA

ARTICLE INFO

Article history:
Received 6 July 2006
Received in revised form 17 February 2008
Accepted 9 March 2008
Available online 22 April 2008

Keywords:
Scheduling
Parallel jobs
Idle regulation

ABSTRACT

The optimization of parallel applications is difficult to achieve by classical optimization techniques because of their diversity and the variety of actual parallel and distributed platforms and/or environments. Adaptive algorithmic schemes, capable of dynamically changing the allocation of jobs during the execution to optimize global system behavior, are the best alternatives for solving this problem. In this paper, we focus on non-clairvoyant scheduling of parallel jobs with known resource requirements but unknown running times, with emphasis on the regulation of idle periods in the context of general list policies. We consider a new family of scheduling strategies based on two phases which successively combine sequential and parallel execution of jobs. We generalize known worst-case performance bounds by considering two extra parameters, in addition to the number of processors and maximum processor requirements considered in the literature, namely, job parallelization penalty and idle regulation factor. Furthermore, we prove that under certain conditions of idle regulation, the performance guarantee of parallel job scheduling in space-sharing mode can be improved.

1. Introduction

Different types of parallel applications introduce different constraints on scheduling policies depending on their dynamic and static characteristics. Unfortunately, in most cases there is no sufficient knowledge about applications to schedule them efficiently based on models found in the classical theory. Moreover, real application characteristics are too complex to be used as formal scheduling constraints. If the application constraints are unpredictable or undefined, the resource allocation for computing jobs could be done by a scheduler based on available information such as parameters of the processors or the operating system. However, they are also very hard to define formally.

1.1. Applications

Though scheduling problems have been studied for decades [1,3], little is known about the efficiency[1] of on-line and non-clairvoyant parallel job scheduling solutions [26,4]. Parallelism inside of applications adds a new dimension to the scheduling problem. The manner in which the job partitioning can be done (when a job requires more than one processor at a time)

* Corresponding author. Tel.: +52 646 175 0593; fax: +52 646 175 0593.
E-mail address: chernykh@cicese.mx (A. Tchernykh).

[1] Typically three groups of measures in the evaluation of the scheduling policies' efficiency are considered: user centric, system centric and algorithmic centric. User centric evaluation criteria such as mean turnaround time, response time, waiting time, and response ratio help to estimate the algorithms from the point of view of parameters important for users. System centric evaluation criteria (processor utilization, throughput) help assess algorithms from the viewpoint of the system use. Algorithmic centric evaluation criteria (competitive ratio) help to estimate the quality of scheduling algorithms.

0166-218X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2008.03.005

depends on its divisibility property. According to the classification of Feitelson et al. [6,14], two general job classes named rigid and flexible are distinguished depending on job processor requirements. Rigid jobs include parallel jobs that require a fixed number of processors for parallel execution: number that is not changed until the job is completed. Moldable jobs can be run on any number of processors. However, once the number of processors has been allotted to the job it remains the same throughout its execution [13]. Malleable jobs belong to the flexible job class and have the flexible divisible property so that they can be divided into any number of segments of any desired fractional size. The number of processors allotted to the job is not imposed in advance and depends on the number of processors available at the time of allocation, and on the change in requirements or load. At any time, when more processors are available, the same job can be preempted, redistributed and resumed on a different number of processors. Evolving parallel jobs require different number of processors during their execution [11].

1.2. Framework for resource management

As parallel and distributed systems are being deployed to support a wide range of parallel workloads, with diverse characteristics and varying quality-of-service requirements, scheduling becomes a challenging research issue. The variety of job scheduling policies intended for different application/system settings has been studied in the literature [38]. It is clear that practical scheduling solutions require a variety of models and techniques. Managing the performance of systems through direct administrator or user adjustment of scheduling policies is impractical. One possible way to handle realistic scheduling problems is to use a framework that can support different solutions and adapt to them on-line.

In [31], a generic adaptive scheduling strategy, called (a, b, c)-*Scheme*, was introduced. The scheme unifies the scheduling of sequential jobs and jobs that may require more than one processor for their execution. It automatically adapts to the right application granularity by using a set of three parameters: a, b and c referring respectively to system, application and scheduling strategy characteristics. This framework appears to be a good starting point for understanding the unification of different dynamic scheduling strategies. Preliminary results show that using the strategy it is possible to design good approximation algorithms for scheduling parallel jobs [32].

1.3. Penalty factor

To find a trade-off between the complexity of actual parallel systems and the desired simplicity of their models for theoretical study, we consider the model of parallel jobs based on a *penalty factor*. Such a model has been used in various actual programs [2,7]. The idea is to add an overhead to the parallel execution time that includes the time lost for communication, synchronization, preemption and any extra factors that come from the management of the parallel execution. The penalty factor implicitly takes into account some constraints, when they are unknown or very hard to define formally. It hides real application characteristics or computer parameters, though known, but too complex to be used as formal scheduling constraints. It can be considered as a ratio of the ideal speedup over the observed one. The penalty of a single job execution or workload processing can be also estimated based on an empirical analysis, benchmarking, counting, profiling, performance evaluation through modeling or measurement, or based on the information provided by a user. In the job model of Downey [8], the speedup function needed to estimate the penalty function is modeled using three parameters: the average parallelism of the job, the variance in parallelism and the number of processors. Given these parameters, a hypothetical parallelism profile can be constructed and the speedup is calculated.

Several qualitative shapes of the penalty as a function of the number of processors allotted to the job have been discussed in the literature [2]: constant, linear, and convex. They are the most common classes when parallelizing actual numerical applications. The constant shape of the penalty function corresponds to a system where applications achieve near linear speedup with increasing number of processors. The linear penalty function corresponds to the logarithmic shape of a job speedup function that exhibits speedup that mostly rises monotonically up to the certain threshold and slows down beyond a certain number of allocated processors. The convex penalty function corresponds to the small start-up overhead. The addition of extra processors costs minimum up to the certain threshold. Beyond the threshold the cost of the parallelism management is increased causing reduction of the speedup. It correlates with the concave speedup function.

We exclude from analysis the applications with super linear speedup that may be obtained for some big applications exceeding cache or available memory, and applications with speedup value less than one (that do not contain enough parallelism or have a large parallelization overhead).

1.4. Idle regulation

The basic principle of list scheduling is a greedy method that keeps processors utilized if there are jobs ready to be executed. Scheduling strategies that use dynamic idle regulation approaches to improve system behavior have received some interest recently. Job partitioning to keep some processors idle was suggested in [23]. An analysis has shown its effectiveness for non-scalable workloads, possibly with high variance in arrival rate and execution. In [17], the design and implementation of a scheduling policy that prevents the degradation in performance due to scheduling idle processors too

early are presented. It keeps some processors idle during some periods of time before scheduling them. Exploiting unused time slots in list scheduling is also considered in [28].

In the work reported here, we emphasize on a simple policy based on list scheduling parameterized for the idle regulation. We consider the general framework introduced in [34], the (a, b, c)-*Scheme*, where the strategies are described by a set of parameters. The value of the first parameter a represents the number of processors allowed to be idle in the system before switching to the second phase, b represents the type of jobs (rigid, malleable, moldable, etc.), and c represents the scheduling strategy of the second phase (list, LPT, first fit, backfill, etc). It is important to note that the optimal settings for these parameters are very site specific and depend on the workload, resources, and other environmental factors. In real parallel systems, the value of the parameter a can be selected in accordance with the current system load, and provided by the scheduler. It depends on the penalty factor of the applications, and, based on measuring runtime workload characteristics, can adapt to the change of quality of the workload. We show that the variation of parameter a allows us to find a trade-off between avoiding idle processors by starting the parallelization sooner when more jobs are parallelized (with a larger overhead), and delaying their parallelization (causing processors to be left idle) until a smaller number of jobs is available. We also show that an improvement of known worst-case performance bounds can be achieved by bounding the maximum number of processors that can remain idle.

Several models of the workload with availability of information about jobs' execution performance are known [8]. However, it is clear that the choice of appropriate policies must be guided at least in part by the behavior of the actual parallel workload that is diverse and can be changed at runtime. The (a, b, c)-*Scheme* uses the idea of tuning the system parameter a at runtime to optimize the system behavior, by measuring workload characteristics to estimate the penalty factors. It can be done by post-mortem analysis of job characteristics during a certain time interval. The runtime measuring of workload characteristics to improve job scheduling has received considerable attention in recent years. In [22], on-line measuring of a job's execution parameters and speedup in making production scheduler decisions was discussed. In [27], the use of runtime measurements to improve job scheduling on a parallel machine with emphasis on gang scheduling-based strategies was studied.

1.5. Processor allocation regulation

The number of processors chosen by the user for job execution is typically based on the nature of the job or user desire, and does not take into account the workload characteristics of the multiprocessor system. This number is usually suitable for light load conditions, but it leads to unacceptably low system performance at heavy load [14]. Since users cannot practically know the system load, one possible way to optimize system behavior is to support different solutions and adapt to them at runtime.

The idea of allocating fewer and fewer processors to each job under heavy load, thus increasing throughput, was proposed in [15]. Maximizing application speedup through runtime selection of an appropriate number of allocated processors was discussed in [21], where the use of a runtime system that dynamically measures job efficiency at different allocations and automatically adjusts a job's processor allocation to maximize its speedup was proposed. While in [20], the study of the dynamic scheduler that uses runtime idleness information to dynamically adjust processor allocations to improve shared memory system utilization was presented. A self-adaptive scheduling strategy in a master–worker paradigm that dynamically adjusts the number of processors based on performance measures gathered during job execution was considered in [16]. In [8], it was shown that, of several strategies with equivalent turnaround times, the strategy that reduces allocation when load is high yields the lowest slowdowns. A general virtualization technique that simulates a virtual machine of p processors on $p' < p$ processors and allows the execution of a parallel job that requests p processors while only p' processors are allotted to it can be found in [25]. The technique yields good results for different network topologies. The two-phase strategy for the processor allocation regulation in the main frame of gang scheduling was introduced in [24]. A number of gang scheduling policies for parallel and distributed systems with different resource allocation scheme have been proposed [10,12,9,18,29,33].

1.6. Outline

The reminder of the paper is organized as follows: in Section 2, we present the notation and statement of the scheduling problem and describe the two-phase strategy for on-line parallel job scheduling with idle regulation. The analysis of the performance guarantee considering a specific job allocation to 1 and/or the whole m processors machine ($1 - m$ allocation policy) is presented in Section 3. The analysis of the case when jobs allocated to a fixed number of processors ($1-k$ allocation policy) is discussed in Section 4. Finally, conclusions and further research directions are discussed in the last section.

2. Preliminaries

We focus on the analysis of scheduling systems where all jobs arrive at time 0 and are processed into the same batch. A set of available and ready jobs will be executed up to the completion of the last one. All jobs that arrive during this time will be processed in the next batch. A relation between this scheme and the scheme where jobs are released over time, either

at their release time, or according to the precedence constraints is known, studied for different scheduling strategies for general or restricted cases, and leads to an additional factor of 2 [30].

2.1. Scheduling problem

We consider a set of independent parallel jobs with the objective of minimizing the total execution time (makespan) in the frame of the two-phase list scheduling strategy. The following problem is studied: given a parallel machine with m identical processors and a set of n independent parallel jobs $T = \{T_1, \ldots, T_n\}$ whose processing times $p_1, \ldots, p_n$ are not known until their completion. The number of processors k_i needed for the execution of job T_i may not be imposed (Section 3), or fixed and known as soon as it becomes available (Section 4). We assume that the job irrespective of k_i can be scheduled either on a single processor, on m processors (if k_i is not preset), and/or on requested k_i processors (if k_i is fixed). Real systems support preemptions, so we assume that the job can be preempted, redistributed and migrated if necessary. Unfortunately, it may incur large overheads. To minimize the number of preemptions we restrict our analysis to the case where the job can be preempted once, assigned to the required number of processors, then it cannot be preempted and/or run on a different set and/or different number of processors. Hence, the algorithms perform at most m preemptions.

A parallel job T_i is characterized by a triple $T_i = \{p_i, k_i, \mu_k^i\}$: namely, its execution time on a single processor (the total work done by the job), the number of requested processors k_i, and the penalty factor of its parallel execution on k_i processors. Its parallel execution time is $p_k^i = \mu_k^i p_i/k_i$. We assume that p_k^i depends non-increasingly on k_i, at least for a reasonable number of processors, and $p_i = p_1^i$. Let $\bar{p} = \max_{1\le i\le n}\{p_i\}$, $\bar{p}_\parallel = \max_{1\le i\le n}\{p_k^i\}$, $\underline{k} = \min_{1\le i\le n}\{k_i\}$, and $\bar{k} = \max_{1\le i\le n}\{k_i\}$.

Calculating the speedup S_k^i of the execution of job T_i on k processors as p_1^i/p_k^i, the penalty μ_k^i is considered as the ratio of the ideal speedup over the actual one $\mu_k^i = k/S_k^i$. We assume that the penalty factor μ_k^i of job T_i depends non-decreasingly on the number of allotted processors k: $\mu_k^i \le \mu_{k+1}^i$ for any $1 \le k < m$. If the job is allocated to one processor then $\mu_1^i = 1$. Let $\bar{\mu} = \max_{1\le i\le n}\{\mu_k^i\}$, and $\underline{\mu} = \min_{1\le i\le n}\{\mu_k^i\}$. According to a job monotonic property [36], assigning more processors to a job decreases its execution time at least up to a certain threshold, but increases its computational area,[2] hence

$$\frac{\underline{\mu}\bar{p}}{\bar{k}} \le \frac{\bar{\mu}\bar{p}}{\bar{k}} \le \bar{p}_\parallel \le \frac{\underline{\mu}\bar{p}}{\underline{k}}. \tag{1}$$

Let $W = \sum_{i=1}^n p_i$ be the total work of all jobs, $W_\parallel = \sum_{i=1}^n \mu_k^i p_i$ be the total work performed in the parallel execution, where

$$\underline{\mu} W \le W_\parallel \le \bar{\mu} W. \tag{2}$$

Let $C_\parallel^*$ denote the makespan of an optimal schedule, and C^* denote the makespan of the optimal schedule, when the parallelization of the jobs is not allowed. For a strategy A, let C_A be the makespan of the corresponding schedule. Note that

$$W/m, \quad \bar{p} \tag{3}$$

are lower bounds of C^*, and

$$W_\parallel/m, \quad \bar{p}_\parallel \tag{4}$$

are lower bounds of $C_\parallel^*$.

Assuming that the workload consists of at least m big jobs (with size $\bar{k}$ and processing time $\bar{p}_\parallel$), hence $W_\parallel \ge m\bar{p}_\parallel\bar{k}$, we obtain

$$C_\parallel^* \ge \bar{p}_\parallel\bar{k}. \tag{5}$$

From (1) we get

$$C_\parallel^* \ge \bar{p}_\parallel\bar{k} \ge \underline{\mu}\bar{p}. \tag{6}$$

All strategies will be analyzed according to their competitive ratios. Let $\rho_\parallel(I) = C_A(I)/C_\parallel^*(I)$ and $\rho(I) = C_A(I)/C^*(I)$ denote the competitive and sequential competitive ratios of algorithm A for an instance I. To measure the overall quality of A we define the performance of A by $\rho_\parallel^* = \sup\{C_A(I)/C_\parallel^*(I)\}$ and $\rho^* = \sup\{C_A(I)/C^*(I)\}$, respectively. The first one shows the worst case of the ratio between the makespan incurred by an algorithm A and the best-case makespan. The sequential competitive ratio will highlight the possible gain to allow the multiprocessing jobs compared to the classical approach of general list scheduling where jobs are purely sequential.

[2] Computational area is equivalent to the product of execution time and number of processors $k_i p_k^i = \mu_k^i p_i$.

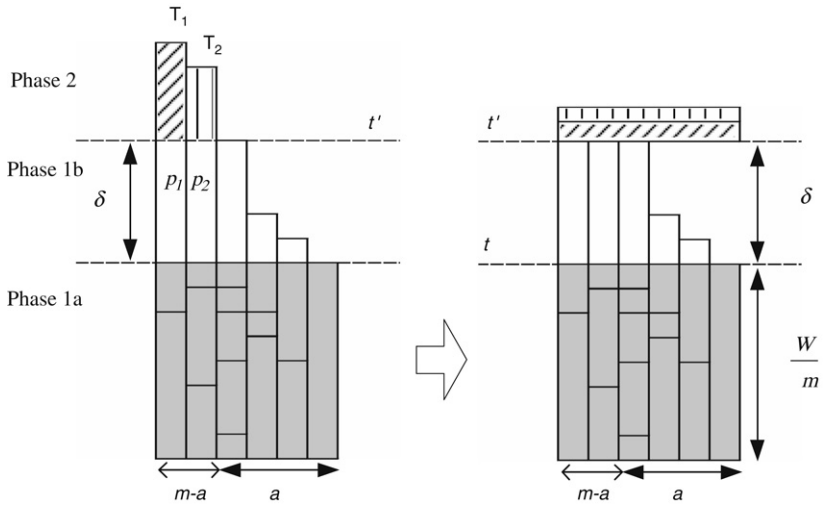

Fig. 1. The two-phase strategy with $a = 4$, $m = 6$.

2.2. The two-phase algorithm

We are interested in studying the influence of idle regulation on a general list scheduling based on a two-phase strategy. This strategy has been introduced in [24] and considers two successive phases. In the first phase, when the number of jobs is large, the strategy allocates processors sequentially without idle times and communications. When enough jobs have been completed, it switches to a second phase with multiprocessor execution. In the following analysis, we assume that in the first phase each processor has one job, each job is assigned to one processor, and thus, the processors' utilization is 1, and the schedule is optimal. Inefficiency appears when less than m jobs remain. In [24], another strategy is applied when the first processor becomes idle. In this paper, we generalize this approach by introducing an idle regulation mechanism. Hence, when the number of idle processors becomes larger than a, where $0 \leq a \leq m$, all remaining jobs are preempted and another strategy is applied to avoid too many idle processors. Fig. 1 illustrates this algorithm, when a special type of scheduling called *gang scheduling* is considered in the second phase ($1 - m$ allocation policy) (see Section 3). The use of parallel job scheduling in the second phase ($1 - k$ allocation policy) is illustrated in Fig. 6 (see Section 4).

The successive phases are shown: phase 1a, when all the processors are busy; phase 1b, when at least $m-a+1$ processors work (both phases use the well-known Graham's list strategy); and phase 2, when a or more processors become idle, and, hence, turn to a second strategy with parallel jobs. The parameter a is equal to the largest number of processors allowed to be idle in the system before switching to the second phase. It determines the balance between the number of jobs processed by the first and the second strategies. The sequential execution of jobs in the first phase under heavy load is a best strategy as there is no extra overhead. In the second phase, when the load is reduced and the number of idle processors becomes equal to or greater than a, the system changes strategy to avoid too many idle times. When such an idle regulation is used, there is a trade-off between starting parallelization sooner, when a is small (hence more jobs are parallelized causing larger parallelization overhead), and delaying their parallelization, when a is big (hence causing more processors to be left idle), until a smaller number of jobs is available. This scheme balances the needs of the user (job) with those of the computer system.

We assume that all parallel jobs can be executed sequentially irrespective of their type and the number of processors needed for the execution. It is a reasonable assumption for many distributed and shared memory applications, in which parallel processes can be executed sequentially (in particular, message passing processes executed by one processor, or a load balancing scheme based on common pool of jobs easily adapted to the processors' number). It is suitable even for rigid jobs that cannot cope with the reducing or increasing the number of allotted processors. Rigid jobs can be executed efficiently on one processor because of exploiting one processor communications locality.

But the assumption excludes from our consideration some class of parallel jobs, for example, synchronous shared memory applications, where parallelism cannot be transformed efficiently to the serial execution.

3. Analysis of $1 - m$ allocation policy

In this section, we provide an analysis of the two-phase algorithm with idle regulation for the case $0 \leq a \leq m$ considering a specific allocation of parallel jobs to 1 and m processors. Before deriving the general bound, we review some known results for the restricted cases of $a = 0$ and $a = 1$ (studied in [24]). The case $a = 0$ corresponds to the strategy that starts from the second phase with an allocation to m processors. The case $a = m$ corresponds simply to list scheduling of sequential jobs (only first phase is applied).

$a = 0$.

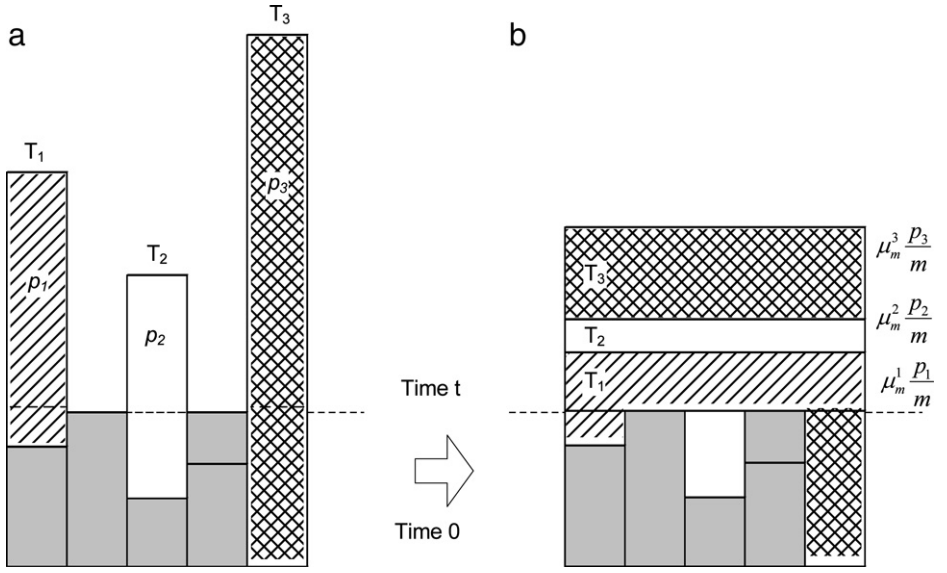

Fig. 2. The two-phase strategy with $a = 1$ and $m = 5$.

The performance guarantees of any list algorithm for scheduling independent parallel jobs on m processors can be found in [24]. It is shown that $\rho_{\parallel}^{*} \le \bar{\mu}/\underline{\mu}$ and $\rho^{*} \le \bar{\mu}$. If all jobs in the optimal solution are allocated to m processors ($\underline{\mu} = \bar{\mu}$) then $\rho_{\parallel}^{*} = 1$, and if they are allocated to one processor (by our assumption $\underline{\mu} = 1$) then $\rho_{\parallel}^{*} = \rho^{*}$.

Lemma 1. *The performance guarantees of any list algorithm for scheduling independent jobs on one processor are bounded as:* $\rho^{*} \le 2 - \frac{1}{m}$, $\rho_{\parallel}^{*} \le \frac{1}{\underline{\mu}} + \frac{\bar{k}}{\underline{\mu}}(1 - \frac{1}{m})$, *and* $\rho_{\parallel}^{*} \le \frac{1}{\underline{\mu}}(2 - \frac{1}{m})$ *in the case of large workload.*

Proof. The proof of the first bound is straightforward and appears to rely mainly on the Graham type arguments. An *idle interval* occurs when a processor does not process a job. Recall that in a greedy schedule, an idle interval can only occur if no tasks are available for processing. Let W° denote the sum of lengths of idle intervals in a schedule within the time span 0 (start time of the schedule) and C (end of the schedule). With $W^{\circ} \le (m-1)\bar{p}$ [37], $C = \frac{W+W^{\circ}}{m} \le \frac{W+(m-1)\bar{p}}{m}$ that is similar to Brent's lemma [5], $\rho_{\parallel}^{*} \le \frac{W}{mC_{\parallel}^{*}} + \frac{\bar{p}}{C_{\parallel}^{*}}(1 - \frac{1}{m})$ and $\rho_{\parallel}^{*} \le \frac{W_{\parallel}}{\underline{\mu} m C_{\parallel}^{*}} + \frac{\bar{p}}{C_{\parallel}^{*}}(1 - \frac{1}{m})$.

From (3), (4) and (1), it follows that $\rho^{*} \le 2 - \frac{1}{m}$ and $\rho_{\parallel}^{*} \le \frac{1}{\underline{\mu}} + \frac{\bar{k}}{\underline{\mu}}(1 - \frac{1}{m})$. And following assumption (6), we get $\rho_{\parallel}^{*} \le \frac{1}{\underline{\mu}}(2 - \frac{1}{m})$, which proves the lemma. □

Remark. If all the jobs in the optimal solution are allocated to m processors ($\underline{\mu} = \bar{\mu}$, $\bar{k} = m$) then $\rho_{\parallel}^{*} \le m/\bar{\mu}$, that corresponds to the bound in [24], and if they are allocated to one processor then $\rho_{\parallel}^{*} = \rho^{*}$.

$a = 1$.

Let us assume now that the strategy switches from the *Graham's* strategy to *Gang* when a first processor becomes idle, so $a = 1$ (Fig. 2). This case has been studied in [24]. It uses a combination of two extreme scheduling strategies *Graham* and *Gang*, a combination of allocation of one processor per job and assigning the whole machine to a job (coordinated strategy). In the first phase, when $n \ge m$, each job is processed on one processor (Fig. 2a). At time t, less than m jobs remain. In order to avoid idle processors the remaining jobs are preempted and assigned to m processors according to *Gang* strategy (Fig. 2b). The performance guarantees are bounded as: $\rho_{\parallel}^{*} \le \frac{\bar{\mu}}{\underline{\mu}}(1 - \frac{1}{m}) + \frac{1}{\underline{\mu} m}$ and $\rho^{*} \le \bar{\mu}\left(1 - \frac{1}{m}\right) + \frac{1}{m}$ (see Theorem 1, $a = 1$).

3.1. Idle regulation with $a > 1$

Let us assume now that the strategy switches from the *Graham's* strategy to *Gang* when a or more processors become idle.

Theorem 1. *Given a set of n independent jobs each with a penalty factor, from* $\underline{\mu}$ *to* $\bar{\mu}$, *allocated to one and/or to m processors, the performance guarantees of the two-phase strategy with idle regulation by the parameter a can be estimated as:*

$$\rho_{\parallel}^{*} = \max\{\rho_{\parallel}^{*1}, \rho_{\parallel}^{*2}\} \quad \textit{and} \quad \rho^{*} = \max\{\rho^{*1}, \rho^{*2}\}, \quad \textit{where}$$

$$\rho_{\parallel}^{*1} \le \frac{1}{\underline{\mu}}\left(1 + \frac{a-1}{m}\right), \qquad \rho_{\parallel}^{*2} \le \frac{\bar{\mu}}{\underline{\mu}}\left(1 - \frac{a}{m}\right) + \frac{a}{\underline{\mu} m},$$

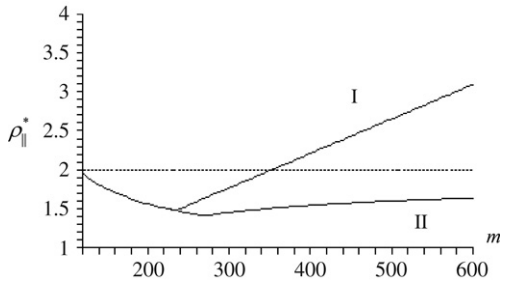

Fig. 3. $\rho_{\|}^{*}$, varying m; $a = 30$, (I) logarithmic speedup function and (II) linear $m/2$ speedup.

$$\rho^{*1} \leq 1 + \frac{a-1}{m}, \qquad \rho^{*2} \leq \bar{\mu}\left(1 - \frac{a}{m}\right) + \frac{a}{m}.$$

Furthermore, $\rho_{\|}^{*1}$ and ρ^{*1} are tight in the case of $m \leq \frac{a\bar{\mu}-1}{\bar{\mu}-1}$, $\rho_{\|}^{*2}$ and ρ_{2}^{*} are tight in the case of $m > \frac{a\bar{\mu}-1}{\bar{\mu}-1}$.

Proof. The following are the phases of the algorithm (Fig. 2):

1a. $[0,\ t)$ All processors are busy.
1b. $[t,\ t')$ At least $m - a + 1$ processors work.
2. $[t',\ t'')$ a or more processors become idle, and *Gang* strategy is applied.

The work done in phase 1a is denoted by $W^1 = m \cdot t$. From time t onwards, there remain less than m unfinished jobs. Each job not finished until t' is executed necessarily in phase 2. Then, assuming that part of these h jobs have not been processed in phase 1a or 1b the remaining work is equal to $W^2 = \sum_{i=1}^{h}\left(p_i^{\text{rem}} - \delta\right) \leq h\,(\bar{p} - \delta)$, where p_i^{rem} is the remaining work for the job i, $h \leq m - a$, and $\delta = t' - t$.

The completion time is $C \leq \frac{1}{m}W^1 + \delta + \bar{\mu}\frac{1}{m}W^2$. Two lower bounds of C^* are δ and $\frac{W^1+(m-a+1)\delta+W^2}{m}$. Hence $\frac{W^1}{m} \leq C^* - \frac{(m-a+1)}{m}\delta - \frac{1}{m}W^2$ and

$$C \leq C^* - \frac{m-a+1}{m}\delta - \frac{1}{m}W^2 + \delta + \bar{\mu}\frac{1}{m}W^2.$$

Assuming h jobs executed after t, it follows that $C \leq C^* - \frac{m-a+1}{m}\delta + \frac{\bar{\mu}-1}{m}\left(\bar{p} - \delta\right)h + \delta = C^* + \left(\frac{a-1}{m} - \frac{\bar{\mu}-1}{m}h\right)\delta + \frac{\bar{\mu}-1}{m}h\bar{p}$.

Let $B\,(a) = \frac{a-1}{m}$, $A\,(h) = \frac{\bar{\mu}-1}{m}h$, and $J\,(a, h) = (B\,(a) - A\,(h))\,\delta + A\,(h)\,\bar{p}$. We consider two different possibilities for values of $A\,(h)$ and $B\,(a)$.

I. If $A\,(h) \geq B\,(a)$ then $J\,(a, h) \leq A\,(h)\,\bar{p}$. Hence $C \leq C^* + \frac{(\bar{\mu}-1)h}{m}\bar{p}$, $C \leq \frac{C_{\|}^*}{\underline{\mu}} + \frac{h}{m}\left(\frac{\bar{\mu}}{\underline{\mu}} - \frac{1}{\underline{\mu}}\right)\bar{p}_{\|}\bar{k}$, and $\rho^* \leq 1 + \frac{(\bar{\mu}-1)(m-a)}{m} = \bar{\mu}\left(1 - \frac{a}{m}\right) + \frac{a}{m}$. From (5), $\rho_{\|}^* \leq \frac{\bar{\mu}}{\underline{\mu}}\left(1 - \frac{a}{m}\right) + \frac{a}{\underline{\mu}m}$. $A\,(h) \geq B\,(a)$ when $\frac{a-1}{m} - \frac{\bar{\mu}-1}{m}h \leq 0$, hence $h \geq \frac{a-1}{\bar{\mu}-1}$. For $h \leq m - a$, we obtain $m - a \geq \frac{a-1}{\bar{\mu}-1}$ and $m \geq \frac{a\bar{\mu}-1}{\bar{\mu}-1}$.

II. If $A\,(h) < B\,(a)$ then $J\,(a, h) \leq (B\,(a) - A\,(h))\,\bar{p} + A\,(h)\,\bar{p} \leq B\,(a)\,\bar{p}$. Hence $C \leq C^* + \frac{a-1}{m}\bar{p}$ and $C \leq \frac{C_{\|}^*}{\underline{\mu}} + \frac{(a-1)\bar{k}}{\underline{\mu}m}\bar{p}_{\|}$. It follows that $\rho^* \leq 1 + \frac{a-1}{m}$ and $\rho_{\|}^* \leq \frac{1}{\underline{\mu}} + \frac{a-1}{\underline{\mu}m}$. $A\,(h) < B\,(a)$ when $\frac{a-1}{m} - \frac{\bar{\mu}-1}{m}h > 0$, hence $h < \frac{a-1}{\bar{\mu}-1}$. For $h = m - a$, $m - a < \frac{a-1}{\bar{\mu}-1}$ and $m < \frac{a\bar{\mu}-1}{\bar{\mu}-1}$. □

Remark. For $a = 0$, $\rho^* \leq \bar{\mu}$ and $\rho_{\|}^* \leq \bar{\mu}/\underline{\mu}$ correspond to performance guarantees of any list algorithm for scheduling jobs allocated to m processors (see Section 3.1). For $a = 1$, $\rho^* \leq \bar{\mu}\left(1 - \frac{1}{m}\right) + \frac{1}{m}$ and $\rho_{\|}^* \leq \frac{\bar{\mu}}{\underline{\mu}}\left(1 - \frac{1}{m}\right) + \frac{1}{m\underline{\mu}}$, which are equal to the bounds of [24]. For $\bar{\mu} = 1$, $\rho^* = \rho_{\|}^* = 1$, the jobs are parallelized without overhead. The schedule has no idle intervals and is optimal. For $a = m$, $\rho^* \leq 2 - 1/m$, only the *Graham* strategy is applied, and $\rho_{\|}^* \leq \frac{1}{\underline{\mu}}(2 - 1/m)$.

Fig. 3 shows the competitive ratio $\rho_{\|}^*$, varying the number of processors, fixed a, logarithmic shape of a job speedup function of m (linear penalty factor), and linear $m/2$ shape of a job speedup function of m (constant penalty factor of 2).

Fig. 4 presents the performance, varying the parameter a, and fixed number of processors. Both figures show that tuning the idle times by the parameter a allows us to get the minimum of the worst-case error bound for two different workload characteristics.

4. Analysis for $1 - k$ allocation policy (rigid jobs)

We provide now an analysis of the two-phase algorithm with allocation to 1 and/or to the fixed number of processors, from $\underline{k}$ to $\bar{k}$, with idle regulation by $0 \leq a \leq m$. We study the performance with regard to the number of processors allotted

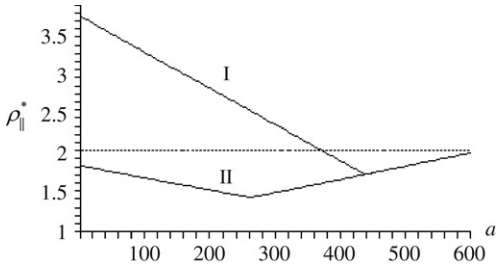

Fig. 4. $\rho^*_\parallel$, varying a, $m = 600$; (I) logarithmic speedup function and (II) linear $m/2$ speedup.

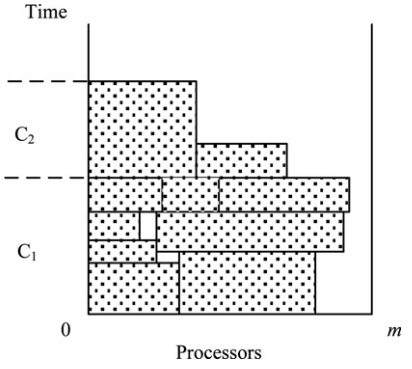

Fig. 5. Scheduling rigid jobs in space-sharing mode.

to a job for its execution and its penalty factor. Before deriving a general bound, we review known results of the restricted case with $a = 0$ that corresponds to the strategy that starts from the second phase (scheduling rigid jobs by a list algorithm). The case $a = m$ corresponds to list scheduling for sequential jobs (only the first phase is applied).

$a = 0$.

Before going into the details of the two-phase algorithm we make some general remarks well known in the literature.

Lemma 2. *The performance guarantees of any list algorithm for scheduling independent parallel rigid jobs are bounded as:*

$$\rho^*_\parallel \le \frac{2m - \bar{k}}{m - \bar{k} + 1} \quad \textit{and} \quad \rho^* \le \frac{1}{m - \bar{k} + 1}\left(\bar{\mu}m + \frac{\underline{\mu}}{\underline{k}}(m - \bar{k})\right).$$

Proof. In the schedule, there are two kinds of time slots which can be combined conceptually into two successive intervals, namely C_1 and C_2 (Fig. 5) [19,35]. Both intervals correspond respectively to the time slots when at most $\bar{k} - 1$ processors are idle, and strictly more than $\bar{k} - 1$ processors are idle.

Let $W_\parallel$ be a total work performed. Noticing that C_2 is limited by the maximum job execution time $\bar{p}_\parallel$ (or the length of the critical path in the case of precedence constrains [19,35]), and $W_\parallel \ge (m - \bar{k} + 1)C_1 + C_2$, we obtain an upper bound of the total completion time

$$C = C_1 + C_2 \le \frac{W_\parallel - C_2}{m - \bar{k} + 1} + C_2 \le \frac{W_\parallel + (m - \bar{k})\bar{p}_\parallel}{m - \bar{k} + 1}.$$

From (4), it follows that $\rho^*_\parallel \le \frac{2m-\bar{k}}{m-\bar{k}+1}$. Recalling (2), (3) and (1), we obtain

$$\rho^* \le \frac{\bar{\mu}m}{m - \bar{k} + 1} + \frac{\underline{\mu}(m - \bar{k})}{\underline{k}(m - \bar{k} + 1)} = \frac{1}{m - \bar{k} + 1}\left(\bar{\mu}m + \frac{\underline{\mu}}{\underline{k}}(m - \bar{k})\right). \quad \square$$

Remark. The result is a generalization of the well-known Graham bound. For sequential jobs we obtain the bound $2 - 1/m$ (no penalty: $\underline{\mu} = \bar{\mu} = 1$, and $\underline{k} = \bar{k} = 1$) and both competitive ratios coincide.

Lemma 3. *The sequential performance guarantee of any list algorithm for scheduling independent parallel and sequential jobs is bounded as:* $\rho^* \le \bar{\mu}\frac{2m-\bar{k}}{m-\bar{k}+1}$.

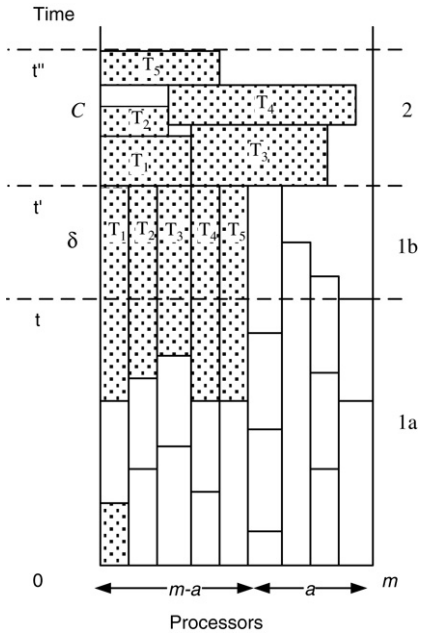

Fig. 6. The two-phase strategy for $a = 4$ and $m = 9$.

Proof. The proof follows directly from Lemma 2 and the assumption that $\underline{k} = 1$.

$$\rho^* \le \frac{1}{m-\bar{k}+1}\left(\bar{\mu}m + \frac{\underline{\mu}}{\underline{k}}(m-\bar{k})\right) \le \bar{\mu}\frac{2m-\bar{k}}{m-\bar{k}+1}. \quad \square$$

4.1. Idle regulation with $a > 0$

Let us turn now to the case in which the strategy switches from the *Graham's* strategy to parallel job scheduling when a processors become idle (Fig. 6).

Theorem 2 (*Sequential Competitivity [32]*). *Given a set of n independent parallel jobs with variation of the penalty factors, from $\underline{\mu}$ to $\bar{\mu}$, allocated to fixed number of processors, from $\underline{k}$ to $\bar{k}$, the sequential performance guarantee of the two-phase strategy can be estimated as: $\rho^* = \max\{\rho^{*1}, \rho^{*2}\}$, with $\rho^{*1} \le 1 + \frac{a-1}{m}$ and $\rho^{*2} \le \frac{a}{m} + \frac{1}{m-\bar{k}+1}(\bar{\mu}(m-a) + \frac{\underline{\mu}}{\underline{k}}(m-\bar{k}))$.*

Proof. Let us denote by W^1 the work executed until t. Each job not finished until t' is executed necessarily in phase 2. Then, assuming that part of these h jobs have not been processed in phase 1a or 1b, the remaining work is $W^2 = \sum_{i=1}^{h}(p_i^{\text{rem}} - \delta) \le h(\bar{p} - \delta)$, where $h \le m - a$, and $\delta = t' - t$. Let $\bar{p}_{\|}^{\text{rem}} = \max_{1\le i\le n}\{\frac{\mu_{\underline{k}}^i(p_i^{\text{rem}}-\delta)}{k_i}\}$. Following Lemma 2 for the completion time of phase 2, the upper bound of the total completion time C is $\frac{W^1}{m} + \delta + \frac{\bar{\mu}W^2 + (m-\bar{k})\bar{p}_{\|}^{\text{rem}}}{m-\bar{k}+1}$. We know that $W \ge W^1 + (m-a+1)\delta + W^2$, hence

$$C \le \frac{W}{m} - \frac{m-a+1}{m}\delta + \delta + W^2\left(\frac{\bar{\mu}}{m-\bar{k}+1} - \frac{1}{m}\right) + \frac{(m-\bar{k})\bar{p}_{\|}^{\text{rem}}}{m-\bar{k}+1}.$$

Considering h jobs executed after t', and that $\bar{p}_{\|}^{\text{rem}} \le \frac{\underline{\mu}(\bar{p}-\delta)}{\underline{k}}$, it follows that

$$C \le \frac{W}{m} + \left(\frac{a-1}{m} - \left(\frac{h\bar{\mu}}{m-\bar{k}+1} + \frac{(m-\bar{k})\underline{\mu}}{\underline{k}(m-\bar{k}+1)} - \frac{h}{m}\right)\right)\delta + \left(\frac{h\bar{\mu}}{m-\bar{k}+1} + \frac{(m-\bar{k})\underline{\mu}}{\underline{k}(m-\bar{k}+1)} - \frac{h}{m}\right)\bar{p}.$$

Let $B(a) = \frac{a-1}{m}$, $A(h) = \frac{h\bar{\mu}}{m-\bar{k}+1} + \frac{(m-\bar{k})\underline{\mu}}{\underline{k}(m-\bar{k}+1)} - \frac{h}{m}$, and $J(a,h) = (B(a) - A(h))\delta + A(h)\bar{p}$. Hence, $C \le \frac{W}{m} + J(a,h)$.

There are two different possibilities for values of $A(h)$ and $B(a)$.

I. If $A(h) \ge B(a)$ then $J(a,h) \le A(h)\bar{p}$, and

$$C \le \frac{W}{m} + \left(\frac{h\bar{\mu}}{m-\bar{k}+1} + \frac{(m-\bar{k})\underline{\mu}}{\underline{k}(m-\bar{k}+1)} - \frac{h}{m}\right)\bar{p}.$$

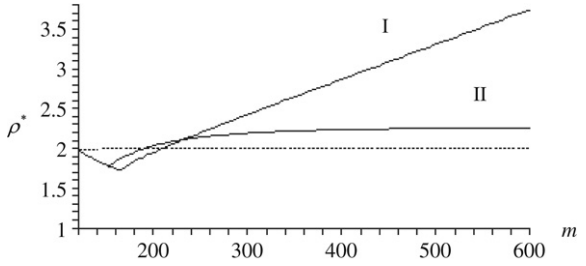

Fig. 7. ρ^*, varying the number of processors, $a = 120$, $\bar{k} = 60$, $\underline{k} = \bar{k}/30$. (I) linear $\bar{\mu}$, and (II) constant $\bar{\mu}$.

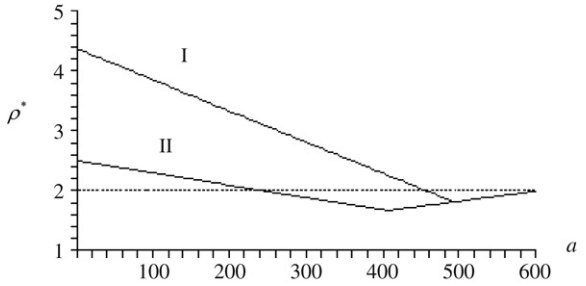

Fig. 8. ρ^*, varying a, $m = 600$, $\bar{k} = 60$, $\underline{k} = \bar{k}/30$, (I) linear $\bar{\mu}$ and (II) constant $\bar{\mu}$.

From (3), $\rho^* \leq \frac{a}{m} + \frac{1}{m-\bar{k}+1}(\bar{\mu}(m-a) + \frac{\bar{\mu}}{\underline{k}}(m-\bar{k}))$.

II. If $A(h) < B(a)$ then $J(a,h) \leq (B(a) - A(h))\,\bar{p} + A(h)\,\bar{p} \leq B(a)\,\bar{p}$, and $C \leq \frac{W}{m} + \frac{a-1}{m}\bar{p}$, hence $\rho^* \leq 1 + \frac{a-1}{m}$. □

Remark. For $a = 0$, $\rho^* \leq \frac{1}{m-\bar{k}+1}(\bar{\mu}m + \frac{\bar{\mu}}{\underline{k}}(m-\bar{k}))$ and corresponds to a sequential performance guarantee of any list algorithm for scheduling parallel jobs (Lemma 3). For $a = m$, and $\bar{k} = \underline{k} = 1$, $\rho^* \leq 2 - 1/m$, only the *Graham's* strategy (phase 1) is applied.

Fig. 7 shows the sequential competitive ratio ρ^*, varying the number of processors, fixed a, $\bar{k}$, $\underline{k}$, linear $\bar{\mu}$ (logarithmic shape of a job speedup function of m), and constant $\bar{\mu}$ (linear $m/2$ shape of a job speedup function of m). We assume here that the number of unfinished jobs in phase 2 is $m - a$. Fig. 8 presents the performance, varying the parameter a, fixed number of processors, $\bar{k}$, $\underline{k}$, linear and constant $\bar{\mu}$. Both figures show that the minimum of the worst-case error bound can be achieved by tuning the idle times.

Theorem 3 (*Sequential Competitivity for a Job Mix*). *Given a set of n independent parallel and sequential jobs with variation of the penalty factors, from* 1 *to* $\bar{\mu}$, *allocated to fixed number of processors, from* 1 *to* $\bar{k}$, *the sequential performance guarantee of the two-phase strategy can be estimated as:*

$$\rho^* \leq \bar{\mu}\frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{\bar{\mu}}{m-\bar{k}+1} - \frac{1}{m}\right).$$

Proof. The proof follows directly from Theorem 2 and the assumption that $\underline{k} = 1$. We also note for any m, $\bar{k}$, $\bar{\mu}$, $\underline{k}$, $A(h) - B(a) = \frac{1}{m-\bar{k}+1}(h\bar{\mu} + \frac{(m-\bar{k})\underline{\mu}}{\underline{k}}) - 1 + \frac{1}{m} \geq 0$, so that $A(h) \geq B(a)$. Hence $\rho^* \leq \frac{a}{m} + \frac{1}{m-\bar{k}+1}(\bar{\mu}(m-a) + \frac{\bar{\mu}}{\underline{k}}(m-\bar{k}))$ and

$$\rho^* \leq \bar{\mu}\frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{\bar{\mu}}{m-\bar{k}+1} - \frac{1}{m}\right). \quad \square$$

Theorem 4. *Given a set of n independent parallel jobs allocated to a fixed number of processors, from* $\underline{k}$ *to* $\bar{k}$, *with minimum penalty factor* $\underline{\mu}$, *the performance guarantee of the two-phase strategy can be estimated as:*

$$\rho^*_{\parallel} = \max\{\rho^{*1}_{\parallel}, \rho^{*2}_{\parallel}\}, \quad \text{with } \rho^{*1}_{\parallel} \leq 1 + \frac{a-1}{m} \quad \text{and} \quad \rho^{*2}_{\parallel} \leq \frac{(2m-\bar{k})}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{\underline{\mu}m}\right).$$

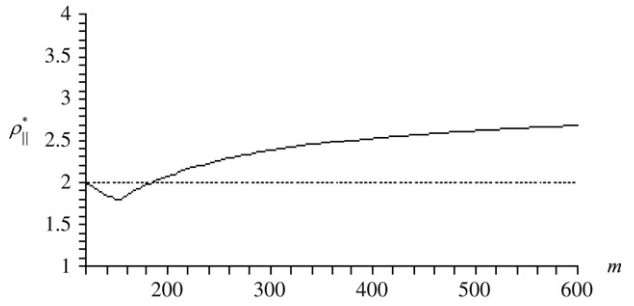

Fig. 9. $\rho^*_{\|}$, varying the number of processors, $a = 120$, $\bar{k} = m/2$, and constant $\underline{\mu}$.

Proof. As shown in Theorem 2, the completion time of the schedule is $C \le \frac{W^1}{m} + \delta + \frac{W^2_{\|} + (m-\bar{k})\bar{p}^{\text{rem}}_{\|}}{m-\bar{k}+1}$. Note that $W_{\|} \ge \underline{\mu} W^1 + (m - a + 1)\delta\underline{\mu} + W^2_{\|}$.

It follows $C \le \frac{W_{\|}}{\underline{\mu} m} + \frac{a-1}{m}\delta + \frac{W^2_{\|}}{m-\bar{k}+1} - \frac{W^2_{\|}}{\underline{\mu} m} + \frac{(m-\bar{k})\bar{p}^{\text{rem}}_{\|}}{m-\bar{k}+1}$.

Note that $W^2 \le h\,\bar{k}\bar{p}^{\text{rem}}_{\|}$ and $\bar{p}^{\text{rem}}_{\|} \le \frac{\underline{\mu}(\bar{\bar{p}}-\delta)}{\underline{k}}$. Let $\bar{\bar{p}}_{\|} = \frac{\underline{\mu}\bar{p}}{\underline{k}}$, $\bar{p}^{\text{rem}}_{\|} \le \bar{\bar{p}}_{\|} - \frac{\underline{\mu}}{\underline{k}}\delta$, it follows

$$C \le \frac{W_{\|}}{\underline{\mu} m} + \frac{a-1}{m}\delta + \left(\frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}\right)\bar{p}^{\text{rem}}_{\|},$$

$$C \le \frac{W_{\|}}{\underline{\mu} m} + \frac{a-1}{m}\delta - \left(\frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}\right)\frac{\underline{\mu}}{\underline{k}}\delta + \left(\frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}\right)\bar{\bar{p}}_{\|}.$$

Let $B(a) = \frac{a-1}{m}$, $A(h) = \frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}$, and $J(a,h) = \left(B(a) - A(h)\frac{\underline{\mu}}{\underline{k}}\right)\delta + A(h)\bar{\bar{p}}_{\|}$. Hence $C \le \frac{W_{\|}}{\underline{\mu} m} + J(a,h)$.

We consider two different possibilities for values of $A(h)$ and $B(a)$.

I. If $A(h)\frac{\underline{\mu}}{\underline{k}} \ge B(a)$ then $J(a,h) \le A(h)\bar{\bar{p}}_{\|}$, and

$$C \le \frac{W_{\|}}{\underline{\mu} m} + \left(\frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}\right)\bar{\bar{p}}_{\|}.$$

Lower bounds for an optimal schedule are (4) and (6). Hence $C^*_{\|} \ge \underline{\mu}\bar{p} \ge \frac{\underline{\mu}}{\underline{k}}\bar{p} = \bar{\bar{p}}_{\|}$, and

$$\rho^*_{\|} \le \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{\underline{\mu} m}\right) \le \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{m}\right).$$

II. If $A(h)\frac{\underline{\mu}}{\underline{k}} < B(a)$ then $J(a,h) \le \left(B(a) - A(h)\frac{\underline{\mu}}{\underline{k}}\right)\bar{p} + A(h)\bar{\bar{p}}_{\|}$.

Note that $\bar{\bar{p}}_{\|} = \frac{\underline{\mu}\bar{p}}{\underline{k}}$, we get $J(a,h) \le B(a)\bar{p}$, and $C \le \frac{W_{\|}}{\underline{\mu} m} + \frac{a-1}{m}\bar{p} \le \frac{W_{\|}}{m} + \frac{a-1}{m}\bar{p}$.

From (6) and (4), we get $C^*_{\|} \ge \underline{\mu}\bar{p} \ge \bar{p}$ and $\rho^*_{\|} \le 1 + \frac{a-1}{m}$, which proves the theorem. □

Remark. For $a = 0$, $\rho^*_{\|} \le \frac{2m-\bar{k}}{m-\bar{k}+1}$ corresponds to the well-known performance guarantee of any list algorithm for scheduling parallel jobs. For $\bar{k} = 1$, $\rho^*_{\|} \le 2 - 1/m$, only the *Graham* strategy is applied.

Fig. 9 shows the competitive ratio $\rho^*_{\|}$, varying a number of processors, fixed a, $\bar{k} = m/2$, and constant $\underline{\mu}$ (linear $m/2$ shape of a job speedup function of m). We assume here that the number of unfinished jobs in phase 2 is $m - a$.

Fig. 10 presents the performance guarantee, varying the parameter a, fixed number of processors, $\bar{k} = m/2$, and constant $\underline{\mu}$. Changing the parameter a allows us to get the minimum of the worst-case error bound.

Theorem 5 (*Competitivity for a Job Mix [32]*). *Given a set of n independent parallel and sequential jobs allocated to a fixed number of processors, from 1 to $\bar{k}$, the performance guarantee of the two-phase strategy can be estimated as:*

$$\rho^*_{\|} \le \frac{(2m-\bar{k})}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{m}\right).$$

Proof. The proof follows directly from Theorem 4, and the assumption that $\underline{k} = \underline{\mu} = 1$. We also note that $A(h) \ge B(a)$ (see Theorem 3), hence

$$\rho^*_{\|} \le \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{\underline{\mu} m}\right) \le \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{m}\right). \quad \square$$

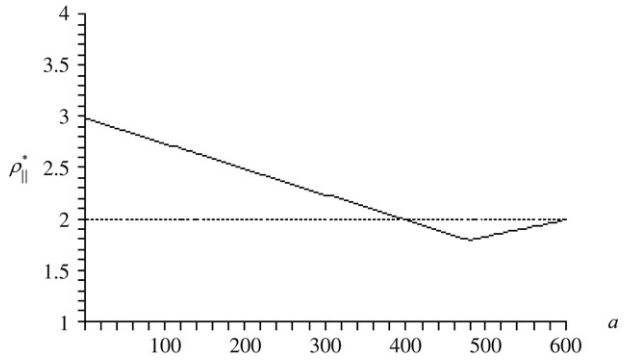

Fig. 10. $\rho^*_{\|}$, varying a, $m = 600$, $\bar{k} = m/2$, and constant $\underline{\mu}$.

Table 1
A summary of performance guarantees of the two-phase strategy for the minimization of the total completion time

a	Allocation	ρ^*	Strategy
0	*Gang*	$\rho^* = \bar{\mu}/\underline{\mu}$	Allocation to m processors
$1 \le a < m$	*1-m-Gang*	$\rho^*_{\|} = \max\{\rho^{*1}_{\|}, \rho^{*2}_{\|}\}$, where $\rho^{*1}_{\|} \le \frac{1}{\underline{\mu}}(1 + \frac{a-1}{m})$, $\rho^{*2}_{\|} \le \frac{\bar{\mu}}{\underline{\mu}}(1 - \frac{a}{m}) + \frac{a}{\underline{\mu}m}$	Allocation to 1 and/or to m processors with idle regulation by a
0	Rigid	$\rho^*_{\|} = \frac{2m-\bar{k}}{m-\bar{k}+1}$	Allocation to k (from $\underline{k}$ to $\bar{k}$) processors
$1 \le a < m$	1-k-Rigid	$\rho^*_{\|} = \max\{\rho^{*1}_{\|}, \rho^{*2}_{\|}\}$, where $\rho^{*1}_{\|} \le 1 + \frac{a-1}{m}$, $\rho^{*2}_{\|} \le \frac{(2m-\bar{k})}{m-\bar{k}+1} - a(\frac{1}{m-\bar{k}+1} - \frac{1}{\underline{\mu}m})$	Allocation to 1 and/or to k (from $\underline{k}$ to $\bar{k}$) processors with idle regulation by a
$1 \le a < m$	1-k-Rigid & serial	$\rho^*_{\|} \le \frac{(2m-\bar{k})}{m-\bar{k}+1} - a(\frac{1}{m-\bar{k}+1} - \frac{1}{m})$	Allocation to 1and/or to k (from 1 to $\bar{k}$) processors with idle regulation by a

5. Conclusions

We have focused on non-clairvoyant scheduling of parallel jobs with emphasis on the regulation of idle periods in the context of general list policies. Adaptive schemes that are capable of dynamically changing the scheduling algorithm during the execution to optimize the global system behavior are proposed. Algorithms that have no knowledge about jobs other than the number of unfinished jobs in the system and their processor requirements are considered. Batch scheduling under a two-phase generic framework is used. It successively combines sequential and parallel job execution in space-sharing mode. A framework extension with capability of idle regulation is introduced.

The main contribution is towards solving the problem of dynamic scheduling of idle intervals on parallel machines by introducing a model that captures some important aspects of the practical scheduling problem. Jobs that are not fully parallelizable are included. The model of parallel job based on a penalty factor is applied to reduce the complexity of real parallel systems and simplify their theoretical study. We have generalized the known worst-case performance bounds by considering two extra parameters: job parallelization penalty and idle regulation, in addition to the number of processors and maximum processor requirements considered in the literature. We have showed that tuning the parameter for regulating idle times improves scheduling performance guarantees. A trade-off between avoiding idle processors by starting parallelization sooner when more tasks are parallelized (increasing total overhead), and delaying their parallelization (causing processors to be left idle) until a smaller number of jobs is available can be found. The parameter a can be calculated based on runtime measuring workload characteristics, and adapted to the change of the workload quality on-line. A summary of the results can be found in Table 1. It shows that within the proposed model it is possible to design good parallel job scheduling approximation algorithms whose qualitative behavior can be estimated. Performance guarantees of job scheduling in space-sharing mode can be improved under certain conditions of idle regulation.

Several problems remain open: firstly, theoretical and experimental analysis of the idle regulation with more variations of job scheduling strategies (largest job first, backfill, etc.), and optimization criteria, both system centric (utilization, throughput, etc.) and user centric (waiting time, turnaround time, etc.). Secondly, the analysis of the system in a practical scheduling environment that supports dependent jobs, and jobs that can arrive at any moment. Also of interest is the application of the proposed scheme to the set of malleable jobs.

Acknowledgements

The authors would like to thank the anonymous referees whose valuable remarks and comments helped to improve the paper. This work is partly supported by CONACYT (Consejo Nacional de Ciencia y Tecnología de México) under grant no. 48385.

References

[1] J. Błażewicz, M Drozdowski, K. Ecker, Management of resources in parallel systems, in: J. Bazewicz, D. Trystram, B. Plateau (Eds.), Handbook on Parallel and Distributed Processing, Springer Verlag, 2000, pp. 263–341.
[2] E. Blayo, L. Debreu, G. Mounie, D. Trystram, Dynamic load balancing for adaptive mesh ocean circulation model, Engineering Simulation 22 (2) (2000) 8–23.
[3] J. Blazewicz, K Ecker, E. Pesch, G. Schmidt, J. Weglarz, Scheduling Computer and Manufacturing Processes, Springer Verlag, Berlin, New York, 2001.
[4] V. Bharadwaj, D Ghose, V. Mani, T. Robertazzi, Scheduling Divisible Loads in Parallel and Distributed Systems, IEEE Computer Society Press, Los Alamos, USA, 1996.
[5] R. Brent, in: J.F. Traub. (Ed.), The Parallel Evaluation of Arithmetic Expressions in Logarithmic Time Complexity of Sequential and Parallel Numerical Algorithms, Academic Press, New York, 1973, pp. 83–102.
[6] S. Chapin, W. Cirne, D. Feitelson, J. Patton Jones, S.T. Leutenegger, U. Schwiegelshohn, W. Smith, D. Talby, Benchmarks and standards for the evaluation of parallel job Schedulers, in: D.G. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing, in: LNCS, vol. 1659, Springer-Verlag, 1999, pp. 66–89.
[7] T Decker, W Krandick, Parallel real root isolation using the Descartes method, in: Proceedings of the 6th High Performance Conference (HiPC99), in: LNCS, vol. 1745, Springer-Verlag, 1999, pp. 261–268.
[8] A Downey, A parallel workload model and its implications for processor allocation, in: Proc. the 6th International Symposium of High Performance Distributed Computing, 1997, pp. 112–123.
[9] D. Feitelson, M. Jette, Improved utilisation and responsiveness with gang scheduling, in: Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, vol. 1291, Springer-Verlag, Berlin, Germany, 1997, pp. 238–261.
[10] D. Feitelson, L. Rudolph, Parallel job scheduling: Issues and approaches, in: Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, vol. 949, Springer-Verlag, Berlin, Germany, 1995, pp. 1–18.
[11] D. Feitelson, L. Rudolph, Toward convergence in job schedulers for parallel supercomputers, in: D. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing, in: LNCS, vol. 1162, Springer-Verlag, 1996, pp. 1–26.
[12] D. Feitelson, L. Rudolph, Evaluation of design choices for gang scheduling using distributed hierarchical control, Journal of Parallel and Distributed Computing 35 (1996) 18–34.
[13] D. Feitelson, L. Rudolph, Metrics and benchmarking for parallel job scheduling, in: D.G. Feitelson, L. Rudolph (Eds.), JSSPP, IPPS/SPDP'98 Workshop, in: Proceedings, LNCS, vol. 1459, Orlando, Florida, USA, 1998, pp. 1–24.
[14] D. Feitelson, L. Rudolph, U. Schweigelshohn, K. Sevcik, P. Wong, Theory and practice in parallel job scheduling, in: D.G. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing, in: LNCS, vol. 1291, Springer-Verlag, 1997, pp. 1–34.
[15] D Ghosal, G. Serazii, S. Tripathi, The processor working set and its use in scheduling multiprocessor system, IEEE Transactions on Software Engineering 17 (5) (1991) 443–453.
[16] E. Heymann, M Senar, E. Luque, M. Livny, Self-adjusting scheduling of master-worker applications on distributed clusters, in: R. Sakellariou et al. (Eds.), Euro-Par 2001, in: LNCS, vol. 2150, Manchester, UK, 2001, pp. 742–751.
[17] S. Iyer, P. Druschel, Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O, in: Symposium on Operating Systems Principles, 2001, pp. 117–130.
[18] H. Karatza, A simulation-based performance analysis of gang scheduling in a distributed system', in: Proc. of the 32nd Annual Simulation Symp. (San Diego, CA, USA, April), IEEE Computer Society, Los Alamitos, CA, USA, 1999, pp. 26–33.
[19] E. Lloyd, Concurrent task systems, Operational Research 29–1 (1981) 189–201.
[20] C. McCann, R. Vaswani, J. Zahorjan, A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors, ACM Transactions on Computer System 11 (2) (1993) 146–178.
[21] T. Nguyen, R. Vaswani, J. Zahorjan, Parallel application characterization for multiprocessor scheduling policy design, in: D.G. Feitelson, L. Rudolph (Eds.), JSSPP, in: LNCS, vol. 1162, Springer-Verlag, 1996.
[22] T. Nguyen, R. Vaswani, J. Zahorjan, Using runtime measured workload characteristics in parallel processor scheduling, in: D.G. Feitelson, L. Rudolph (Eds.), JSSPP, in: LNCS, vol. 1162, Springer-Verlag, 1996.
[23] E. Rosti, E. Smirni, G. Serazzi, L. Dowdy, Analysis of non-work-conserving processor partitioning policies, in: D. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing, in: LNCS, vol. 949, Springer-Verlag, 1995, pp. 165–181.
[24] C. Rapine, I. Scherson, D. Trystram, On-line scheduling of parallelizable jobs, in: Proceedings of EUROPAR'98 Conference, in: LNCS, vol. 1470, Springer-Verlag, 1998, pp. 322–327.
[25] Sgall, A. Feldmann, M. Kao, S. Teng, Optimal online scheduling of parallel jobs with dependencies, Journal of Combinatorial Optimization 1 (4) (1998) 393–411.
[26] Sgall, On-line scheduling-A survey, in: A. Fiat, G.J. Woeginger (Eds.), Online Algorithms: The State of the Art, in: LNCS, vol. 1442, Springer-Verlag, 1998, pp. 196–231.
[27] F. Silva, I. Scherson, Improving parallel job scheduling using runtime measurements, in: D. Feitelson, L. Rudolph (Eds.), JSSPP, in: LNCS, vol. 1911, Springer-Verlag, 2000, pp. 18–39.
[28] O. Sinnen, L. Sousa, et al., Exploiting unused time slots in list scheduling, considering communication contention, in: R. Sakellariou (Ed.), Euro-Par, in: LNCS, vol. 2150, Springer-Verlag, 2001, pp. 166–170.
[29] P Sobalvarro, W. Weihl, Demand based coscheduling of parallel jobs on multiprogrammed multiprocessors, in: Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, vol. 949, Springer-Verlag, Berlin, Germany, 1995, pp. 106–126.
[30] D. Shmoys, J. Wein, D. Williamson, Scheduling parallel machines on-line, SIAM Journal on Computing 24 (1995) 1313–1331.
[31] A. Tchernykh, D. Trystram, Adaptive strategy for on-line scheduling of real world parallel applications, in: SOCAL'02 Southern California Workshop on Parallel and Distributed Processing and Architecture, Santa Barbara, USA, 2002, pp. 7–8.
[32] A. Tchernykh, D. Trystram, et al., On-line scheduling of multiprocessor jobs with idle regulation, in: Wyrzykowski (Ed.), Parallel Processing and Applied Mathematics, in: LNCS, vol. 3019, Springer-Verlag, Berlin, Heidelberg, 2004, pp. 131–144.
[33] F. Wang, M. Papaefthymiou, M. Squillante, Performance evaluation of gang scheduling for parallel and distributed systems, in: Job Scheduling for Parallel Processing, in: LNCS, vol. 1291, Springer-Verlag, Berlin, Germany, 1997, pp. 184–195.
[34] A. Tchernykh, D. Trystram, C. Rapine, Adaptive (a,b,c)-scheme strategy for on-line scheduling, in: New Trends in Scheduling in Parallel and Distributed Systems, CIRM, Luminy, Marseille, France, 2001.
[35] R. Lepere, G. Mounie, D. Trystram, An approximation algorithm for scheduling trees of malleable tasks, EJOR 142 (2002) 242–249.
[36] R. Lepere, G. Mounie, D. Trystram, Malleable tasks: An efficient model for solving actual parallel applications, in: PARCO99, World Scientific, 1999.
[37] R. Graham, Bounds on multiprocessor timing anomalies, SIAM Journal on Applied Mathematics 17 (1969) 263–269.
[38] J. Leung (Ed.), Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapman and Hall/CRC, Boca Raton, FL, 2004, p. 1120.