

On-Line Scheduling of Multiprocessor Jobs with Idle Regulation*

Andrei Tchernykh^{1**}, Denis Trystram²

¹CICESE, Ensenada, Baja California, México
chernykh@cicese.mx

²ID- IMAG, 38330 Montbonnot St. Martin, France
trystram@imag.fr

Abstract. In this paper, we focus on on-line scheduling of multiprocessor jobs with emphasis on the regulation of idle periods in the frame of general list policies. We consider a new family of scheduling strategies based on two phases which successively combine sequential and parallel executions of the jobs. These strategies are part of a more generic scheme introduced in [6]. The main result is to demonstrate that it is possible to estimate the amount of resources that should remain idle for a better regulation of the load and to obtain approximation bounds.

1 Introduction

Different types of parallel applications, depending on their dynamic and static characteristics, introduce different constraints on the scheduling policies. Unfortunately, usually there is no sufficient knowledge about applications in order to schedule them efficiently based on models of the classical theory. Moreover, real application characteristics are too complex to be used as formal scheduling constraints. If the application constraints are unpredictable or, even, undefined, the resource allocation for computing jobs could be done by a scheduler, based on available information as parameters of the processors or the operating system. However, they are very hard to define formally too. For the case of sequential jobs, scheduling problems have been studied for decades [1,3], much less is known about the efficient on-line parallel jobs' scheduling solutions [23,4].

1.1 Applications

Parallelism inside applications adds a new dimension to the scheduling problem. The manner in which jobs partitioning can be done (when a job requires more than one processor at a time) depends on their divisible property. According to the

* This work was supported in part by CONACYT under grant #32989-A

** Part of the work was done during a research stay of Andrei Tchernykh at ID-IMAG, France

classification of Feitelson et al. [5,11], several types of job processor requirements in two general classes named rigid and flexible are distinguished. Rigid jobs include multiprocessor jobs with a fixed number of processors requested for parallel execution that is not changed until the job is completed. They can also have moldable property, and, hence, they can be run on any number of processors, but once the number of processors has been allotted to the job it remains the same throughout its execution [10]. Malleable parallel jobs have flexible 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 moment of allotment, on change in requirements or load. At any moment, 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 numbers of processors during its execution [9].

1.2 Framework for Resource Management

Big variety of job scheduling policies for parallel computers that have been studied in the literature makes it clear that practical scheduling solutions are different and 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 be adapted to them on-line.

In [6], a generic adaptive scheduling strategy, called *(a,b,c)-Scheme*, has been introduced. This framework appears to be a good starting point for understanding the unification of different dynamic scheduling strategies. Preliminary results show that it is possible to design good approximation algorithms for scheduling parallel jobs. The scheme unifies the scheduling of usual sequential jobs and jobs that may require more than one processor for their execution, and automatically adapts to the right granularity of the applications by using a set of three parameters (*a*, *b* and *c* referring respectively to system, application and scheduling strategies characteristics).

1.3 Penalty Factor

To propose a trade-off between the complexity of parallel systems, applications, and desired simplicity of their models for a theoretical study, we consider the model of parallel job based on a *penalty factor*. Such a model has been used in various actual codes [2,7]. The idea is to add an overhead into the parallel execution time, which includes the time lost for communication, synchronization, preemption or 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, very hard to be defined formally, or hides real application characteristics or computer parameters, though known, but too complex to be used as formal scheduling constraints. The penalty of a single job execution or workload processing can be estimated based on an

empirical analysis, benchmarking, counting, profiling, performance evaluation through modelling or measurement, or based on the information provided by a user.

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]. We usually distinguish three broad classes of job penalty factor: namely, constant, linear, and convex. They are the most common classes while parallelizing actual numerical applications. The constant shape of the penalty functions corresponds to the system where applications achieve near linear increasing speed-up with increasing number of processors. The linear penalty functions correspond to the logarithmic shape of a job speed-up function that exhibits speed-up that mostly rises monotonically up to the certain threshold and slowdown beyond a certain number of allocated processors. The convex penalty functions correspond to the small start-up overhead and the addition of extra processors up to the certain threshold costs minimum. After the threshold with greater number of processors, the cost of management of parallelism is increased causing degradation of the speed-up. It correlates with the concave processing speed-up function.

We assume that the penalty factor depends non-decreasingly on the number of processors: $\mu_k^i \leq \mu_{k+1}^i$ for any $1 \leq k \leq m$. For a job T_i allocated to k processors,

$p_k^i = \frac{\mu_k^i p_1^i}{k}$, where p_k^i is the processing time of job T_i executed on k processors.

We consider that p_k^i depends non-increasingly on k , at least for a reasonable number of processors, where $k_{\min} \leq k \leq k_{\max}$. Calculating the speed-up 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 speed-up over achieved one k / S_k^i .

In this paper, we exclude from analysis the applications with super linear speed-up that may be obtained for some big applications exceeding cache or available memory, and applications with speed-up value less than one (that contain not enough parallelism or large parallelization overhead for being executed by several processors).

Let μ_{\max} be $\max_{1,n} \{\mu_k^i\}$, μ_{\min} be $\min_{1,n} \{\mu_k^i\}$, $p_i = p_1^i$, p_{\max} be $\max_{1,n} \{p_i\}$, p_{\max}^{par} be $\max_{1,n} \{\frac{\mu_k^i p_i}{k_i}\}$, k_{\min} be $\min_{1,n} \{k_i\}$, and k_{\max} be $\max_{1,n} \{k_i\}$.

1.4 Idle Regulation

The natural principle of the on-line list scheduling is a greedy method such that a processor cannot be idle if there exist some jobs ready to be executed. Scheduling

strategies used a dynamic idle regulation approach in order to improve system behavior has received some interest recently. Job partitioning with keeping some processors idle was suggested in [20]. An analysis has shown its effectiveness for non-scalable workloads, possibly with high variance in arrival rate and execution. In [14], the authors presented the design and implementation of a scheduling policy that prevents the degradation in performance appeared from scheduling idle processors too early. It keeps some processors idle during some periods of time before scheduling them. Exploiting unused time slots in list scheduling was also considered in [25].

We put emphasize in this work on how to parametrize easily a simple policy based on list scheduling. We consider the general framework introduced in [6], 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. This parameter can be selected in accordance to the current system load, performance of the system, provided by the scheduler or administrator. It depends on the penalty factor of the system and applications, and can be adapted to the change of quality of the workload on-line, based on runtime measuring of workload characteristics. The variation of parameter a gives the possibility 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.

Many models of the workload assumed that information about its performance (speed-up, inefficiency) is available a priori. However, it is clear that the choice of appropriate policies must be guided at least in part by the behaviour of the actual parallel workload that is diverse and can be changed at runtime. *(a,b,c)-Scheme* uses the idea of the runtime tuning of the system parameter a in order to optimize the system behaviour, measuring of workload characteristics (post-mortem analysis of jobs characteristics during a certain time interval) to estimate the penalty factors. The runtime measuring of workload characteristics to improve job scheduling has received considerable attention in recent years. In [19] on-line measuring of job efficiency and speedup in making production schedulers decisions were discussed. In [24], the use of runtime measurements to improve job scheduling on a parallel machine with emphasis on gang scheduling based strategies was investigated.

1.5 Processor Allocation Regulation

The idea of allocation of fewer and fewer processors to each job under heavy load, thus increasing efficiency, was proposed in [12]. The authors considered a processor allocation scheme based on the number of processors, for which the ratio of execution time to efficiency is minimized. The maximizing of application speedup through runtime selection of an appropriate number of processors on which to run was discussed in [18]. The use of a runtime system that dynamically measures job efficiencies at different allocations and automatically adjusts a job's processor allocation to maximize its speedup was proposed. In [17], the dynamic scheduler that uses runtime idleness information to dynamically adjust processor allocations to improve shared memory system utilization was proposed. Self-adaptive scheduling

strategy in a master-worker paradigm that dynamically adjusts the number of processors based on performance measures gathered during its execution was considered in [13]. In [8], it is shown that of several strategies with equivalent turnaround times, the strategy that reduces allocations 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 available can be found in [22]. The technique yields good results for different network topologies. The two-phases strategy for the processor allocation regulation was introduced in [21].

The objective of this paper is to study the impact of idle regulation and processor allocation regulation policies solving on-line rigid multiprocessor jobs scheduling problems in the framework of the two-phases strategy.

The rest of paper is organized as follows: in section 2 we present the notation of scheduling problem and we describe the two-phases strategy for on-line multiprocessor job scheduling. The analysis of the performance guarantee for some cases is presented in section 3. Finally, further research directions are discussed.

2 Preliminaries

In this work, we focus on on-line batch style of scheduling. That means that a set of available ready jobs will be executed up to the completion of the last one. All jobs which arrive in the system during this time will be processed in the next batch.

2.1 Scheduling Problem

We consider a set of independent parallel multiprocessor jobs with the objective to minimize the total execution time (makespan) in the frame of the two-phases list scheduling strategy described in the next section. We restrict the analysis to the scheduling systems where all the jobs are given at time 0 and are processed into the same batch. A relation between this scheme and the scheme where jobs released over time, either at their release time, according to the precedence constraints, or released by different users is known and studied for different scheduling strategies for general or restricted cases [26].

The following problem is studied. Given a parallel machine with m identical processors and a set of n independent multiprocessor jobs $T = \{T_1, \dots, T_n\}$. The main on-line feature is the fact that the processing times p_1, \dots, p_n are not known until their completion. The number of processors k_i needed for the execution of job T_i is fixed and known as soon as it becomes available. We assume that the job irrespective of k_i can be scheduled either on the single processor or requested k_i processors. The job can be preempted, assigned to the required number of processors, then it cannot be preempted and/or continue to run on a different set and/or different number of

processors. 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 processors k_i , and the penalty factor of the execution on these processors.

All the strategies will be analysed according to their competitive ratio. Let C_{opt}^{par} denote the makespan of an optimal schedule, where the parallelization of the jobs is allowed, and C_{opt}^{seq} 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 corresponding schedule. Let $\rho_A^{par} = \frac{C_A}{C_{opt}^{par}}$ and $\rho_A^{seq} = \frac{C_A}{C_{opt}^{seq}}$ denote the parallel and

sequential competitive ratios of strategy A . The sequential competitive ratio will highlight the possible gain to allow the multiprocessing of jobs compared to the classical approach of general list scheduling where jobs are purely sequential.

2.2 The Two-phases Algorithm

In this work, we are interested in studying the influence of idle regulation on a general strategy based on list scheduling. This strategy have been introduced in [21] and consider two successive phases.

When the number of jobs is large, it allocates processors without idle times and communications. When enough jobs have been completed, we switch 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 the efficiency is 1. Inefficiency appears when less than m jobs remain. Then, when the number of idle processors become 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. Figure 1 illustrates this algorithm.

The successive phases are shown: phase 1a (time interval $[0, t]$), when all the processors are busy; phase 1b (time interval $[t, t']$), when at least $m-a+1$ processors work (both phases use the well-known Graham's strategy); and phase 2 (time interval $[t', t'']$), when a or more processors become idle, and, hence, turn to a second strategy with multiprocessor jobs.

Parameter a determines the balance between the number of jobs processed by the first and the second strategies. The execution of jobs sequentially in the first phase under heavy load is a best strategy as there is no extra overhead. We assume that all parallel jobs can be executed sequentially irrespective of their type and the number of processors needed for the execution is imposed or not. Even rigid jobs that carefully tuned-up to achieve a large degree of latency hiding and cannot cope with the reducing or increasing the number of allotted processors are executed efficiently on one processor because of the locality of communications. It is suitable 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), but excludes from our consideration some class of parallel jobs, for example, synchronous shared memory applications, where parallelism cannot be transformed to the time sequence.

On the second phase, when the load is reduced and the number of idle processors becomes equal to or more than a , the system changes the strategy to avoid too many idle times.

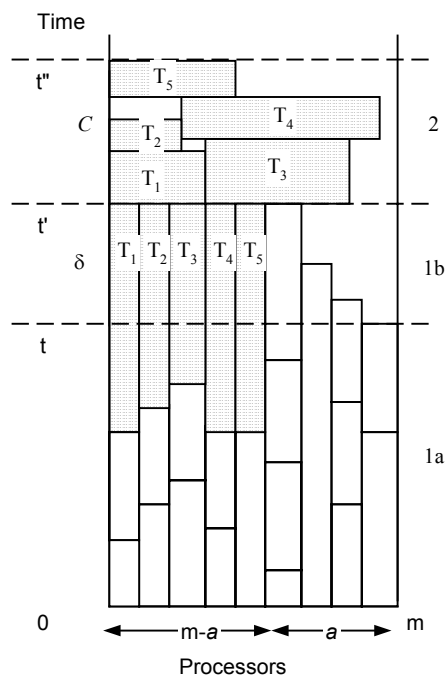


Fig. 1. The two-phases strategy for $a=4$ and $m=9$.

Such an idle regulation gives a possibility to find a trade-off between starting parallelization sooner, when a is smaller (hence more jobs are parallelized causing bigger parallelization overhead), and delaying their parallelization, when a is bigger (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 the computer system. The number of processors for the job execution selected by users is typically suitable for light load conditions, but it leads to unacceptably low processor system efficiency at heavy load [11]. Since users cannot practically know the load on the system, one possible way to optimize system behavior is to support different solutions and be adapted to them on-line.

3 Analysis

We provide now an analysis of the two-phases algorithm with idle regulation for the case $0 \leq a \leq m$. We study the performance in regard to the number of processors allotted to a job for its execution and its penalty factor. Before deriving the general bound, we study the restricted case of $a=0$ (scheduling rigid jobs by a list algorithm). The case $a=1$ for a specific allocation of processors in gang (to m processors) has been studied in [21]. The case $a=m$ corresponds simply to list scheduling for sequential jobs (only first phase).

3.1 $a=0$

We should first notice that multiprocessor rigid jobs strategy corresponds to a 2 dimensional packing problem. Only the second phase of the general algorithm is considered for this case.

Lemma 1. The sequential and parallel performance guarantees of any list algorithm for scheduling of independent multiprocessor jobs are

$$\rho^{seq} \leq \frac{1}{m - k_{\max} + 1} (\mu_{\max} m + \frac{\mu_{\min}}{k_{\min}} (m - k_{\max})) \text{ and } \rho^{par} \leq \frac{2m - k_{\max}}{m - k_{\max} + 1} \quad [15,16]$$

Proof. Let us consider that in the schedule, there are two kind of time slots which can be combined conceptually into two successive intervals, namely C_1 and C_2 . Both intervals correspond respectively to the time slots when at most $k_{\max} - 1$ processors are idle, and strictly more than $k_{\max} - 1$ processors are idle.

Let $W_{seq} = \sum_{i=1}^n p_i$ be the total work of all jobs, $W_{par} = \sum_{i=1}^n \mu_k^i p_i \leq \mu_{\max} W_{seq}$ be the

work performed in the parallel execution, and W_{par}^1 be a the work performed in C_1 .

Noticing that the number of idles in the interval C_2 is limited by $(m - k_{\max})C_2$, and $C_2 \leq p_{\max}^{par}$, we obtain an upper bound of the total completion time

$$C = C_1 + C_2 \leq \frac{W_{par}^1}{m - k_{\max} + 1} + \frac{(W_{par} - W_{par}^1) + (m - k_{\max})p_{\max}^{par}}{m}.$$

$$C \leq \frac{W_{par} + (m - k_{\max})p_{\max}^{par}}{m - k_{\max} + 1}.$$

Noticing also that $\frac{W_{seq}}{m}$ and p_{\max} are lower bounds of C_{opt}^{seq} , and $\frac{W_{par}}{m}$ and p_{\max}^{par} are lower bounds for C_{opt}^{par} , it follows that

$$\rho^{seq} \leq \frac{\mu_{\max} m}{m - k_{\max} + 1} + \frac{\mu_{\min} (m - k_{\max})}{k_{\min} (m - k_{\max} + 1)} \leq \frac{1}{m - k_{\max} + 1} (\mu_{\max} m + \frac{\mu_{\min}}{k_{\min}} (m - k_{\max}))$$

and $\rho^{par} \leq \frac{2m - k_{\max}}{m - k_{\max} + 1}$. \square

Remark. Let us check briefly what happens for sequential jobs: we obtain the same bound as Graham (no penalty, $\mu_{\min} = \mu_{\max} = 1$ and $k_{\min} = k_{\max} = 1$) and both competitive ratio coincide.

3.2 Idle Regulation with $a > 0$.

Let us consider now that the strategy switches from the *Graham's* strategy to multiprocessor job scheduling when a processors become idle (Figure 1).

Theorem 1 (Sequential competitiveness). Given a set of n independent multiprocessor jobs with variation of the penalty factors from μ_{\min} to μ_{\max} when allotted on fixed number of processors from k_{\min} to k_{\max} , the sequential performance guarantee can be estimated by $\rho^{seq} = \max\{\rho^1, \rho^2\}$, with

$$\rho^1 \leq 1 + \frac{a-1}{m} \text{ and } \rho^2 \leq \frac{a}{m} + \frac{1}{m - k_{\max} + 1} (\mu_{\max} (m - a) + \frac{\mu_{\min}}{k_{\min}} (m - k_{\max})).$$

Proof. Let denote by W_{seq} the total work, and by W_{seq}^1 the work executed until t . Each job not finished until t' is executed necessarily in phase 2. Then, assuming that part of these jobs have not been processed in phase 1a or 1b the remaining work is less than $W_{seq}^2 \leq \sum_{i=1}^h (p_i - \delta) \leq h(p_{\max} - \delta)$, where $h \leq m - a$, and $\delta = t' - t$. Let

p_{rem}^{par} be $\max_{1,n} \{\frac{\mu_k^i (p_i - \delta)}{k_i}\}$. Following Lemma 1 for the completion time of the

phase 2, the total completion time C is $\frac{W_{seq}^1}{m} + \delta + \frac{\mu_{\max} W_{seq}^2 + (m - k_{\max}) p_{rem}^{par}}{m - k_{\max} + 1}$.

$W_{seq} \geq W_{seq}^1 + (m - a + 1)\delta + W_{seq}^2$. Hence

$$C \leq \frac{W_{seq}}{m} - \frac{m - a + 1}{m} \delta + \delta + W_{seq}^2 \left(\frac{\mu_{\max}}{m - k_{\max} + 1} - \frac{1}{m} \right) + \frac{(m - k_{\max}) p_{rem}^{par}}{m - k_{\max} + 1}$$

Considering h jobs executed after t' , and $p_{rem}^{par} \leq \frac{\mu_{k \min} (p_{\max} - \delta)}{k_{\min}}$ it follows that

$$C \leq \frac{W_{seq}}{m} + \left(\frac{a-1}{m} - \left(\frac{h\mu_{max}}{m-k_{max}+1} + \frac{(m-k_{max})\mu_{min}}{k_{min}(m-k_{max}+1)} - \frac{h}{m} \right) \delta + \left(\frac{h\mu_{max}}{m-k_{max}+1} + \frac{(m-k_{max})\mu_{min}}{k_{min}(m-k_{max}+1)} - \frac{h}{m} \right) p_{max} \right).$$

Let denote $B(a) = \frac{a-1}{m}$, $A(h) = \frac{h\mu_{max}}{m-d} + \frac{(m-k_{max})\mu_{min}}{k_{min}(m-d)} - \frac{h}{m}$, and

$$J(a, h) = (B(a) - A(h))\delta + A(h)p_{max}. \text{ Hence, } C \leq \frac{W_{seq}}{m} + J(a, h).$$

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

If $A(h) = B(a)$ then $J(a, h) \leq A(h)p_{max}$,

$$C \leq \frac{W_{seq}}{m} + \left(\frac{h\mu_k}{m-k_{max}+1} + \frac{(m-k_{max})\mu_{kmin}}{k_{min}(m-k_{max}+1)} - \frac{h}{m} \right) p_{max}, \text{ and}$$

$$\rho^{seq} \leq \frac{a}{m} + \frac{1}{m-k_{max}+1} (\mu_{max}(m-a) + \frac{\mu_{min}}{k_{min}}(m-k_{max})).$$

If $A(h) < B(a)$ then $J(a, h) \leq (B(a) - A(h))p + A(h)p \leq B(a)p$,

$$C \leq \frac{W_{seq}}{m} + \frac{a-1}{m} p_{max}, \text{ and } \rho^{seq} \leq 1 + \frac{a-1}{m}$$

If $A(h) > B(a)$ then $J(a, h) \leq A(h)p$,

$$C \leq \frac{W_{seq}}{m} + \left(\frac{\mu_{max}h + (m-k_{max})}{m-k_{max}+1} - \frac{h}{m} \right) p_{max} \text{ and}$$

$$\rho^{seq} \leq \frac{a}{m} + \frac{1}{m-k_{max}+1} (\mu_{max}(m-a) + \frac{\mu_{min}}{k_{min}}(m-k_{max})) \quad \square$$

Remarks. For $a = 0$, $\rho^{seq} \leq \frac{1}{m-k_{max}+1} (\mu_{max}m + \frac{\mu_{min}}{k_{min}}(m-k_{max}))$ that

corresponds to a sequential performance guarantee of any list algorithm for scheduling multiprocessor jobs (Lemma 1). For $a = m$, and $k_{max} = k_{min} = 1$,

$\rho^{seq} \leq 2 - \frac{1}{m}$, only the *Graham* strategy (phase 1) is applied.

Figure 2 shows the sequential competitive ratio ρ^{seq} when varying the number of processors with fixed a , k_{max} , k_{min} , linear μ_k (logarithmic shape of a job speed-up function of k), and constant μ_k (linear $m/2$ shape of a job speed-up function of k). We assume here that the number of unfinished jobs on phase 2 is $m-a$. Figure 3 presents the performance when varying the parameter a with fixed number of processors, k_{max} , k_{min} , linear and constant μ_k . It shows that tuning of the idle times by the parameter a allows to get the minimum of the worst case error bound.

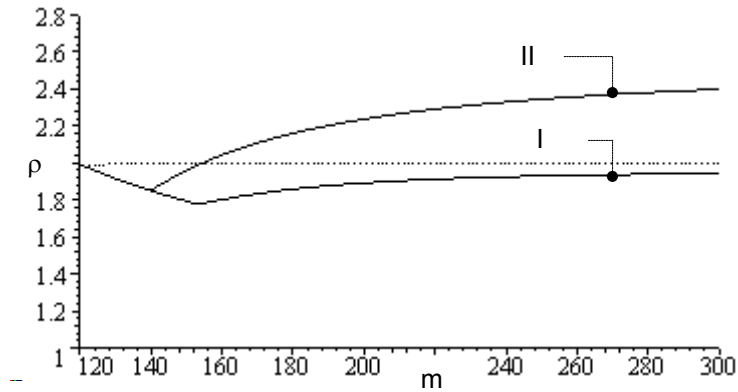


Fig. 2. ρ^{seq} when varying the number of processors; with $a=120$, $k_{max}=60$, $k_{min} = k_{max}/30$. (I) linear μ_k (logarithmic shape of a job speed-up function), and (II) constant μ_k (linear $m/2$ shape of a job speed-up function of k).

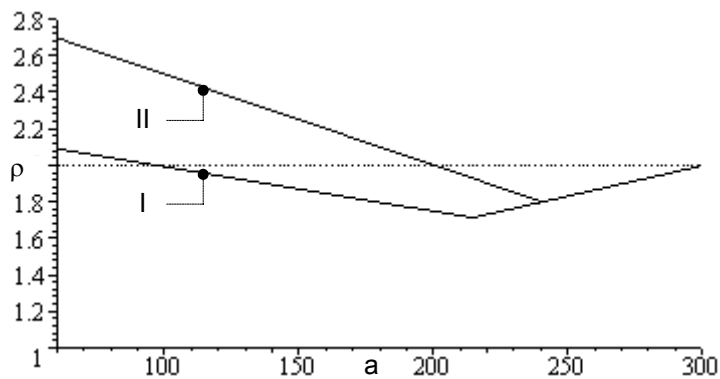


Fig. 3. ρ^{seq} when varying a , with $m=300$; (I) linear μ_k (logarithmic shape of a job speed-up function) and (II) constant μ_k (linear $m/2$ shape of a job speed-up function of k), $k_{max}=60$, $k_{min} = k_{max}/30$.

Theorem 2 (Parallel competitiveness). Given a set of n independent multiprocessor jobs, the parallel performance guarantee can be estimated by :

$$\rho^{par} \leq \frac{(2m - k_{max})}{m - k_{max} + 1} - a \left(\frac{1}{m - k_{max} + 1} - \frac{1}{m} \right).$$

Proof. As shown in Theorem 1, the completion time of the schedule is

$$C \leq \frac{W_{seq}^1}{m} + \delta + \frac{W_{par}^2 + (m - k_{max}) p_{rem}^{par}}{m - k_{max} + 1}. \text{ Notice that}$$

$W_{par} \geq \mu_{\min} W_{seq}^1 + (m - a + 1)\delta\mu_{\min} + W_{par}^2 \geq W_{seq}^1 + (m - a + 1)\delta + W_{par}^2$ It follows

$$C \leq \frac{W_{par}}{m} + \frac{a-1}{m}\delta + W_{par}^2 \left(\frac{1}{m-k_{\max}+1} - \frac{1}{m} \right) + \frac{(m-k_{\max})p_{rem}^{par}}{m-k_{\max}+1}. \quad \text{Considering}$$

$W_{par}^2 \leq hp_{rem}^{par}$, where $p_{rem}^{par} = \frac{\mu'(p' - \delta)}{k'}$ be $\max_{1,n} \left\{ \frac{\mu_k^i(p_i - \delta)}{k_i} \right\}$, it follows that

$$C \leq \frac{W_{par}}{m} + \left(\frac{a-1}{m} - \left(\frac{h}{m-k_{\max}+1} - \frac{h}{m} + \frac{(m-k_{\max})}{m-k_{\max}+1} \right) \frac{\mu'}{k'} \right) \delta +$$

$$\left(\frac{h}{m-k_{\max}+1} - \frac{h}{m} + \frac{(m-k_{\max})}{m-k_{\max}+1} \right) \frac{\mu' p'}{k'}$$

Let denote $B(a) = \frac{a-1}{m}$, $A(h) = \left(\frac{h}{m-k_{\max}+1} - \frac{h}{m} + \frac{(m-k_{\max})}{m-k_{\max}+1} \right) \frac{\mu'}{k'}$, and

$$J(a, h) = (B(a) - A(h))\delta + A(h)p'. \text{ Hence } C \leq \frac{W_{par}}{m} + J(a, h).$$

Noticing that $A(h) \geq B(a)$, it follows $J(a, h) \leq A(h)p'$. Hence

$$C \leq \frac{W_{par}}{m} + \left(\frac{h}{m-k_{\max}+1} - \frac{h}{m} + \frac{(m-k_{\max})}{m-k_{\max}+1} \right) \frac{\mu' p'}{k'}$$

Lower bounds of an optimal schedule are $C_{opt}^{par} \geq \frac{W_{par}}{m}$ and

$$C_{opt}^{par} \geq p_{rem}^{par} + \frac{\mu'}{k'}\delta = \frac{\mu'}{k'}p'. \text{ Thus } \rho^{par} \leq \frac{(2m-k_{\max})}{m-k_{\max}+1} - a \left(\frac{1}{m-k_{\max}+1} - \frac{1}{m} \right) \square$$

Remarks. For $a = 0$, $\rho^{par} \leq \frac{2m-k_{\max}}{m-k_{\max}+1}$ that corresponds to a performance guarantee of any list algorithm for scheduling multiprocessor jobs. For $k_{\max} = 1$, $\rho^{par} \leq 2 - \frac{1}{m}$, only the *Graham* strategy is applied.

4 Conclusion and Future Work

In this paper, we focused on on-line scheduling of multiprocessor rigid jobs with emphasis on the idle regulation. We considered non-clairvoyant scheduler that have no information about the jobs other than the number of unfinished jobs in the system and their processor requirements. We proposed to solve on-line scheduling problems

using batch scheduling under a generic framework of two-phases of list scheduling (for sequential jobs and rigid multiprocessor jobs).

Scheduling strategies used a dynamic idle regulation approach in order to improve system behavior has receive attention recently. In this paper, we showed that tuning of the parameter a that regulates idle times gives a possibility to find a tradeoff between avoiding idle processors by starting parallelization sooner when more tasks are parallelized and delaying their parallelization until a smaller number of jobs is available. A tradeoff is a minimum of the performance guarantee for a parallel computer system with fixed number of processors and known overhead. The minimum depends on the penalty factor of the system and applications, and can be adapted to the change of quality of the workload on-line, based on runtime measuring of workload characteristics.

Several important and interesting questions still remain unanswered. Firstly, analysis and simulations of the strategy with more variations of other features like moldable or malleable jobs, and different scheduling strategies (FCFS, largest job first, etc.). Secondly, the analysis of the system to be able to support scheduling dependent jobs, or when new jobs can arrive in any moment that is more practical considering that workloads vary in a day cycle. There is considerable scope to obtain results by using the scheme to the scheduling of malleable jobs that can be scheduled on any available number of processors. The issues related to the implementation of the strategy on real parallel or distributed computing systems are also important.

References

1. J. Błażewicz, M. Drozdowski, K. Ecker, "Management of resources in parallel systems", in the 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. V22, 2, pp. 8-23, 2000
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. S. Chapin, W. Cirne, D. Feitelson, J. Patton Jones, S.T. Leutenegger, U. Schwiigelshohn, W. Smith, and D. Talby, "Benchmarks and Standards for the Evaluation of Parallel Job Schedulers". In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1999, LNCS. vol. 1659, pp. 66-89.
6. A. Chernykh, C. Rapine, D. Trystram. Adaptive scheme for on-line scheduling of parallel jobs. Technical report at IMAG, 2003.
7. T. Decker, W. Krandick, Parallel real root isolation using the Descartes method, Proceedings of the 6th High Performance Conference (HiPC99), 261-268, LNCS, vol. 1745, Springer-Verlag, 1999
8. A. Downey. A parallel workload model and its implications for processor allocation. In Proc. the 6th International Symposium of High Performance Distributed Computing, pp 112-123, 1997.

9. D. Feitelson and L. Rudolph. "Toward convergence in job schedulers for parallel supercomputers" In Job Scheduling Strategies for Parallel Processing, D. Feitelson and L. Rudolph (Eds.), pp. 1-26, Springer-Verlag, 1996. LNCS 1162.
10. D. Feitelson and L. Rudolph. Metrics and Benchmarking for Parallel Job Scheduling. D.G. Feitelson, L. Rudolph (Eds.): JSSPP, IPPS/SPDP'98 Workshop, Orlando, Florida, USA, March 1998. Proceedings, LNCS 1459, p. 1-24.
11. D. Feitelson, L. Rudolph, U. Schweigelshohn, K. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling". In Job Scheduling Strategies for Parallel Processing D. G. Feitelson and L. Rudolph (Eds.), pp. 1-34, Springer-Verlag, 1997. LNCS, Vol. 1291.
12. D. Ghosal, G. Serazii, and S. Tripathi. "The processor working set and its use in scheduling multiprocessor system", IEEE Trans. Soft. Eng. 17(5), pp. 443-453, May 1991
13. E. Heymann, M. Senar, E. Luque, and M. Livny. Self-Adjusting Scheduling of Master-Worker Applications on Distributed Clusters. R. Sakellariou et al. (Eds.): Euro-Par 2001, Manchester, UK August 28-31, 2001, Proceedings, LNCS 2150, p. 742-751.
14. S. Iyer, P. Druschel, Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O, Symposium on Operating Systems Principles, 117-130, 2001
15. E. Lloyd Concurrent task systems, Operational Research 29/1, 1981, pp. 189-201
16. R. Lepere, G. Mounie, D. Trystram. An Approximation algorithm for scheduling Trees of Malleable Tasks. EJOR, 2002
17. C. McCann, R. Vaswani, and J. Zahorjan. A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors. ACM Transactions on Computer System, 11(2): 146-178, May 1993
18. T. Nguyen, R. Vaswani, and J. Zahorjan. Parallel Application Characterization for Multiprocessor Scheduling Policy Design. In JSSPP, D.G. Feitelson and L. Rudolph (eds.), Volume 1162 of LNCS, Springer-Verlag, 1996.
19. T. Nguyen, R. Vaswani, and J. Zahorjan. Using Runtime Measured Workload Characteristics in Parallel Processor Scheduling. In JSSPP, D. G. Feitelson and L. Rudolph (editors), vol. 1162, LNCS. Springer-Verlag, 1996.
20. E. Rosti, E. Smirni, G. Serazzi, L. Dowdy. "Analysis of non-work-conserving processor partitioning policies". In Job Scheduling Strategies for Parallel Processing, D. Feitelson and L. Rudolph (Eds.), pp. 165-181, Springer-Verlag, 1995. LNCS 949.
21. C. Rapine, I. Scherson, D. Trystram. On-Line Scheduling of Parallelizable Jobs. Proceedings of EUROPAR'98 Conference, Springer-Verlag, LNCS, Vol. 1470, pp. 322-327, 1998
22. J. Sgall, A. Feldmann, M. Kao, and S. Teng. Optimal online scheduling of parallel jobs with dependencies J. of Combinatorial Optimization, 1(4):393-411, 1998
23. J. Sgall. On-line scheduling-A Survey, In Online Algorithms: The State of the Art, eds. A. Fiat and G. J. Woeginger, LNCS 1442, pp 196-231, Springer-Verlag, 1998.
24. F. Silva and I. Scherson. Improving Parallel Job Scheduling Using Runtime Measurements. In D.G. Feitelson and L. Rudolph (Eds.): JSSPP 2000, LNCS 1911, Springer-Verlag, pp. 18-39, 2000
25. O. Sinnen and L. Sousa. Exploiting Unused Time Slots in List Scheduling, Considering Communication Contention, R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 166-170, Springer-Verlag, 2001
26. D. Shmoys, J. Wein, D. Williamson. Scheduling parallel machines on-line. SIAM J. Comput., 24:1313-1331, 1995.