Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n^o 5668

The impact of heterogeneity on master-slave on-line scheduling

Jean-François Pineau,
Yves Robert,
Frédéric Vivien

October 2005

Research Report N^o 2005-51

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

The impact of heterogeneity on master-slave on-line scheduling

Jean-François Pineau, Yves Robert, Frédéric Vivien

October 2005

Abstract

In this paper, we assess the impact of heterogeneity for scheduling independent tasks on master-slave platforms. We assume a realistic one-port model where the master can communicate with a single slave at any time-step. We target on-line scheduling problems, and we focus on simpler instances where all tasks have the same size. While such problems can be solved in polynomial time on homogeneous platforms, we show that there does not exist any optimal deterministic algorithm for heterogeneous platforms. Whether the source of heterogeneity comes from computation speeds, or from communication bandwidths, or from both, we establish lower bounds on the competitive ratio of any deterministic algorithm. We provide such bounds for the most important objective functions: the minimization of the makespan (or total execution time), the minimization of the maximum response time (difference between completion time and release time), and the minimization of the sum of all response times. Altogether, we obtain nine theorems which nicely assess the impact of heterogeneity on on-line scheduling. These theoretical contributions are complemented on the practical side by the implementation of several heuristics on a small but fully heterogeneous MPI platform. Our (preliminary) results show the superiority of those heuristics which fully take into account the relative capacity of the communication links.

Keywords: Scheduling, Master-slave platforms, Heterogeneous computing, On-line

Résumé

Dans cet article, nous regardons l'impact de l'hétérogénéité sur l'ordonnancement de tâches indépendantes sur une plate-forme maître-esclave. Nous supposons avoir un modèle un-port, où le maître ne peut communiquer qu'à un seul esclave à la fois. Nous regardons les problèmes d'ordonnancement à la volée, et nous nous concentrons sur les cas simples où toutes les tâches sont indépendantes et de même taille. Tandis que de tels problèmes peuvent être résolus de façon polynomiale sur des plates-formes homogènes, nous montrons qu'il n'existe pas d'algorithme optimal pour des plates-formes hétérogènes, que la source de l'hétérogénéité vienne des processeurs, des bandes passantes, ou des deux à la fois. Dans tous les cas, nous donnons des bornes inférieures de compétitivité pour les fonctions objectives suivantes : la minimisation du makespan (temps total d'exécution), la minimisation du temps de réponse maximal (différence entre la date d'arrivée et la date de fin de calcul), et la minimisation de la somme des temps de réponse. En tout, nous obtenons neuf théorèmes, qui traduisent l'impact de l'hétérogénéité sur l'ordonnancement à la volée. Ces contributions théoriques sont complétées sur le plan pratique par l'implémentation de plusieurs heuristiques sur une plate-forme MPI, petite mais hétérogène. Nos résultats (préliminaires) montrent la supériorité des heuristiques qui prennent pleinement en compte la capacité des liens de communication.

Mots-clés: Ordonnancement en ligne, Ordonnancement hors-ligne, Calcul hétérogène, Plate-forme maître-esclave

1 Introduction

The main objective of this paper is to assess the impact of heterogeneity for scheduling independent tasks on master-slave platforms. We make two important assumptions that render our approach very significant in practice. First we assume that the target platform is operated under the one-port model, where the master can communicate with a single slave at any time-step. This model is much more realistic than the standard model from the literature, where the number of simultaneous messages involving a processor is not bounded. Second we consider on-line scheduling problems, i.e., problems where release times and sizes of incoming tasks are not known in advance. Such dynamic scheduling problems are more difficult to deal with than their static counterparts (for which all task characteristics are available before the execution begins) but they encompass a broader spectrum of applications.

We endorse the somewhat restrictive hypothesis that all tasks are identical, i.e., that all tasks are equally demanding in terms of communications (volume of data sent by the master to the slave which the task is assigned to) and of computations (number of flops required for the execution). We point out that many important scheduling problems involve large collections, or *bags*, of identical tasks [10, 1].

Without the hypothesis of having same-size tasks, the impact of heterogeneity cannot be studied. Indeed, scheduling different-size tasks on a homogeneous platform reduced to a master and two identical slaves, without paying any cost for the communications from the master, and without any release time, is already an NP-hard problem [13]. In other words, the simplest (off-line) version is NP-hard on the simplest (two-slave) platform.

On the contrary, scheduling same-size tasks is easy on fully homogeneous platforms. Consider a master-slave platform with m slaves, all with the same communication and computation capacity; consider a set of identical tasks, whose release times are not known in advance. An optimal approach to solve this on-line scheduling problem is the following list-scheduling strategy: process tasks in a FIFO order, according to their release times; send the first unscheduled task to the processor whose ready-time is minimum. Here, the ready-time of a processor is the time at which it has completed the execution of all the tasks that have already been assigned to it. It is striking that this simple strategy is optimal for many classical objective functions, including the minimization of the makespan (or total execution time), the minimization of the maximum response time (difference between completion time and release time), and the minimization of the sum of the response times.

On-line scheduling of same-size tasks on heterogeneous platforms is much more difficult. In this paper, we study the impact of heterogeneity from two sources. First we consider a communication-homogeneous platform, i.e., where communication links are identical: the heterogeneity comes solely from the computations (we assume that the slaves have different speeds). On such platforms, we show that there does not exist any optimal deterministic algorithm for on-line scheduling. This holds true for the previous three objective functions (makespan, maximum response time, sum of response times). We even establish lower bounds on the competitive ratio of any deterministic algorithm. For example, we prove that there exist problem instances where the makespan of any deterministic algorithm is at least 1.25 times larger than that of the optimal schedule. This means that the competitive ratio of any deterministic algorithm is at least 1.25. We prove similar results for the other objective functions.

The second source of heterogeneity is studied with computation-homogeneous platforms, i.e., where computation speeds are identical: the heterogeneity comes solely from the different-speed communication links. In this context, we prove similar results, but with different competitive ratios.

Not surprisingly, when both sources of heterogeneity add up, the complexity goes beyond the worst scenario with a single source. In other words, for fully heterogeneous platforms, we derive competitive ratios that are higher than the maximum of the ratios with a single source of heterogeneity.

The main contributions of this paper are mostly theoretical. However, on the practical side, we have implemented several heuristics on a small but fully heterogeneous MPI platform. Our (preliminary) results show the superiority of those heuristics which fully take into account the relative capacity of the communication links.

The rest of the paper is organized as follows. In Section 2, we state some notations for the scheduling problems under consideration. Section 3 is devoted to the theoretical results. We start in Section 3.1 with a global overview of the approach and a summary of all results: three platform types and three objective functions lead to nine theorems! We study communication-homogeneous platforms in Section 3.2, computation-homogeneous platforms in Section 3.3, and fully heterogeneous platforms in Section 3.4. We provide an experimental comparison of several scheduling heuristics in Section 4. Section 5 is devoted to an overview of related work. Finally, we state some concluding remarks in Section 6.

2 Framework

Assume that the platform is composed of a master and m slaves $P_1, P_2, \ldots, P_m$. Let c_j be the time needed by the master to send a task to P_j, and let p_j be the time needed by P_j to execute a task. As for the tasks, we simply number them $1, 2, \ldots, i, \ldots$ We let r_i be the release time of task i, i.e., the time at which task i becomes available on the master. In on-line scheduling problems, the r_i's are not known in advance. Finally, we let C_i denote the end of the execution of task i under the target schedule.

To be consistent with the literature [17, 9], we describe our scheduling problems using the notation $\alpha \mid \beta \mid \gamma$ where:

α: **the platform**– As in the standard, we use P for platforms with identical processors, and Q for platforms with different-speed processors. As we only target sets of same-size tasks, we always fall within the model of *uniform processors*: the execution time of a task on a processor only depends on the processor running it and not on the task itself. We add *MS* to this field to indicate that we work with master-slave platforms.

β: **the constraints**– We write *on-line* for on-line problems. We also write $c_j = c$ for communication-homogeneous platforms, and $p_j = p$ for computation-homogeneous platforms.

γ: **the objective**– We deal with three objective functions:

- the *makespan* or total execution time *max* C_i;
- the maximum flow *max-flow* or maximum response time *max* $(C_i - r_i)$: indeed, $C_i - r_i$ is the time spent by task i in the system;
- the sum of response times $\sum(C_i - r_i)$ or *sum-flow*, which is equivalent to the sum of completion times $\sum C_i$ (because $\sum r_i$ is a constant independent of the scheduling).

3 Theoretical results

3.1 Overview and summary

Given a platform (say, with homogeneous communication links) and an objective function (say, makespan minimization), how can we establish a bound on the competitive ratio on the performance of any deterministic scheduling algorithm? Intuitively, the approach is the following. We assume a scheduling algorithm, and we run it against a scenario elaborated by an adversary. The adversary analyzes the decisions taken by the algorithm, and reacts against them. For instance if the algorithm has scheduled a given task T on P_1 then the adversary will send two more tasks, while if the algorithm schedules T on P_3 then the adversary terminates the instance. In the end, we compute the performance ratio: we divide the makespan achieved by the algorithm by the makespan of the optimal solution, which we determine off-line, i.e., with a complete knowledge

Platform type	Objective function		
	Makespan	Max-flow	Sum-flow
Communication homogeneous	$\frac{5}{4} = 1.250$	$\frac{5-\sqrt{7}}{2} \approx 1.177$	$\frac{2+4\sqrt{2}}{7} \approx 1.093$
Computation homogeneous	$\frac{6}{5} = 1.200$	$\frac{5}{4} = 1.250$	$\frac{23}{22} \approx 1.045$
Heterogeneous	$\frac{1+\sqrt{3}}{2} \approx 1.366$	$\sqrt{2} \approx 1.414$	$\frac{\sqrt{13}-1}{2} \approx 1.302$

Table 1: Lower bounds on the competitive ratio of on-line algorithms, depending on the platform type and on the objective function.

of the problem instance (all tasks and their release dates). In one execution (task T on P_1) this performance ratio will be, say, 1.1 while in another one (task T on P_3) it will be, say, 1.2. Clearly, the minimum of the performance ratios over all execution scenarios is the desired bound on the competitive ratio of the algorithm: no algorithm can do better than this bound!

Because we have three platform types (communication-homogeneous, computation-homogeneous, fully heterogeneous) and three objective functions (makespan, max-flow, sum-flow), we have nine bounds to establish. Table 1 summarizes the results, and shows the influence on the platform type on the difficulty of the problem. As expected, mixing both sources of heterogeneity (i.e., having both heterogeneous computations and communications) renders the problem the most difficult.

3.2 Communication-homogeneous platforms

In this section, we have $c_j = c$ but different-speed processors. We order them so that P_1 is the fastest processor (p_1 is the smallest computing time p_i), while P_m is the slowest processor.

Theorem 1. *There is no scheduling algorithm for the problem* $Q, MS \mid online,\ r_i,\ p_j,\ c_j = c \mid \max\ C_i$ *with a competitive ratio less than* $\frac{5}{4}$.

Proof. Suppose the existence of an on-line algorithm $\mathcal{A}$ with a competitive ratio $\rho = \frac{5}{4} - \epsilon$, with $\epsilon > 0$. We will build a platform and study the behavior of $\mathcal{A}$ opposed to our adversary. The platform consists of two processors, where $p_1 = 3$, $p_2 = 7$, and $c = 1$.

Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ sends the task i either on P_1, achieving a makespan at least[1] equal to $c + p_1 = 4$, or on P_2, with a makespan at least equal to $c + p_2 = 8$. At time $t_1 = c$, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and the adversary reacts consequently:

1. If $\mathcal{A}$ did not begin the sending of the task i, the adversary does not send other tasks. The best makespan is then $t_1 + c + p_1 = 5$. As the optimal makespan is 4, we have a competitive ratio of $\frac{5}{4} > \rho$. This refutes the assumption on ρ. Thus the algorithm $\mathcal{A}$ must have scheduled the task i at time c.

2. If $\mathcal{A}$ scheduled the task i on P_2 the adversary does not send other tasks. The best possible makespan is then equal to $c + p_2 = 8$, which is even worse than the previous case. Consequently, algorithm $\mathcal{A}$ does not have another choice than to schedule the task i on P_1 in order to be able to respect its competitive ratio.

At time $t_1 = c$, the adversary sends another task, j. In this case, we look, at time $t_2 = 2c$, at the assignment $\mathcal{A}$ made for j:

1. If j is sent on P_2, the adversary does not send any more task. The best achievable makespan is then $\max\{c + p_1, 2c + p_2\} = \max\{1 + 3, 2 + 7\} = 9$, whereas the optimal is to send the two tasks to P_1 for a makespan of $\max\{c + 2p_1, 2c + p_1\} = 7$. The competitive ratio is then $\frac{9}{7} > \frac{5}{4} > \rho$.

[1] Nothing forces $\mathcal{A}$ to send the task i as soon as possible.

2. If j is sent on P_1 the adversary sends a last task at time $t_2 = 2c$. Then the schedule has the choice to execute the last task either on P_1 for a makespan of $\max\{c{+}3p_1, 2c{+}2p_1, 3c{+}p_1\} = \max\{10, 6\} = 10$, or on P_2 for a makepsan of $\max\{c + 2p_1, 3c + p_2\} = \max\{6, 5, 10\} = 10$. The best achievable makespan is then 10. However, scheduling the first task on P_2 and the two others on P_1 leads to a makespan of $\max\{c + p_2, 2c + 2p_1, 3c + p_1\} = \max\{8, 8, 6\} = 8$. The competitive ratio is therefore at least equal to $\frac{10}{8} = \frac{5}{4} > \rho$.

3. If j is not sent yet, then the adversary sends a last task at time $t_2 = c_2$. $\mathcal{A}$ has the choice to execute j on P_1, and to achieve a makespan worse than the previous case, or on P_2. And it has then the choice to send k either on P_2 for a makespan of $\max\{c{+}p_1, t_2{+}c{+}p_2{+}\max\{c, p_2\}\} = \max\{4, 17\} = 17$, or on P_1 for a makespan of $\max\{c + 2p_1, t_2 + c + p_2, t_2 + 2c + p_1\} = max\{7, 10, 7\} = 10$. The best achievable makespan is then 10. The competitive ratio is therefore at least equal to $\frac{10}{8} = \frac{5}{4} > \rho$. Hence the desired contradiction.

□

Theorem 2. *There is no scheduling algorithm for the problem $Q, MS \mid online,\ r_i,\ p_j, c_j = c \mid \sum (C_i - r_i)$ with a competitive ratio less than $\frac{2+4\sqrt{2}}{7}$.*

Proof. Suppose the existence of an on-line algorithm $\mathcal{A}$ with a competitive ratio $\rho = \frac{2+4\sqrt{2}}{7} - \epsilon$, with $\epsilon > 0$. We will build a platform and study the behavior of $\mathcal{A}$ opposed to our adversary. The platform consists of two processors, where $p_1 = 2$, $p_2 = 4\sqrt{2} - 2$, and $c = 1$.

Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ sends the task i either on P_1, achieving a sum-flow at least equal to $c + p_1 = 3$, or on P_2, with a sum-flow at least equal to $c + p_2 = 4\sqrt{2} - 1$. At time $t_1 = c$, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and the adversary reacts consequently:

1. If $\mathcal{A}$ did not begin the sending of the task i, the adversary does not send other tasks. The best sum-flow is then $t_1 + c + p_1 = 4$. As the optimal sum-flow is 3, we have a competitive ratio of $\frac{4}{3} > \rho$. This refutes the assumption on ρ. Thus the algorithm $\mathcal{A}$ must have scheduled the task i at time c.

2. If $\mathcal{A}$ scheduled the task i on P_2 the adversary does not send other tasks. The best possible sum-flow is then equal to $c + p_2 = 4\sqrt{2} - 1$, which is even worse than the previous case. Consequently, algorithm $\mathcal{A}$ does not have another choice than to schedule the task i on P_1 in order to be able to respect its competitive ratio.

At time $t_1 = c$, the adversary sends another task, j. In this case, we look, at time $t_2 = 2c$, at the assignment $\mathcal{A}$ made for j:

1. If j is sent on P_2, the adversary does not send any more task. The best achievable sum-flow is then $(c + p_1) + ((2c + p_2) - t_1) = 2 + 4\sqrt{2}$, whereas the optimal is to send the two tasks to P_1 for a sum-flow of $(c + p_1) + (\max\{2c + p_1, c + 2p_1\} - t_1) = 7$. The competitive ratio is then $\frac{2+4\sqrt{2}}{7} > \rho$.

2. If j is sent on P_1 the adversary sends a last task at time $t_2 = 2c$. Then the schedule has the choice to execute the last task either on P_1 for a sum-flow of $(c + p_1) + (\max\{c + 2p_1, 2c + p_1\} - t_1) + (\max\{3c + p_1, c + 3p_1\} - t_2) = 12$, or on P_2 for a sum-flow of $(c + p_1) + (\max\{c + 2p_1, 2c + p_1\} - t_1) + ((3c + p_2) - t_2) = 6 + 4\sqrt{2}$. The best achievable sum-flow is then $6 + 4\sqrt{2}$. However, scheduling the *second* task on P_2 and the two others on P_1 leads to a sum-flow of $(c + p_1) + ((2c + p_2) - t_1) + (\max\{3c + p_1, c + 2p_1\} - t_2) = 5 + 4\sqrt{2}$. The competitive ratio is therefore at least equal to $\frac{6+4\sqrt{2}}{5+4\sqrt{2}} = \frac{2+4\sqrt{2}}{7} > \rho$.

3. If j is not send yet, then the adversary sends a last task k at time $t_2 = 2c$. Then the schedule has the choice to execute j either on P_1, and achieving a sum-flow worse than the previous case, or on P_2. Then, it can choose to execute the last task either on P_2 for a sum-flow of

$(c+p_1)+(t_2+c+p_2-t_1)+(t_2+c+p_2+\max\{c,p_2\}-t_2) = 12\sqrt{2}+2$, or on P_1 for a sum-flow of $(c+p_1)+(t_2+c+p_2-t_1)+(\max\{t_2+c+p_1, c+2p_1\}-t_2) = 7+4\sqrt{2}$. The best achievable sum-flow is then $7+4\sqrt{2}$ which is even worse than the previous case. Hence the desired contradiction.

□

Theorem 3. *There is no scheduling algorithm for the problem $Q, MS \mid online,\ r_i,\ p_j,\ c_j = c \mid \max\ (C_i - r_i)$ with a competitive ratio less than $\frac{5-\sqrt{7}}{2}$.*

Proof. Suppose the existence of an on-line algorithm $\mathcal{A}$ with a competitive ratio $\rho = \frac{5-\sqrt{7}}{2} - \epsilon$, with $\epsilon > 0$. We will build a platform and study the behavior of $\mathcal{A}$ opposed to our adversary. The platform consists of two processors, where $p_1 = \frac{2+\sqrt{7}}{3}$, $p_2 = \frac{1+2\sqrt{7}}{3}$, and $c = 1$.

Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ sends the task i either on P_1, achieving a max-flow at least equal to $c + p_1 = \frac{5+\sqrt{7}}{3}$, or on P_2, with a max-flow at least equal to $c+p_2 = \frac{4+2\sqrt{7}}{3}$. At time $\tau = \frac{4-\sqrt{7}}{3}$, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and the adversary reacts consequently:

1. If $\mathcal{A}$ did not begin the sending of the task i, the adversary does not send other tasks. The best possible max-flow is then $\tau + c + p_1 = 3$. As the optimal max-flow is $\frac{5+\sqrt{7}}{3}$, we have a competitive ratio of $\frac{9}{5+\sqrt{7}} = \frac{5-\sqrt{7}}{2} > \rho$. This refutes the assumption on ρ. Thus the algorithm $\mathcal{A}$ must have scheduled the task i at time τ.

2. If $\mathcal{A}$ scheduled the task i on P_2 the adversary does not send other tasks. The best possible max-flow is then equal to $\frac{4+2\sqrt{7}}{3}$, which is even worse than the previous case. Consequently, algorithm $\mathcal{A}$ does not have another choice than to schedule the task i on P_1 in order to be able to respect its competitive ratio.

At time $\tau = \frac{4-\sqrt{7}}{3}$, the adversary sends another task, j. The best schedule would have been to send the first task on P_2 and the second on P_1 achieving a max-flow of $\max\{c+p_2, 2c+p_1-\tau\} = \max\{\frac{4+2\sqrt{7}}{3}, \frac{4+2\sqrt{7}}{3}\} = \frac{4+2\sqrt{7}}{3}$. We look at the assignment $\mathcal{A}$ made for j:

1. If j is sent on P_2, the best achievable max-flow is then $\max\{c+p_1, 2c+p_2-\tau\} = \max\{\frac{5+\sqrt{7}}{3}, 1+\sqrt{7}\} = 1+\sqrt{7}$, whereas the optimal is $\frac{4+2\sqrt{7}}{3}$. The competitive ratio is then $\frac{5-\sqrt{7}}{2} > \rho$.

2. If j is sent on P_1, the best possible max-flow is then $\max\{c+p_1, \max\{c+2p_1, 2c+p_1\}-\tau\} = \max\{\frac{5+\sqrt{7}}{3}, 1+\sqrt{7}\} = 1+\sqrt{7}$. The competitive ratio is therefore once again equal to $\frac{5-\sqrt{7}}{2} > \rho$.

□

3.3 Computation-homogeneous platforms

In this section, we have $p_j = p$ but processor links with different capacities. We order them, so that P_1 is the fastest communicating processor (c_1 is the smallest computing time c_i). Just as in Section 3.2, we can bound the competitive ratio of any deterministic algorithm:

Theorem 4. *There is no scheduling algorithm for the problem $P, MS \mid online,\ r_i,\ p_j = p,\ c_j \mid \max C_i$ whose competitive ratio ρ is strictly lower than $\frac{6}{5}$.*

Proof. Assume that there exists a deterministic on-line algorithm $\mathcal{A}$ whose competitive ratio is $\rho = \frac{6}{5} - \epsilon$, with $\epsilon > 0$. We will build a platform and an adversary to derive a contradiction. The platform is made up with two processors P_1 and P_2 such that $p_1 = p_2 = p = \max\{5, \frac{12}{25\epsilon}\}$, $c_1 = 1$ and $c_2 = \frac{p}{2}$.

Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ executes the task i, either on P_1 with a makespan at least equal to $1+p$, or on P_2 with a makespan at least equal to $\frac{3p}{2}$.

At time-step $\frac{p}{2}$, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and which one:

1. If $\mathcal{A}$ scheduled the task i on P_2 the adversary does not send other tasks. The best possible makespan is then $\frac{3p}{2}$. The optimal scheduling being of makespan $1+p$, we have a competitive ratio of
$$\rho \geq \frac{\frac{3p}{2}}{1+p} = \frac{3}{2} - \frac{3}{2(p+1)} > \frac{6}{5}$$
because $p \geq 5$ by assumption. This contradicts the hypothesis on ρ. Thus the algorithm $\mathcal{A}$ cannot schedule task i on P_2.

2. If $\mathcal{A}$ did not begin to send the task i, the adversary does not send other tasks. The best makespan that can be achieved is then equal to $\frac{p}{2} + (1+p) = 1 + \frac{3p}{2}$, which is even worse than the previous case. Consequently, the algorithm $\mathcal{A}$ does not have any other choice than to schedule task i on P_1.

At time-step $\frac{p}{2}$, the adversary sends three tasks, j, k and l. No schedule which executes three of the four tasks on the same processor can have a makespan lower than $1+3p$ (minimum duration of a communication and execution without delay of the three tasks). We now consider the schedules which compute two tasks on each processor. Since i is computed on P_1, we have three cases to study, depending upon which other task (j, k, or l) is computed on P_1:

1. If j is computed on P_1, at best we have:
   (a) Task i is sent to P_1 during the interval $[0, 1]$ and is computed during the interval $[1, 1+p]$.
   (b) Task j is sent to P_1 during the interval $[\frac{p}{2}, 1 + \frac{p}{2}]$ and is computed during the interval $[1+p, 1+2p]$.
   (c) Task k is sent to P_2 during the interval $[1+\frac{p}{2}, 1+p]$ and is computed during the interval $[1+p, 1+2p]$.
   (d) Task l is sent to P_2 during the interval $[1+p, 1+\frac{3p}{2}]$ and is computed during the interval $[1+2p, 1+3p]$.

   The makespan of this schedule is then $1+3p$.

2. If k is computed on P_1:
   (a) Task i is sent to P_1 during the interval $[0, 1]$ and is computed during the interval $[1, 1+p]$.
   (b) Task j is sent to P_2 during the interval $[\frac{p}{2}, p]$ and is computed during the interval $[p, 2p]$.
   (c) Task k is sent to P_1 during the interval $[p, 1+p]$ and is computed during the interval $[1+p, 1+2p]$.
   (d) Task l is sent to P_2 during the interval $[1+p, 1+\frac{3p}{2}]$ and is computed during the interval $[2p, 3p]$.

   The makespan of this scheduling is then $3p$.

3. If l is computed on P_1:
   (a) Task i is sent to P_1 during the interval $[0, 1]$ and is computed during the interval $[1, 1+p]$.
   (b) Task j is sent to P_2 during the interval $[\frac{p}{2}, p]$ and is computed during the interval $[p, 2p]$.
   (c) Task k is sent to P_2 during the interval $[p, \frac{3p}{2}]$ and is computed during the interval $[2p, 3p]$.
   (d) Task l is sent to P_1 during the interval $[\frac{3p}{2}, 1+\frac{3p}{2}]$ and is computed during the interval $[1+\frac{3p}{2}, 1+\frac{5p}{2}]$.

   The makespan of this schedule is then $3p$.

Consequently, the last two schedules are equivalent and are better than the first. Altogether, the best achievable makespan is $3p$. But a better schedule is obtained when computing i on P_2, then j on P_1, then k on P_2, and finally l on P_1. The makespan of the latter schedule is equal to $1+\frac{5p}{2}$. The competitive ratio of algorithm $\mathcal{A}$ is necessarily larger than the ratio of the best reachable makespan (namely $3p$) and the optimal makespan, which is not larger than $1+\frac{5p}{2}$. Consequently:

$$\rho \geq \frac{3p}{1+\frac{5p}{2}} = \frac{6}{5} - \frac{6}{5(5p+2)} > \frac{6}{5} - \frac{6}{25p} \geq \frac{6}{5} - \frac{\epsilon}{2}$$

which contradicts the assumption $\rho = \frac{6}{5} - \epsilon$ with $\epsilon > 0$. □

Theorem 5. *There is no scheduling algorithm for the problem $P, MS \mid online,\ r_i,\ p_j = p,\ c_j \mid \max\ (C_i - r_i)$ whose competitive ratio ρ is strictly lower than $\frac{5}{4}$.*

Proof. Assume that there exists a deterministic on-line algorithm $\mathcal{A}$ whose competitive ratio is $\rho = \frac{5}{4} - \epsilon$, with $\epsilon > 0$. We will build a platform and an adversary to derive a contradiction. The platform is made up with two processors P_1 and P_2, such that $p_1 = p_2 = p = 2c_2 - c_1$, and $c_1 = \epsilon$, $c_2 = 1$. Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ executes the task i either on P_1, with a max-flow at least equal to $c_1 + p$, or on P_2 with a max-flow at least equal to $c_2 + p$.

At time step $\tau = c_2 - c_1$, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and which one:

1. If $\mathcal{A}$ scheduled the task i on P_2 the adversary send no more task. The best possible max-flow is then $c_2 + p = 3 - \epsilon$. The optimal scheduling being of max-flow $c_1 + p = 2$, we have a competitive ratio of
$$\rho \geq \frac{c_2+p}{c_1+p} = \frac{3}{2} - \frac{\epsilon}{2} > \frac{5}{4} - \epsilon$$
Thus the algorithm $\mathcal{A}$ cannot schedule the task i on P_2.

2. If $\mathcal{A}$ did not begin to send the task i, the adversary does not send other tasks. The best max-flow that can be achieved is then equal to $\frac{\tau+c_1+p}{c_1+p} = \frac{3-\epsilon}{2}$, which is the same than the previous case. Consequently, the algorithm $\mathcal{A}$ does not have any choice but to schedule the task i on P_1.

At time-step τ, the adversary sends three tasks, j, k and l. No schedule which executes three of the four tasks on the same processor can have a max-flow lower than $max(c_1 + 3p - \tau, max(c_1, \tau) + c_1 + p + \max\{c_1, p\} - \tau) = 6 - 2\epsilon$. We now consider the schedules which compute two tasks on each processor. Since i is computed on P_1, we have three cases to study, depending upon which other task (j, k, or l) is computed on P_1:

1. If j is computed on P_1:
   (a) Task i is sent to P_1 and achieved a flow of $c_1 + p = 2$.
   (b) Task j is sent to P_1 and achieved a flow of $\max\{c_1 + 2p - \tau, \max\{\tau, c_1\} + c_1 + p - \tau\} = 3$.
   (c) Task k is sent to P_2 and achieved a flow of $\max\{\tau, c_1\} + c_1 + c_2 + p - \tau = 3$
   (d) Task l is sent to P_2 and achieved a flow of $\max\{\tau, c_1\} + c_1 + c_2 + p + max\{c_2, p\} - \tau = 5 - \epsilon$.

   The max-flow of this schedule is then $\max\{\tau, c_1\} + c_1 + c_2 + p + \max\{c_2, p\} - \tau = 5 - \epsilon$.

2. If k is computed on P_1:
   (a) Task i is sent to P_1 and achieved a flow of $c_1 + p = 2$.
   (b) Task j is sent to P_2 and achieved a flow of $\max\{\tau, c_1\} + c_2 + p - \tau = 3 - \epsilon$.
   (c) Task k is sent to P_1 and achieved a flow of $\max\{c_1 + 2p, \max\{\tau, c_1\} + c_2 + c_1 + p\} - \tau = 3$.
   (d) Task l is sent to P_2 and achieved a flow of $\max\{\max\{\tau, c_1\} + c_2 + 2p, \max\{\tau, c_1\} + 2c_2 + c_1 + p\} - \tau = 5 - 2\epsilon$.

The max-flow of this scheduling is then $\max\{\max\{\tau, c_1\} + c_2 + 2p, \max\{\tau, c_1\} + 2c_2 + c_1 + p\} - \tau = 5 - 2\epsilon$.

3. If l is computed on P_1:

   (a) Task i is sent to P_1 and achieved a flow of $c_1 + p = 2$.

   (b) task j is sent to P_2 and achieved a flow of $\max\{\tau, c_1) + c_2 + p\} = 3 - \epsilon$.

   (c) Task k is sent to P_2 and achieved a flow of $\max\{\max\{\tau, c_1\} + c_2 + 2p, \max\{\tau, c_1\} + 2c_2 + p\} = 5 - 2\epsilon$.

   (d) Task l is sent to P_1 and achieved a flow of $\max\{c_1 + 2p, \max\{\tau, c_1\} + 2c_2 + c_1 + p\} - \tau = 4$.

   The max-flow of this schedule is then $\max\{\max\{\tau, c_1\} + c_2 + 2p, \max\{\tau, c_1\} + 2c_2 + p\} = 5 - 2\epsilon$.

Consequently, the last two schedules are equivalent and are better than the first. Altogether, the best achievable max-flow is $5 - 2\epsilon$. But a better schedule is obtained when computing i on P_2, then j on P_1, then k on P_2, and finally l on P_1. The max-flow of the latter schedule is equal to $\max\{c_2 + p, \max\{\tau, c_2\} + c_1 + p - \tau, \max\{\max\{\tau, c_2\} + c_1 + c_2 + p, c_2 + 2p\} - \tau, \max\{\max\{\tau, c_2\} + 2c_1 + c_2 + p, \max\{\tau, c_2\} + c_1 + 2p\} - \tau\} = 4$. The competitive ratio of algorithm $\mathcal{A}$ is necessarily larger than the ratio of the best reachable max-flow (namely $5 - 2\epsilon$) and the optimal max-flow, which is not larger than 4. Consequently:

$$\rho \geq \frac{5 - 2\epsilon}{4} = \frac{5}{4} - \frac{\epsilon}{2}$$

which contradicts the assumption $\rho = \frac{5}{4} - \epsilon$ with $\epsilon > 0$.

□

Theorem 6. *There is no scheduling algorithm for the problem $P, MS \mid online, r_i, p_j = p, c_j \mid \sum (C_i - r_i)$ whose competitive ratio ρ is strictly lower than* 23/22.

Proof. Assume that there exists a deterministic on-line algorithm $\mathcal{A}$ whose competitive ratio is $\rho = 23/22 - \epsilon$, with $\epsilon > 0$. We will build a platform and an adversary to derive a contradiction. The platform is made up with two processors P_1 and P_2, such that $p_1 = p_2 = p = 3$, and $c_1 = 1$, $c_2 = 2$. Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ executes the task i either on P_1, with a max-flow at least equal to $c_1 + p$, or on P_2 with a max-flow at least equal to $c_2 + p$.

At time step $\tau = c_2$, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and which one:

1. If $\mathcal{A}$ scheduled the task i on P_2 the adversary sends no more task. The best possible sum-flow is then $c_2 + p = 5$. The optimal scheduling being of sum-flow $c_1 + p = 4$, we have a competitive ratio of
$$\rho \geq \frac{c_2 + p}{c_1 + p} = \frac{5}{4} > \frac{23}{22}.$$
   Thus the algorithm $\mathcal{A}$ cannot schedule the task i on P_2.

2. If $\mathcal{A}$ did not begin to send the task i, the adversary does not send other tasks. The best sum-flow that can be achieved is then equal to $\frac{\tau + c_1 + p}{c_1 + p} = \frac{6}{4}$, which is even worse than the previous case. Consequently, the algorithm $\mathcal{A}$ does not have any choice but to schedule the task i on P_1.

At time-step τ, the adversary sends three tasks, j, k, and l. We look at all the possible schedules, with i computed on P_1:

1. If all tasks are executed on P_1 the sum-flow is $(c_1 + p) + (\max\{c_1 + 2p, \max\{\tau, c_1\} + c_1 + p - \tau) + (\max\{c_1 + 3p, \max\{\tau, c_1\} + c_1 + p + \max\{c_1, p\} - \tau) + (\max\{c_1 + 4p, \max\{\tau, c_1\} + c_1 + p + 2\max\{c_1, p\} - \tau) = 28$.

2. If j is the only task executed on P_2 the sum-flow is $(c_1+p)+(\max\{\tau,c_1\}+c_2+p-\tau)+(\max\{c_1+2p,\max\{\tau,c_1\}+c_2+c_1+p\}-\tau)+(\max\{c_1+3p,\max\{\tau,c_1\}+c_2+c_1+p+\max\{c_1,p\}\}-\tau)=24$.

3. If k is the only task executed on P_2 the sum-flow is $(c_1+p)+(\max\{c_1+2p,\max\{\tau,c_1\}+c_1+p-\tau)+(\max\{\tau,c_1\}+c_1+c_2+p-\tau)+(\max\{c_1+3p,\max\{\tau,c_1\}+c_2+2c_1+p\}-\tau)=23$.

4. If l is the only task executed on P_2 the sum-flow is $(c_1+p)+(\max\{c_1+2p,\max\{\tau,c_1\}+c_1+p-\tau)+(\max\{c_1+3p,\max\{\tau,c_1\}+c_1+p+\max\{c_1,p\}-\tau)+(\max\{\tau,c_1\}+2c_1+c_2+p-\tau)=24$.

5. If j,k,l are executed on P_2 the sum-flow is $(c_1+p)+(\max\{\tau,c_1\}+c_2+p-\tau)+(\max\{\tau,c_1\}+c_2+p+\max\{c_2,p\}-\tau)+(\max\{\tau,c_1\}+c_2+p+2\max\{c_2,p\}-\tau)=28$.

We now consider the schedules which compute two tasks on each processor. Since i is computed on P_1, we have three cases to study, depending upon which other task (j, k, or l) is computed on P_1:

1. If j is computed on P_1 the sum-flow is: $(c_1+p)+(\max\{c_1+2p,\max\{\tau,c_1\}+c_1+p-\tau)+(\max\{\tau,c_1\}+c_1+c_2+p-\tau)+(\max\{\tau,c_1\}+c_1+c_2+p+\max\{c_2,p\}-\tau)=24$.

2. If k is computed on P_1: $(c_1+p)+(\max\{\tau,c_1\}+c_2+p-\tau)+(\max\{c_1+2p,\max\{\tau,c_1\}+c_2+c_1+p\}-\tau)+(\max\{\tau,c_1\}+c_2+p+\max\{c_1+c_2,p\}-\tau)=23$.

3. If l is computed on P_1: $(c_1+p)+(\max\{\tau,c_1\}+c_2+p-\tau)+(\max\{\tau,c_1\}+c_2+p+\max\{c_2,p\}-\tau)+(\max\{c_1+2p,\max\{\tau,c_1\}+2c_2+c_1+p\}-\tau)=25$.

Consequently, the best achievable sum-flow is 23. But a better schedule is obtained when computing i on P_2, then j on P_1, then k on P_2, and finally l on P_1. The sum-flow of the latter schedule is equal to $(c_2+p)+(\max\{\tau,c_2\}+c_1+p-\tau)+(\max\{\max\{\tau,c_2\}+c_1+c_2+p,c_2+2p\}-\tau+\max\{\max\{\tau,c_2\}+2c_1+c_2+p,\max\{\tau,c_2\}+c_1+2p\}-\tau\}=22$. The competitive ratio of algorithm $\mathcal{A}$ is necessarily larger than the ratio of the best reachable sum-flow (namely 23) and the optimal sum-flow, which is not larger than 22. Consequently:

$$\rho\geq\frac{23}{22}$$

which contradicts the assumption $\rho=\frac{23}{22}-\epsilon$ with $\epsilon>0$. □

3.4 Fully heterogeneous platforms

Just as in the previous two sections, we can bound the competitive ratio of any deterministic algorithm. As expected, having both heterogeneous computations and communications increases the difficulty of the problem.

Theorem 7. *There is no scheduling algorithm for the problem $Q,MS\mid online, r_i, p_j, c_j\mid\max C_i$ whose competitive ratio ρ is strictly lower than $\frac{1+\sqrt{3}}{2}$.*

Proof. Assume that there exists a deterministic on-line algorithm $\mathcal{A}$ whose competitive ratio is $\rho=\frac{1+\sqrt{3}}{2}-\epsilon$, with $\epsilon>0$. We will build a platform and an adversary to derive a contradiction. The platform is made up with three processors P_1, P_2, and P_3 such that $p_1=\epsilon$, $p_2=p_3=1+\sqrt{3}$, $c_1=1+\sqrt{3}$ and $c_2=c_3=1$.

Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ executes the task i, either on P_1 with a makespan at least equal to $c_1+p_1=1+\sqrt{3}+\epsilon$, or on P_2 or P_3, with a makespan at least equal to $c_2+p_2=c_3+p_3=2+\sqrt{3}$.

At time-step 1, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and which one:

1. If $\mathcal{A}$ scheduled the task i on P_2 or P_3, the adversary does not send any other task. The best possible makespan is then $c_2 + p_2 = c_3 + p_3 = 2 + \sqrt{3}$. The optimal scheduling being of makespan $c_1 + p_1 = 1 + \sqrt{3} + \epsilon$, we have a competitive ratio of:

$$\rho \geq \frac{2+\sqrt{3}}{1+\sqrt{3}+\epsilon} > \frac{1+\sqrt{3}}{2} - \epsilon,$$

   because $\epsilon > 0$ by assumption. This contradicts the hypothesis on ρ. Thus the algorithm $\mathcal{A}$ cannot schedule task i on P_2 or P_3.

2. If $\mathcal{A}$ did not begin to send the task i, the adversary does not send any other task. The best makespan that can be achieved is then equal to $1 + c_1 + p_1 = 2 + \sqrt{3} + \epsilon$, which is even worse than the previous case. Consequently, the algorithm $\mathcal{A}$ does not have any other choice than to schedule task i on P_1.

Then, at time-step $\tau = 1$, the adversary sends two tasks, j and k. We consider all the scheduling possibilities:

- j and k are scheduled on P_1. Then the best achievable makespan is:

$$\max\{c_1 + 3p_1, \max\{c_1, \tau\} + c_1 + p_1 + \max\{c_1, p_1\}\} = 3(1+\sqrt{3}) + \epsilon,$$

  as $\epsilon < \frac{1+\sqrt{3}}{2}$.

- The first of the two jobs, j and k, to be scheduled is scheduled on P_2 (or P_3) and the other one on P_1. Then, the best achievable makespan is:

$$\begin{aligned}&\max\{c_1 + p_1, \max\{c_1, \tau\} + c_2 + p_2, \max\{c_1 + 2p_1, \max\{c_1, \tau\} + c_2 + c_1 + p_1\}\}\\ &\quad = \max\{1 + \sqrt{3} + \epsilon, 3 + 2\sqrt{3}, \max\{1 + \sqrt{3} + 2\epsilon, 3 + 2\sqrt{3} + \epsilon\}\\ &\quad = 3 + 2\sqrt{3} + \epsilon.\end{aligned}$$

- The first of the two jobs j and k to be scheduled is scheduled on P_1 and the other one on P_2 (or P_3). Then, the best achievable makespan is:

$$\begin{aligned}&\max\{c_1 + p_1, \max\{\max\{c_1, \tau\} + c_1 + p_1, c_1 + 2p_1\}, \max\{c_1, \tau\} + c_1 + c_2 + p_2\}\\ &\quad = \max\{1 + \sqrt{3} + \epsilon, \max\{2 + 2\sqrt{3} + \epsilon, 1 + \sqrt{3} + 2\epsilon\}, 4 + 3\sqrt{3}\}\\ &\quad = 4 + 3\sqrt{3}.\end{aligned}$$

- One of the jobs j and k is scheduled on P_2 and the other one on P_3.

$$\begin{aligned}&\max\{c_1 + p_1, \max\{c_1, \tau\} + c_2 + p_2, \max\{c_1, \tau\} + c_2 + c_3 + p_3\}\\ &\quad = \max\{1 + \sqrt{3} + \epsilon, 3 + 2\sqrt{3}, 4 + 2\sqrt{3}\}\\ &\quad = 4 + 2\sqrt{3}.\end{aligned}$$

- The case where j and k are both executed on P_2, or both on P_3, leads to an even worse makespan than the previous case. Therefore, we do not need to study it.

Therefore, the best achievable makespan for $\mathcal{A}$ is: $3 + 2\sqrt{3} + \epsilon$ (as $\epsilon < 1$). However, we could have scheduled i on P_2, j on P_3, and then k on P_1, thus achieving a makespan of:

$$\begin{aligned}&\max\{c_2 + p_2, \max\{c_2, \tau\} + c_3 + p_3, \max\{c_2, \tau\} + c_3 + c_1 + p_1\}\\ &\quad = \max\{2 + \sqrt{3}, \max\{1, 1\} + 2 + \sqrt{3}, \max\{1, 1\} + 2 + \sqrt{3} + \epsilon, \}\\ &\quad = 3 + \sqrt{3} + \epsilon.\end{aligned}$$

Therefore, we have a competitive ratio of:

$$\rho \geq \frac{3+2\sqrt{3}+\epsilon}{3+\sqrt{3}+\epsilon} > \frac{1+\sqrt{3}}{2} - \epsilon.$$

This contradicts the hypothesis on ρ. □

Theorem 8. *There is no scheduling algorithm for the problem $Q, MS \mid online, r_i, p_j, c_j \mid \sum(C_i - r_i)$ whose competitive ratio ρ is strictly lower than $\frac{\sqrt{13}-1}{2}$.*

Proof. Assume that there exists a deterministic on-line algorithm $\mathcal{A}$ whose competitive ratio is $\rho = \frac{\sqrt{13}-1}{2} - \epsilon$, with $\epsilon > 0$. We will build a platform and an adversary to derive a contradiction. The platform is made up of three processors P_1, P_2, and P_3 such that $p_1 = \epsilon$, $p_2 = p_3 = \tau + c_1 - 1$, $c_2 = c_3 = 1$, and $\tau = \frac{\sqrt{52c_1^2+12c_1+1}-(6c_1+1)}{4}$. Note that $\tau < c_1$ and that:

$$\lim_{c_1 \to +\infty} \frac{\tau}{c_1} = \frac{\sqrt{13}-3}{2}$$

Therefore the exists a value N_0 such that:

$$c_1 \geq N_0 \quad \Rightarrow \quad c_1 > \epsilon \text{ and } \tau > \epsilon.$$

The value of c_1 will be defined later on. For now, we just assume that $c_1 \geq N_0$.

Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ executes the task i, either on P_1 with a sum-flow at least equal to $c_1 + p_1$, or on P_2 or P_3, with a sum-flow at least equal to $c_2 + p_2 = c_3 + p_3 = \tau + c_1$.

At time-step τ, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and which one:

1. If $\mathcal{A}$ scheduled the task i on P_2 or P_3, the adversary does not send any other task. The best possible sum-flow is then $c_2 + p_2 = c_3 + p_3 = \tau + c_1$. The optimal scheduling being of sum-flow $c_1 + p_1 = c_1 + \epsilon$, we have a competitive ratio of:

   $$\rho \geq \frac{\tau + c_1}{c_1 + \epsilon}.$$

   However,

   $$\lim_{c_1 \to +\infty} \frac{\tau + c_1}{c_1 + \epsilon} = \frac{\frac{\sqrt{13}-3}{2}c_1 + c_1}{c_1} = \frac{\sqrt{13}-1}{2}.$$

   Therefore, there exists a value N_1 such that:

   $$c_1 \geq N_1 \quad \Rightarrow \quad \frac{\tau + c_1}{c_1 + \epsilon} > \frac{\sqrt{13}-1}{2} - \frac{\epsilon}{2},$$

   which contradicts the hypothesis on ρ. We will now suppose that $c_1 \geq \max\{N_0, N_1\}$. Then the algorithm $\mathcal{A}$ cannot schedule task i on P_2 or P_3.

2. If $\mathcal{A}$ did not begin to send the task i, the adversary does not send any other task. The best sum-flow that can be achieved is then equal to $\tau + c_1 + p_1 = \tau + c_1 + \epsilon$, which is even worse than the previous case. Consequently, algorithm $\mathcal{A}$ does not have any other choice than to schedule task i on P_1.

Then, at time-step τ, the adversary sends two tasks, j and k. We consider all the scheduling possibilities:

- Tasks j and k are scheduled on P_1. Then the best achievable sum-flow is:

$$\begin{aligned}(c_1+p_1)+(\max\{\max\{c_1,\tau\}+c_1+p_1,c_1+2p_1\}-\tau)\\ +(\max\{\max\{c_1,\tau\}+c_1+p_1+\max\{c_1,p_1\},c_1+3p_1\}-\tau)\\ =6c_1+3p_1-2\tau\\ =6c_1-2\tau+3\epsilon\end{aligned}$$

  as $p_1<c_1$.

- The first of the two jobs, j and k, to be scheduled is scheduled on P_2 (or P_3) and the other one on P_1. Then, the best achievable sum-flow is:

$$\begin{aligned}(c_1+p_1)+((\max\{c_1,\tau\}+c_2+p_2)-\tau)\\ +(\max\{\max\{c_1,\tau\}+c_2+c_1+p_1,c_1+2p_1\}-\tau)\\ =4c_1+2+2p_1+p_2-2\tau\\ =5c_1-\tau+1+2\epsilon\end{aligned}$$

- The first of the two jobs j and k to be scheduled is scheduled on P_1 and the other one on P_2 (or P_3). Then, the best achievable sum-flow is:

$$\begin{aligned}(c_1+p_1)+(\max\{\max\{c_1,\tau\}+c_1+p_1,c_1+2p_1\}-\tau)\\ +((\max\{c_1,\tau\}+c_1+c_2+p_2)-\tau)\\ =5c_1+c_2+2p_1+p_2-2\tau\\ =6c_1-\tau+2\epsilon\end{aligned}$$

- One of the jobs j and k is scheduled on P_2 and the other one on P_3.

$$\begin{aligned}(c_1+p_1)+((\max\{c_1,\tau\}+c_2+p_2)-\tau)+((\max\{c_1,\tau\}+c_2+c_3+p_3)-\tau)\\ =3c_1+3c_2+p_1+2p_2-2\tau\\ =5c_1+1+\epsilon\end{aligned}$$

- The case where j and k are both executed on P_2, or both on P_3, leads to an even worse sum-flow than the previous case. Therefore, we do not need to study it.

Therefore, the best achievable sum-flow for $\mathcal{A}$ is: $5c_1-\tau+1+2\epsilon$ (as $c_1>\tau>\epsilon$). However, we could have scheduled i on P_2, j on P_3, and then k on P_1, thus achieving a sum-flow of:

$$\begin{aligned}(c_2+p_2)+((\max\{c_2,\tau\}+c_3+p_3)-\tau)+((\max\{c_2,\tau\}+c_3+c_1+p_1)-\tau)\\ =c_1+3c_2+p_1+2p_2\\ =3c_1+2\tau+1+\epsilon.\end{aligned}$$

Therefore, we have a competitive ratio of:

$$\rho\geq\frac{5c_1-\tau+1+2\epsilon}{3c_1+2\tau+1+\epsilon}$$

However,

$$\lim_{c_1\to+\infty}\frac{5c_1-\tau+1+2\epsilon}{3c_1+2\tau+1+\epsilon}=\lim_{c_1\to+\infty}\frac{5c_1-\frac{\sqrt{13}-3}{2}c_1}{3c_1+2\frac{\sqrt{13}-3}{2}c_1}=\frac{\sqrt{13}-1}{2}$$

Therefore, there exists a value N_2 such that:

$$c_1\geq N_2\quad\Rightarrow\quad\frac{5c_1-\tau+1+2\epsilon}{3c_1+2\tau+1+\epsilon}>\frac{\sqrt{13}-1}{2}-\frac{\epsilon}{2},$$

which contradicts the hypothesis on ρ.

Therefore, if we initially choose c_1 greater than $\max\{N_0, N_1, N_2\}$, we obtain the desired contradiction. □

Theorem 9. *There is no scheduling algorithm for the problem $Q, MS \mid online, r_i, p_j, c_j \mid \max(C_i - r_i)$ whose competitive ratio ρ is strictly lower than $\sqrt{2}$.*

Proof. Assume that there exists a deterministic on-line algorithm $\mathcal{A}$ whose competitive ratio is $\rho = \sqrt{2} - \epsilon$, with $\epsilon > 0$. We will build a platform and an adversary to derive a contradiction. The platform is made up of three processors P_1, P_2, and P_3 such that $p_1 = \epsilon$, $p_2 = p_3 = \sqrt{2}c_1 - 1$, $c_1 = 2(1 + \sqrt{2})$, $c_2 = c_3 = 1$, and $\tau = (\sqrt{2} - 1)c_1$. Note that $\tau < c_1$, and that $c_1 + p_1 < p_2$.

Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ executes the task i, either on P_1 with a max-flow at least equal to $c_1 + p_1 = c_1 + \epsilon$, or on P_2 or P_3, with a max-flow at least equal to $c_2 + p_2 = c_3 + p_3 = \sqrt{2}c_1$.

At time-step τ, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and which one:

1. If $\mathcal{A}$ scheduled the task i on P_2 or P_3, the adversary does not send any other task. The best possible max-flow is then $c_2 + p_2 = c_3 + p_3 = \sqrt{2}c_1$. The optimal scheduling being of max-flow $c_1 + p_1 = c_1 + \epsilon$, we have a competitive ratio of:
$$\rho \geq \frac{c_2 + p_2}{c_1 + p_1} = \frac{\sqrt{2}c_1}{c_1 + \epsilon} > \sqrt{2} - \epsilon,$$
as $c_1 > \sqrt{2}$. This contradicts the hypothesis on ρ. Thus the algorithm $\mathcal{A}$ cannot schedule task i on P_2 or P_3.

2. If $\mathcal{A}$ did not begin to send the task i, the adversary does not send any other task. The best max-flow that can be achieved is then equal to $\tau + c_1 + p_1 = \sqrt{2}c_1 + \epsilon$, which is even worse than the previous case. Consequently, algorithm $\mathcal{A}$ does not have any other choice than to schedule task i on P_1.

Then, at time-step τ, the adversary sends two tasks, j and k. We consider all the scheduling possibilities:

- j and k are scheduled on P_1. Then the best achievable max-flow is:
$$\begin{aligned}\max\{c_1 + p_1, \max\{\max\{c_1, \tau\} + c_1 + p_1, c_1 + 2p_1\} - \tau,\\ \max\{\max\{c_1, \tau\} + c_1 + p_1 + \max\{c_1, p_1\}, c_1 + 3p_1\} - \tau\}\\ = 3c_1 + p_1 - \tau\\ = (4 - \sqrt{2})c_1 + \epsilon\end{aligned}$$
as $p_1 < c_1$.

- The first of the two jobs, j and k, to be scheduled is scheduled on P_2 (or P_3) and the other one on P_1. Then, the best achievable max-flow is:
$$\begin{aligned}\max\{c_1 + p_1, (\max\{c_1, \tau\} + c_2 + p_2) - \tau,\\ \max\{\max\{c_1, \tau\} + c_2 + c_1 + p_1, c_1 + 2p_1\} - \tau\}\\ = c_1 + c_2 - \tau + \max\{p_2, c_1 + p_1\}\\ = 2c_1\end{aligned}$$

- The first of the two jobs j and k to be scheduled is scheduled on P_1 and the other one on P_2 (or P_3). Then, the best achievable max-flow is:

$$\begin{aligned}\max\{c_1+p_1, \max\{\max\{c_1,\tau\}+c_1+p_1, c_1+2p_1\}-\tau,\\ (\max\{c_1,\tau\}+c_1+c_2+p_2)-\tau\}\\ =2c_1-\tau+\max\{p_1,c_2+p_2\}\\ =3c_1\end{aligned}$$

- One of the jobs j and k is scheduled on P_2 and the other one on P_3.

$$\begin{aligned}\max\{c_1+p_1,(\max\{c_1,\tau\}+c_2+p_2)-\tau,(\max\{c_1,\tau\}+c_2+c_3+p_3)-\tau\}\\ =c_1+2c_2+p_2-\tau\\ =2c_1+1\end{aligned}$$

- The case where j and k are both executed on P_2, or both on P_3, leads to an even worse max-flow than the previous case. Therefore, we do not need to study it.

Therefore, the best achievable max-flow for $\mathcal{A}$ is: $2c_1$. However, we could have scheduled i on P_2, j on P_3, and then k on P_1, thus achieving a max-flow of:

$$\begin{aligned}\max\{c_2+p_2,(\max\{c_2,\tau\}+c_3+p_3)-\tau,(\max\{c_2,\tau\}+c_3+c_1+p_1)-\tau\}\\ =\max\{c_2,\tau\}+c_2+\max\{p_2,c_1+p_1\}-\tau\\ =\sqrt{2}c_1\end{aligned}$$

Therefore, we have a competitive ratio of:

$$\rho \geq \frac{2c_1}{\sqrt{2}} = \sqrt{2},$$

which contradicts the hypothesis on ρ. □

4 MPI experiments

To complement the previous theoretical results, we looked at some efficients on-line algorithms, and we compared them experimentally on different kind of platforms. In particular, we include in the comparison the two new heuristics of [23], which were specifically designed to work well on communication-homogeneous and on computation-homogeneous platforms respectively.

4.1 The algorithms

We describe here the different algorithms used in the practical tests:

1. ***SRPT*:** *Shortest Remaining Processing Time* is a well known algorithm on a platform where preemption is allowed, or with task of different size. But in our case, with identical tasks and no preemption, its behavior is the following: it sends a task to the fastest free slave; if no slave is currently free, it waits for the first slave to finish its task, and then sends it a new one.

2. ***LS*:** *List Scheduling* can be viewed as the static version of *SRPT*. It uses its knowledge of the system and sends a task as soon as possible to the slave that would finish it first, according to the current load estimation (the number of tasks already waiting for execution on the slave).

3. ***RR*** **:** *Round Robin* is the simplest algorithm. It simply sends a task to each slave one by one, according to a prescribed ordering. This ordering first choose the slave with the smallest $p_i + c_i$, then the slave with the second smallest value, etc.

4. ***RRC*** has the same behavior than *RR*, but uses a different ordering: it sends the tasks starting from the slave with the smallest c_i up to the slave with the largest one.

5. ***RRP*** has the same behavior than *RR*, but uses yet another ordering: it sends the tasks starting from the slave with the smallest p_i up to the slave with the largest one.

6. ***SLJF*** **:** *Scheduling the Last Job First* is one of the two algorithms we built. We proved in [23] that this algorithm is optimal to minimize the makespan on a communication-homogeneous platform, as soon as it knows the total number of tasks, even with release-dates. As it name says, it calculates, before scheduling the first task, the assignment of all tasks, starting with the last one.

7. ***SLJFWC*** **:** *Scheduling the Last Job First With Communication* is a variant of *SLJF* conceived to work on processor-homogeneous platforms. See [23] for a detailed description.

The last two algorithms were initially built to work with off-line models, because they need to know the total number of tasks to perform at their best. So we transform them for on-line execution as follows: at the beginning, we start to compute the assignment of a certain number of tasks (the greater this number, the better the final assignment), and start to send the first tasks to their assigned processors. Once the last assignment is done, we continue to send the remaining tasks, each task being sent to the processor that would finish it the earliest. In other words, the last tasks are assigned using a list scheduling policy.

4.2 The experimental platform

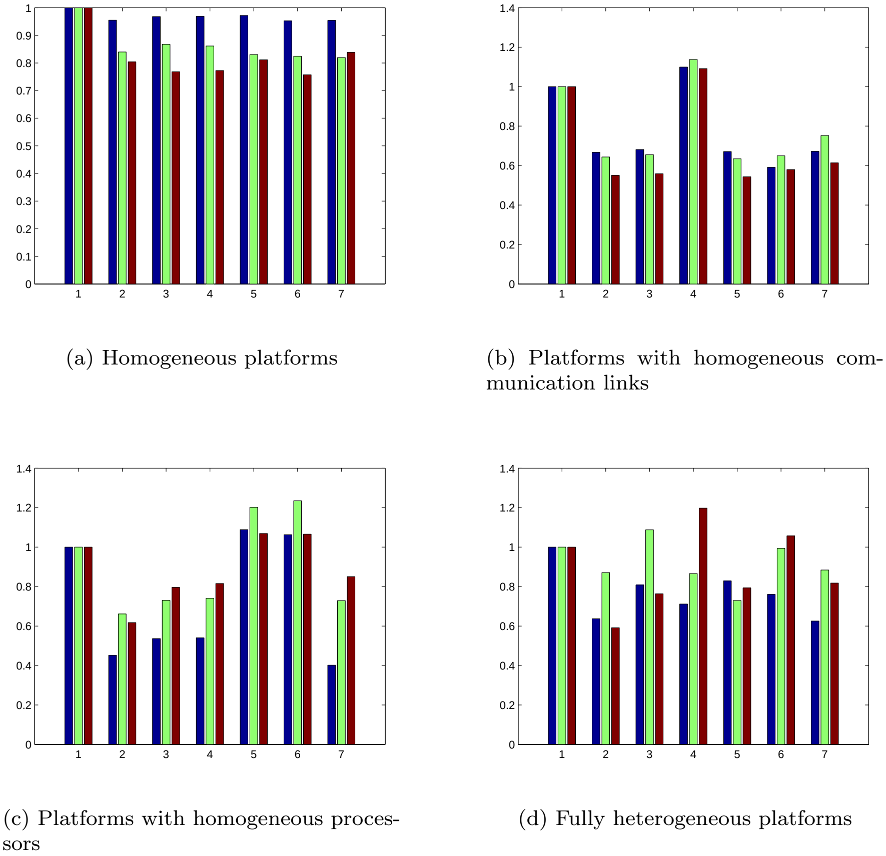
We build a small heterogeneous master-slave platform with five different computers, connected to each other by a fast Ethernet switch (100 Mbit/s). The five machines are all different, both in terms of CPU speed and in the amount of available memory. The heterogeneity of the communication links is mainly due to the differences between the network cards. Each task will be a matrix, and each slave will have to calculate the determinant of the matrices that it will receive. Whenever needed, we play with matrix sizes so as to achieve more heterogeneity or on the contrary some homogeneity in the CPU speeds or communication bandwidths. We proceed as follows: in a first step, we send one single matrix to each slave one after a other, and we calculate the time needed to send this matrix and to calculate its determinant on each slave. Thus, we obtain an estimation of c_i and p_i, according to the matrix size. Then we determine the number of times this matrix should be sent (n_{c_i}) and the number of times its determinant should be calculated (n_{p_i}) on each slave in order to modify the platform characteristics so as to reach the desired level of heterogeneity. Then, a task (matrix) assigned on P_i will actually be sent n_{c_i} times to P_i (so that $c_i \leftarrow n_{c_i}.c_i$), and its determinant will actually be calculated n_{p_i} times by P_i (so that $p_i \leftarrow n_{p_i}.p_i$).

The experiments are as follows: for each diagram, we create ten random platforms, possibly with one prescribed property (such as homogeneous links or processors) and we execute the different algorithms on it. Our platforms are composed with five machines P_i with c_i between 0.01 s and 1 s, and p_i between 0.1 s and 8 s. Once the platform is created, we send one thousand tasks on it, and we calculate the makespan, sum-flow, and max-flow obtained by each algorithm. After having executed all algorithms on the ten platforms, we calculate the average makespan, sum-flow, and max-flow. The following section shows the different results that have been obtained.

4.3 Results

In the following figures, we compare the seven algorithms: *SRPT*, *List Scheduling*, the three *Round-Robin* variants, *SLJF*, and *SLJFWC*. For each algorithm we plot its normalized makespan, sum-flow, and max-flow, which are represented in this order, from left to right. We normalize everything to the performance of *SRPT*, whose makespan, max-flow and sum-flow are therefore set equal to 1.

First of all, we consider fully homogeneous platforms. Figure 1(a) shows that all static algorithms perform equally well on such platforms, and exhibit better performance than the dynamic heuristic *SRPT*. On communication-homogeneous platforms (Figure 1(b)), we see that *RRC*, which does not take processor heterogeneity into account, performs significantly worse than the others; we also observe that *SLJF* is the best approach for makespan minimization. On computation-homogeneous platforms (Figure 1(c)), we see that *RRP* and *SLJF*, which do not take communication heterogeneity into account, perform significantly worse than the others; we also observe that *SLJFWC* is the best approach for makespan minimization. Finally, on fully heterogeneous platforms (Figure 1(d)), the best algorithms are *LS* and *SLJFWC*. Moreover, we see that algorithms taking communication delays into account actually perform better.



(a) Homogeneous platforms

(b) Platforms with homogeneous communication links

(c) Platforms with homogeneous processors

(d) Fully heterogeneous platforms

Figure 1: Comparison of the seven algorithms on different platforms.

In another experiment, we try to test the robustness of the algorithms. We randomly change the size of the matrix sent by the master at each round, by a factor of up to 10%. Figure 2 represents the average makespan (respectively sum-flow and max-flow) compared to the one obtained on the same platform, but with identical size tasks. Thus, we see that our algorithms are quite robust for makespan minimization problems, but not as much for sum-flow or max-flow problems.

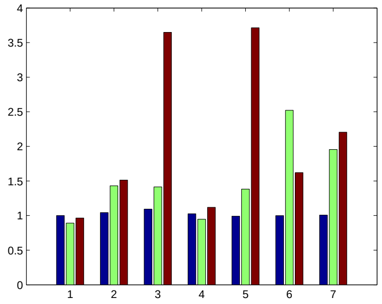


Figure 2: Assessing the robustness of the algorithms.

5 Related work

We classify several related papers along the following four main lines:

Models for heterogeneous platforms– In the literature, one-port models come in two variants. In the unidirectional variant, a processor cannot be involved in more than one communication at a given time-step, either a send or a receive. In the bidirectional model, a processor can send and receive in parallel, but at most from a given neighbor in each direction. In both variants, if P_u sends a message to P_v, both P_u and P_v are blocked throughout the communication.

The bidirectional one-port model is used by Bhat et al [7, 8] for fixed-size messages. They advocate its use because "current hardware and software do not easily enable multiple messages to be transmitted simultaneously". Even if non-blocking multi-threaded communication libraries allow for initiating multiple send and receive operations, they claim that all these operations "are eventually serialized by the single hardware port to the network". Experimental evidence of this fact has recently been reported by Saif and Parashar [26], who report that asynchronous MPI sends get serialized as soon as message sizes exceed a few megabytes. Their results hold for two popular MPI implementations, MPICH on Linux clusters and IBM MPI on the SP2.

The one-port model fully accounts for the heterogeneity of the platform, as each link has a different bandwidth. It generalizes a simpler model studied by Banikazemi et al. [2], Liu [20], and Khuller and Kim [16]. In this simpler model, the communication time only depends on the sender, not on the receiver: in other words, the communication speed from a processor to all its neighbors is the same.

Finally, we note that some papers [3, 4] depart from the one-port model as they allow a sending processor to initiate another communication while a previous one is still on-going on the network. However, such models insist that there is an overhead time to pay before being engaged in another operation, so there are not allowing for fully simultaneous communications.

Task graph scheduling– Task graph scheduling is usually studied using the so-called *macro-dataflow* model [21, 28, 11, 12], whose major flaw is that communication resources are not limited. In this model, a processor can send (or receive) any number of messages in parallel, hence an unlimited number of communication ports is assumed (this explains the name *macro-dataflow* for the model). Also, the number of messages that can simultaneously circulate between processors is not bounded, hence an unlimited number of communications can simultaneously occur on a given link. In other words, the communication network is assumed to be contention-free, which of course is not realistic as soon as the processor number exceeds a few units. More recent papers [30, 22, 25, 5, 6, 29] take communication resources into account.

Hollermann et al. [14] and Hsu et al. [15] introduce the following model for task graph scheduling: each processor can either send or receive a message at a given time-step (bidirectional communication is not possible); also, there is a fixed latency between the initiation of the communication by the sender and the beginning of the reception by the receiver. Still, the model is rather close to the one-port model discussed in this paper.

On-line scheduling– A good survey of on-line scheduling can be found in [27, 24]. Two papers focus on the problem of on-line scheduling for master-slaves platforms. In [18], Leung and Zhao proposed several competitive algorithms minimizing the total completion time on a master-slave platform, with or without pre- and post-processing. In [19], the same authors studied the complexity of minimizing the makespan or the total response time, and proposed some heuristics. However, none of these works take into consideration communication costs. To the best of our knowledge, the only previously known results for on-line problems with communication costs are those reported in our former work [23]; in the current paper, we have dramatically improved several of the competitive ratios given in [23] and we have added new ones.

6 Conclusion

In this paper, we have dealt with the problem of on-line scheduling independent, same-size tasks on master-slave platforms. We enforce the one-port model, and we study the impact of heterogeneity on the performance of scheduling algorithms.

The major contribution of this paper lies on the theoretical side, and is well summarized by Table 1. We have provided a comprehensive set of lower bounds for the competitive ratio of any deterministic scheduling algorithm, for each source of heterogeneity and for each target objective function. An important direction for future work would be to see which of these bounds can be met, if any, and to design the corresponding approximation algorithms.

On the practical side, we have to widen the scope of the MPI experiments. A detailed comparison of all the heuristics that we have implemented needs to be conducted on significantly larger platforms (with several tens of slaves). Such a comparison would, we believe, further demonstrate the superiority of those heuristics which fully take into account the relative capacity of the communication links.

References

[1] M. Adler, Y. Gong, and A. L. Rosenberg. Optimal sharing of bags of tasks in heterogeneous clusters. In *15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'03)*, pages 1–10. ACM Press, 2003.

[2] M. Banikazemi, V. Moorthy, and D. K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of the 27th International Conference on Parallel Processing (ICPP'98)*. IEEE Computer Society Press, 1998.

[3] M. Banikazemi, J. Sampathkumar, S. Prabhu, D.K. Panda, and P. Sadayappan. Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations. In *HCW'99, the 8th Heterogeneous Computing Workshop*, pages 125–133. IEEE Computer Society Press, 1999.

[4] Amotz Bar-Noy, Sudipto Guha, Joseph (Seffi) Naor, and Baruch Schieber. Message multicasting in heterogeneous networks. *SIAM Journal on Computing*, 30(2):347–358, 2000.

[5] Olivier Beaumont, Vincent Boudet, and Yves Robert. A realistic model and an efficient heuristic for scheduling with heterogeneous processors. In *HCW'2002, the 11th Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2002.

[6] Olivier Beaumont, Arnaud Legrand, and Yves Robert. A polynomial-time algorithm for allocating independent tasks on heterogeneous fork-graphs. In *ISCIS XVII, Seventeenth International Symposium On Computer and Information Sciences*, pages 115–119. CRC Press, 2002.

[7] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *ICDCS'99 19th International Conference on Distributed Computing Systems*, pages 15–24. IEEE Computer Society Press, 1999.

[8] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63:251–263, 2003.

[9] J. Blazewicz, J.K. Lenstra, and A.H. Kan. Scheduling subject to resource constraints. *Discrete Applied Mathematics*, 5:11–23, 1983.

[10] H. Casanova and F. Berman. *Grid Computing: Making The Global Infrastructure a Reality*, chapter Parameter Sweeps on the Grid with APST. John Wiley, 2003. Hey, A. and Berman, F. and Fox, G., editors.

[11] P. Chrétienne, E. G. Coffman Jr., J. K. Lenstra, and Z. Liu, editors. *Scheduling Theory and its Applications*. John Wiley and Sons, 1995.

[12] H. El-Rewini, H. H. Ali, and T. G. Lewis. Task scheduling in multiprocessing systems. *Computer*, 28(12):27–37, 1995.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[14] L. Hollermann, T. S. Hsu, D. R. Lopez, and K. Vertanen. Scheduling problems in a practical allocation model. *J. Combinatorial Optimization*, 1(2):129–149, 1997.

[15] T. S. Hsu, J. C. Lee, D. R. Lopez, and W. A. Royce. Task allocation on a network of processors. *IEEE Trans. Computers*, 49(12):1339–1353, 2000.

[16] S. Khuller and Y.A. Kim. On broadcasting in heterogenous networks. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1011–1020. Society for Industrial and Applied Mathematics, 2004.

[17] J.K. Lenstra, R. Graham, E. Lawler, and A.H. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[18] Joseph Y-T. Leung and Hairong Zhao. Minimizing total completion time in master-slave systems, 2004. Available at `http://web.njit.edu/~hz2/papers/masterslave-ieee.pdf`.

[19] Joseph Y-T. Leung and Hairong Zhao. Minimizing mean flowtime and makespan on master-slave systems. *J. Parallel and Distributed Computing*, 65(7):843–856, 2005.

[20] P. Liu. Broadcast scheduling optimization for heterogeneous cluster systems. *Journal of Algorithms*, 42(1):135–152, 2002.

[21] M. G. Norman and P. Thanisch. Models of machines and computation for mapping in multicomputers. *ACM Computing Surveys*, 25(3):103–117, 1993.

[22] J. M. Orduna, F. Silla, and J. Duato. A new task mapping technique for communication-aware scheduling strategies. In T. M. Pinkston, editor, *Workshop for Scheduling and Resource Management for Cluster Computing (ICPP'01)*, pages 349–354. IEEE Computer Society Press, 2001.

[23] Jean-François Pineau, Yves Robert, and Frédéric Vivien. Off-line and on-line scheduling on heterogeneous master-slave platforms. Research Report 2005-31, LIP, ENS Lyon, France, July 2005. Available at graal.ens-lyon.fr/~yrobert/.

[24] Irk Pruhs, Jiri Sgall, and Eric Torng. On-line scheduling. In J. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pages 15.1–15.43. CRC Press, 2004.

[25] C. Roig, A. Ripoll, M. A. Senar, F. Guirado, and E. Luque. Improving static scheduling using inter-task concurrency measures. In T. M. Pinkston, editor, *Workshop for Scheduling and Resource Management for Cluster Computing (ICPP'01)*, pages 375–381. IEEE Computer Society Press, 2001.

[26] T. Saif and M. Parashar. Understanding the behavior and performance of non-blocking communications in MPI. In *Proceedings of Euro-Par 2004: Parallel Processing*, LNCS 3149, pages 173–182. Springer, 2004.

[27] J. Sgall. On line scheduling-a survey. In *On-Line Algorithms*, Lecture Notes in Computer Science 1442, pages 196–231. Springer-Verlag, Berlin, 1998.

[28] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems.* IEEE Computer Science Press, 1995.

[29] Oliver Sinnen and Leonel Sousa. Communication contention in task scheduling. *IEEE Trans. Parallel Distributed Systems*, 16(6):503–515, 2004.

[30] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li. Minimizing the aplication execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):857–871, 1997.