Laboratoire de l'Informatique du Parallélisme

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

Independent and Divisible Task Scheduling on Heterogeneous Star-shaped Platforms with Limited Memory

Olivier Beaumont,
Arnaud Legrand,
Loris Marchal,
Yves Robert

April 2004

Research Report N° 2004-22

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`

Independent and Divisible Task Scheduling on Heterogeneous Star-shaped Platforms with Limited Memory

Olivier Beaumont, Arnaud Legrand, Loris Marchal, Yves Robert

April 2004

Abstract

In this paper, we consider the problem of allocating and scheduling a collection of independent, equal-sized tasks on heterogeneous star-shaped platforms. We also address the same problem for divisible tasks. For both cases, we take memory constraints into account. We prove strong NP-completeness results for different objective functions, namely makespan minimization and throughput maximization, on simple star-shaped platforms. We propose an approximation algorithm based on the unconstrained version (with unlimited memory) of the problem. We introduce several heuristics, which are evaluated and compared through extensive simulations. An unexpected conclusion drawn from these experiments is that classical scheduling heuristics that try to greedily minimize the completion time of each task are outperformed by the simple heuristic that consists in assigning the task to the available processor that has the smallest communication time, regardless of computation power (hence a "bandwidth-centric" distribution).

Keywords: Scheduling, makespan, steady-state, divisible load, memory constraints, bounded buffers, memory limitation

Résumé

Dans ce rapport, nous nous intéressons au problème de l'allocation d'un grand nombre de taches indépendantes et de taille identiques sur des plateformes de calcul hétérogènes organisées en étoile. Nous nous intéressons également au modèle des tâches divisibles. Pour ces deux modèles, nous prenons en compte les contraintes mémoires et démontrons des résultats de NP-complétude pour diverses métriques (le «makespakan» et le débit). Nous proposons un algorithme d'approximation basé sur la version non-contrainte (c'est-à-dire avec une mémoire infinie) du problème. Nous proposons également d'autres heuristiques que nous évalons à l'aide d'un grand nombre de simulations. Une conclusion inattendue qui ressort de ces expériences est que les heuristiques de listes classiques qui essaient de minimiser gloutonnement la durée de l'ordonnancement sont bien moins performantes que l'heuristique toute simple consistant à envoyer les tâches aux processeurs disponibles ayant le temps de communication le plus faible, sans même tenir compte de leur puissance de calcul.

Mots-clés: Ordonnancement, ressources hétérogènes, régime permanent, tâches divisibles, contraintes mémoire.

1 Introduction

In this paper, we consider the problem of allocating and scheduling a collection of independent, equal-sized tasks on heterogeneous computing platforms. Master-slave tasking on such platforms has received a lot of attention recently [28, 23, 21, 29].

The minimization of the total execution time is a NP-hard problem, even if the platform is a simple tree [17]. In contrast, the optimal steady-state throughput, i.e. the maximal number of tasks that can be processed per time-unit, can be computed in polynomial time, using rational linear programming [1]. Moreover, when the overlay network of computing nodes is tree-shaped, the optimal throughput can be characterized by a set of recursive equations, which are solved via a bottom-up traversal of the tree [5]. From these equations, it is possible to derive a *local* allocation of tasks to resources. This best allocation is *bandwidth-centric*: if enough bandwidth is available at the node, then all children must be kept busy all the time; if bandwidth is limited, then tasks should be allocated only to children which have sufficiently fast communication links, in order of fastest communication time. Counter-intuitively, the maximum throughput in the tree is achieved by delegating tasks to children as quickly as possible, and not by seeking their fastest resolution. The bandwidth-centric strategy is asymptotically optimal, and enjoys the key advantage that the optimal allocation can be computed locally. This enables each component to make autonomous, local scheduling decisions. The approach is thus more scalable than a fully centralized approach.

Nevertheless, the bandwidth-centric periodic solution presented in [5, 1] may well require a huge amount of memory. Indeed, the length of the period may be very large, while the number of buffers required on each resource is proportional to the length of this period. This drawback may prevents the use of the bandwidth-centric in practical situations. In this paper, we take memory constraints into account, and we aim at deriving efficient scheduling strategies for scheduling independent tasks when computing resources have a limited number of buffers.

As expected, the problem becomes more difficult with memory constraints. We prove that finding the minimum number of buffers required to reach the optimal steady-state throughput is a strongly NP-complete problem, and we derive an approximation algorithm based on the unconstrained (without memory limitations) version of this problem. We introduce several heuristics, which are evaluated and compared through extensive simulations.

We also address the counterpart of the previous scheduling problem for divisible tasks: instead of fixed-granularity tasks, we have a *divisible load* application [10] consists of an amount of computation, or *load*, that can be divided into any number of independent pieces. This corresponds to a perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers. For further information on the model, we refer the reader to the recent surveys [11, 27], to the special issue of the Cluster Computing journal [20], and to the Web page collecting related literature [26]. We prove the NP-completeness of the scheduling problem for divisible tasks.

The rest of the paper is organized as follows. Section 2 formally states the different scheduling problems that we consider. In Section 3, we survey related work on scheduling under memory constraints. Then, in Section 4, we derive several complexity results, and we propose an approximation algorithm. In Section 5, we present introduce several heuristics for the problem with independent equal-sized tasks, and we compare them through extensive simulations. Finally, we state some final remarks and conclude the paper in Section 6.

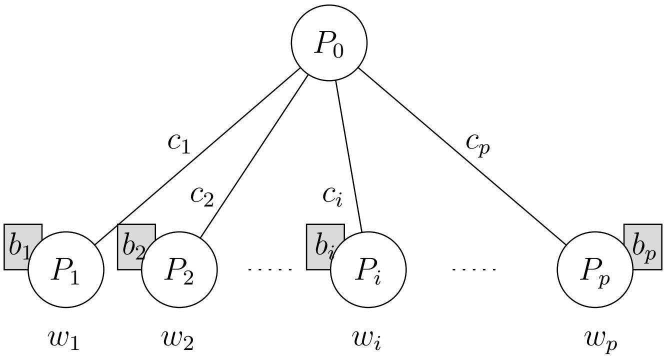

Figure 1: Star Platform

2 Scheduling problems with memory constraints

In this section, we formally state all the scheduling problems under consideration. We apologize that notations are not always consistent: for instance the time-bound is denoted by K in makespan minimization problems, and by T in divisible load problems while T is the period for steady-state problems. We have chosen to stick with the usual notations from the literature, and these vary from one topic to another.

2.1 Makespan Minimization

Consider the star-shaped platform depicted in Figure 1. The master processor P_0 initially holds all the identical tasks $\{T_1, T_2, \ldots, T_N\}$. There are p slave processors, numbered from P_1 to P_p. The time needed to send a task from P_0 to P_i is given by c_i. The time necessary for P_i to process a task is given by w_i. We assume that the communication medium is exclusive: the master can only communicate with a single slave at each time-step. We also assume the possibility of overlapping computations with independent computations. More precisely, a slave P_i can simultaneously execute a task whose data was received in one of its buffers, and receive the data for another task in another buffer, provided that is has enough buffers to do so. Throughout the paper, this star-shaped platform and its operating model will be referred to as the *reference* platform.

Definition 1 (UNBOUNDED-MAKESPAN $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, \{T_1, T_2, \ldots, T_N\}), K)$. *Let $K > O$ be a time-bound, and consider the reference platform. Is it possible to process all the N tasks within K time units on this platform?*

Note that we have to use $\{T_1, T_2, \ldots, T_N\}$ in the description of the problem instance; specifying only N is not possible. Indeed, a solution to the problem is a schedule for the N tasks, and has a size at least proportional to N. Therefore, if we had used N instead of $\{T_1, T_2, \ldots, T_N\}$, the size of the solution would not have been polynomial (i.e. $O(\log N)$ bits) in the size of the problem instance.

A $O(N^2p^2)$ algorithm that solves the UNBOUNDED-MAKESPAN problem is described in [7]. The memory-constrained version can be defined as follows:

Definition 2 (BOUNDED-MAKESPAN $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, (b_i)_{1\leqslant i\leqslant p}, \{T_1, T_2, \ldots, T_N\}, K))$. *Let $K > O$ be a time-bound. Consider the reference platform, and assume that each slave processor P_i is equipped with a bounded buffer that can hold at most b_i tasks. Is it possible to process all the N tasks within K time units on this platform?*

We show in Section 4.1 that this problem is NP-complete.

2.2 Throughput Maximization

When the platform model is more complicated than a star (e.g. a tree or a general graph), the makespan minimization problem turns out to be very difficult [17]. A nice idea to circumvent this difficulty is to relax the objective function by maximizing the steady-state throughput. This problem is polynomial and leads to asymptotically optimal solutions for the makespan minimization problem. On a star platform, finding the optimal steady-state throughput, i.e. the optimal number of tasks that can be processed per time-unit, can be formalized as follows:

Definition 3 (UNBOUNDED-THROUGHPUT $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, \rho)$). *Let $K > O$ be a time-bound, and consider the reference platform. Is there a K-periodic schedule of period T, i.e. a schedule that executes K tasks every T time-units in steady state, and such that $\frac{K}{T} \geqslant \rho$?*

Using a linear-programming formulation, this problem can be solved by a $O(p\log(p))$ algorithm [5, 1]. The bounded version of the throughput problem can be defined as follows:

Definition 4 (BOUNDED-THROUGHPUT $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, (b_i)_{1\leqslant i\leqslant p}, \rho)$). *Let $K > O$ be a time-bound. Consider the reference platform, and assume that each slave processor P_i is equipped with a bounded buffer that can hold at most b_i tasks. Is there a K-periodic schedule of period T, i.e. a schedule that executes K tasks every T time-units in steady state, and such that $\frac{K}{T} \geqslant \rho$?*

The formulation of the UNBOUNDED-THROUGHPUT and BOUNDED-THROUGHPUT problems is questionable, because as stated these problems may not belong to NP. Indeed, the size of K and T could be exponential in the size of the problem instance. For the UNBOUNDED-THROUGHPUT problem, it turns out the polynomial-time algorithm given in [5, 1] does provide a solution where K and T have a polynomial size. For the BOUNDED-THROUGHPUT problem, the size of K has no reason to be polynomial in the size of the original instance. Therefore, we need to define the following "compact" version of the problem:

Definition 5 (COMPACT-BOUNDED-THROUGHPUT $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, (b_i)_{1\leqslant i\leqslant p}, S, \rho)$). *Let $K > O$ be a time-bound. Consider the reference platform, and assume that each slave processor P_i is equipped with a bounded buffer that can hold at most b_i tasks. Is there a K-periodic schedule of period T, i.e. a schedule that executes K tasks every T time-units in steady state, and such that $K \leqslant \log S$ and $\frac{K}{T} \geqslant \rho$?*

COMPACT-BOUNDED-THROUGHPUT belongs to NP (see Lemma 2) but is more constrained than BOUNDED-THROUGHPUT. We show in section 4.2 that both problems are complete. Hence the difficulty is intrinsically due to the memory limitation, and not to the statement of the problem.

2.3 Divisible load scheduling

A divisible task corresponds to a perfect parallel job that can be arbitrarily split into several independent parts. In the simplest variant, computation and communication times for a given chunk are assumed to grow linearly with the chunk size. However, this is not realistic for communications, and recent papers have added a start-up overhead in the model, to take link latency into account. In this paper we also focus on this affine cost model: it takes w_iX time-units to execute X units of load on worker P_i, and $G_i + X.g_i$ time-units to send X units of load from the master processor P_0 to P_i. Note that in the case of independent tasks, the latency is directly incorporated in the value c_i (since the size X of a task is fixed beforehand). Two algorithmic techniques have been proposed to schedule divisible loads, namely one-round and multi-round algorithms:

- In a one-round distribution, each processor is used at most once. Therefore, the first problem is to select a subset of workers and to determine in which order the chunks should be sent to the different workers, given that the master can perform only one communication at a time. Once the communication order has been determined, the second problem is to decide how much work should be allocated to each worker P_i: each P_i receives α_i units of load, where $\sum_{i=1}^{p} \alpha_i = W_{\text{total}}$. The final objective is to minimize the makespan, i.e. the total execution time. Selection and ordering are the most difficult parts of the problem since the α_i's can then be found by solving a simple linear program (closed-form expressions are also available [9, 2]).

- One-round distributions lead to a poor utilization of the workers. To alleviate this problem, *multi-round* algorithms have been proposed. These algorithms dispatch the load in multiple rounds of work allocation, and thus improve the overlapping of communications with computations. By comparison with one-round algorithms, work on multi-round algorithms has been scarce. The two main questions that must be answered are: (i) what should the chunk sizes be at each round? and (ii) how many rounds should be used? It is widely acknowledged that the latencies introduced in the affine model make the model more realistic and cannot be avoided when dealing with multi-round algorithms.

We target the most general case of multi-round distributions. We define the *divisible* platform as the platform of Figure 1, where the master P_0 initially holds W units of load to distribute; the time necessary to send x units of load from P_0 to P_i is given by $G_i + x.g_i$ and the time necessary to process this work by P_i is given by $x.w_i$.

Definition 6 (UNBOUNDED-DIVISIBLE $((g_i)_{1\leqslant i\leqslant p}, (G_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, W, T)$). *Let $T > O$ be a time-bound, and consider the divisible platform. Is it possible to process all the W load units within T time-units on this platform, using a multi-round distribution?*

The complexity of this problem is still an open problem, even for one-round distributions.

Definition 7 (BOUNDED-DIVISIBLE $((g_i)_{1\leqslant i\leqslant p}, (G_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, (b_i)_{1\leqslant i\leqslant p}, W, T)$). *Let $T > O$ be a time-bound. Consider the divisible platform, where each slave P_i cannot hold more than b_i units of load at any moment. Is it possible to process all the W load units within T time-units on this platform, using a multi-round distribution?*

We show the NP-completeness of BOUNDED-DIVISIBLE in Section 4.3.

3 Related Work

To the best of our knowledge, the closest work on throughput maximization under memory constraints is presented in [12, 24]. The authors study the number of buffers required to reach the optimal steady-state throughput. Assuming that communications are non-interruptible, they state the following results:

- For every positive integer k, there exists a tree such that there is at least one node needing more than k buffers to reach optimal steady-state throughput.

- For every tree, there is a positive integer m such that if each node in the tree has m buffers, optimal steady-state can be reached.

- Even with unlimited buffers, a flawed protocol may not reach optimal steady-state because a child may request too many tasks from its parent.

From these remarks, they conclude that with non-interruptible communications, a bandwidth-centric protocol using a fixed number of buffers will not reach optimal steady-state throughput in all trees. Therefore they propose an autonomous buffer growing protocol that automatically adjusts the number of required buffers. To solve the anarchic growth of buffers problem, they study the situation where communications are interruptible. They experimentally show that, when allowing interruptible communications, three buffers are sufficient to reach optimal steady-state throughput in 99,6% of the cases. Allowing interruptible communications allows communications to flow continuously at a fixed rate and therefore amounts to modify the granularity, hence minimizing buffering. Nevertheless, many programming environments do not provide such a precise control on communications and prevent from using these results. Providing dynamic strategies that minimize the number of buffers in the model with non-interruptible communications is essential.

Another theoretical result whose framework is close to ours is given by Drozdowski et al. [16]. The authors consider scheduling divisible loads on a distributed computing system with limited available memory. They use the same model as in this paper: communication costs are modeled by affine functions whereas computation costs are modeled by linear functions. They show that finding the optimal one-round load distribution (i.e. selecting which resources are used and in which order to send the data to the selected processors) is NP-hard under memory constraints (using a reduction to 2-Partition that is weakly NP-hard [19]). Using integer linear programming, they propose a robust (albeit possibly non-polynomial) algorithm to tackle the difficulty of this problem and demonstrate its efficiency using extensive simulations.

Other related papers deal with operating system problems, particularly memory-aware scheduling on time sliced parallel machines. Paging activity damages the synchronism among the processes of a given job, and delaying some jobs leads to better overall performance by preventing the harmful effects of paging and thrashing. Co-scheduling and memory-aware scheduling may be implemented by modifying the kernel [3, 15].

In the next section, we derive some complexity results for the distribution of tasks on a graph in presence of limited memory. More precisely, we prove that finding the optimal distribution on a star graph in NP-Complete in the strong sense when we aim at minimizing the makespan or maximizing the throughput. Of course these results induce the NP-Completeness for more complex graphs such as trees or general graphs, and should be contrasted with the case of unlimited memory. On the one hand, in the case of makespan minimization, polynomial algorithms are known for the case of star [7] or spider graphs [17], but the problem becomes NP-Complete in the strong sense for trees [18]. On the other hand, when we aim at maximizing the throughput, i.e. at finding a periodic schedule that maximizes the number of processed tasks per time unit, there exist polynomial algorithms for trees and general graphs [5, 1]. The complexity results for the distribution of independent tasks on different platforms, with or without memory limitations, are summarized in Table 1.

4 Complexity results

In this section we first prove NP-completeness results for the various problems defined in Section 2. Then we propose an approximation algorithm for the BOUNDED-MAKESPAN problem in Section 4.4. All proofs presented in this section use a reduction to 3-Partition, which is NP-Complete in the strong sense [19].

Definition 8 (3-Partition). *Given $3n$ integers $a_1, \ldots, a_{3n}$, such that $\sum_1^{3n} a_i = nB$ and $\forall i$, $\frac{B}{4} < a_i < \frac{B}{2}$. Is there a partition of the a_i's into n groups of 3 integers, such that each a_i belongs exactly to one group, and each group sums to B.*

Objective function	Memory limitation	Star and Spider Graphs	Trees and General Graphs
Makespan Min	No	Polynomial [7, 17]	NP Complete [18]
Makespan Min	Yes	NP Complete (this paper)	NP Complete (this paper)
Throughput Max	No	Polynomial [5]	Polynomial [1]
Throughput Max	Yes	NP Complete (this paper)	NP Complete (this paper)
Divisible Linear One-round	No	Polynomial [8]	Polynomial for trees [8] Open for general graphs
Divisible Linear One-round	Yes	Open	Open
Divisible Affine One-round	No	Open	Open
Divisible Affine One-round	Yes	Weakly NP Complete [16] NP Complete (this paper)	Weakly NP Complete [16] NP Complete (this paper)
Divisible Linear Multi-round	No	Asymptotically optimal algorithm [8]	Asymptotically optimal algorithm [8]
Divisible Linear Multi-round	Yes	Open	Open
Divisible Affine Multi-round	No	Asymptotically optimal algorithm [8]	Asymptotically optimal algorithm [8]
Divisible Affine Multi-round	Yes	NP Complete (this paper)	NP Complete (this paper)

Table 1: Summary of complexity results.

4.1 BOUNDED-MAKESPAN

Theorem 1. *BOUNDED-MAKESPAN (b_i, c_i, w_i, K, N) is NP-Complete in the strong sense.*

Proof. We have to polynomially transform the instance of 3-Partition into an instance of BOUNDED-MAKESPAN which has a solution if and only if the original instance of 3-Partition has a solution.

Let us consider the following instance of BOUNDED-MAKESPAN, consisting of a master processor P_0, $3n$ slave processors $P_1, \dots, P_{3n}$ with the following characteristics.

- $b_i = 1$, i.e. processor P_i cannot start receiving a new task until it has processed the task it holds
- $c_i = a_i, w_i = 2nB$, i.e. it takes a_i time units to P_i to receive a task from P_0 and $2nB$ time units to process it,

and one processor P_B with $b_B = 1, c_B = B, w_B = B$. Moreover let us set $N = 5n$ and $K = 4nB$.

$\boxed{\Rightarrow}$ Let us first suppose that there is a solution to the original instance of 3-Partition and let us suppose, without loss of generality, that

$$\forall 0 \leqslant j \leqslant n-1, \quad a_{3j+1} + a_{3j+2} + a_{3j+3} = B.$$

Then, the schedule depicted in Figure 2 provides a solution to BOUNDED-MAKESPAN (b_i, c_i, w_i, K, N).

$\boxed{\Leftarrow}$ Let us now suppose that there is a solution to the instance of BOUNDED-MAKESPAN (b_i, c_i, w_i, K, N) we have built.

Lemma 1. *In a solution of BOUNDED-MAKESPAN (b_i, c_i, w_i, K, N), each processor P_i processes exactly 1 task, and P_B processes exactly $2n$ tasks.*

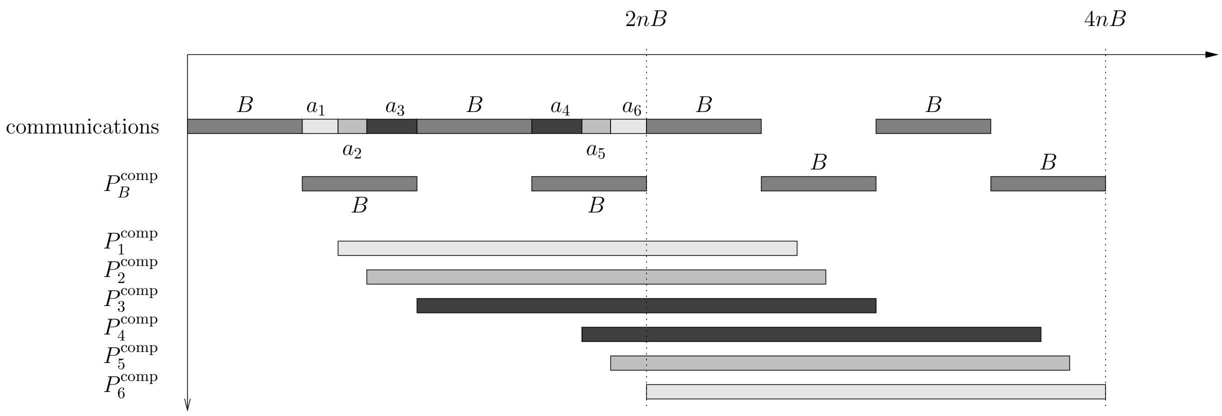

Figure 2: solution to BOUNDED-MAKESPAN

Proof. Let us first consider the case of P_i. It cannot receive its first task before time step a_i, and thus, it cannot finish its processing before time step $a_i + 2nB$. Therefore, it cannot receive its second task before time step $2a_i + 2nB$, and therefore, it cannot process a second task within the time bound $K = 4nB$. Thus, each P_i processes at most one task.
Clearly, for the same reasons, P_B cannot process more than $2n$ tasks within $4nB$ time units. Indeed, since it only holds one buffer, communications and processing cannot overlap.
Therefore, each P_i cannot process more than one task, and P_B cannot process more than $2n$ tasks, such that, since $N = 5n$, in any optimal solution, each P_i processes exactly one task, and P_B processes exactly $2n$ tasks. ∎

Using Lemma 1, we can prove that communications to P_B are necessarily organized as depicted in Figure 2. Moreover, since it takes $2nB$ time units to P_i to process one task, all the communications to the P_i's must be finish before time step $2nB$. Therefore, those communications must take place in the n disjoint holes of size B left free by the communications to P_B, thus providing a solution to the original instance of 3-Partition. ∎

4.2 BOUNDED-THROUGHPUT

Theorem 2. *COMPACT-BOUNDED-THROUGHPUT (b_i, c_i, w_i, ρ, S) is NP-Complete in the strong sense.*

Proof. We first prove that COMPACT-BOUNDED-THROUGHPUT $\in NP$ and then prove its completeness.

Lemma 2. *COMPACT-BOUNDED-THROUGHPUT $\in NP$.*

Proof. Let us prove that, given the schedule, i.e. for each task T_i processed during the period, the couple $(d(i), P(i))$ where $d(i)$ is the date when the i-th task is sent and $P(i)$ the processor to whom it is sent, we can check in polynomial time that this schedule is feasible. We can check easily that the outgoing port of P_0 is not saturated, i.e. that

$$\forall 0 \leqslant i < K-1, \quad d(i+1) \geqslant d(i) + c_{P(i)} \text{ and } d(0) + T \geqslant d(K-1) + c_{P(K-1)}.$$

We also need to check that memory constraints (due to limited buffer) are not exceeded. Let us consider the case of processor P_i, and let us denote by t_j^i the date when the j-th task is sent to P_i during the period. Since P_i can hold at most b_i tasks, we need to check that for every group of b_i successive tasks, at least one task has been processed between the end of the first communication of the group and the beginning of the last communication, i.e.

$$\forall 0 \leqslant k \leqslant K-1-b_i, \quad t^i_k + c_i + w_i \leqslant t^i_{k+b_i} \quad \text{and}$$

$$\forall K-1-b_i < k \leqslant K-1, \quad t^i_k + c_i + w_i \leqslant T + t^i_{k+b_i \bmod K}$$

Thus, since all above conditions can be checked in polynomial time, COMPACT-BOUNDED-THROUGHPUT $\in NP$. ■

Again, we use a reduction to 3-Partition, which is NP-Complete in the strong sense [19]. We have to polynomially transform the instance of 3-Partition into an instance of COMPACT-BOUNDED-THROUGHPUT which has a solution if and only if the original instance of 3-Partition has a solution.

Let us consider the following instance of COMPACT-BOUNDED-THROUGHPUT, consisting of a master processor P_0, $3n$ slave processors $P_1, \ldots, P_{3n}$ with the following characteristics:

- $b_i = 1$, i.e. processor P_i cannot start receiving a new task until it has processed the task it holds
- $c_i = a_i, w_i = 2nB - a_i$, i.e. it takes a_i time units to P_i to receive a task from P_0 and $2nB - a_i$ time units to process it,

and one processor P_B with $b_B = 1, c_B = B, w_B = B$. Moreover let us set $\rho = \frac{2}{B}$ and $\log S = 4n$.

$\boxed{\Rightarrow}$ Let us first suppose that there is a solution to the original instance of 3-Partition and let us suppose, without loss of generality, that

$$\forall 0 \leqslant j \leqslant n-1, \quad a_{3j+1} + a_{3j+2} + a_{3j+3} = B.$$

Then, the schedule depicted in Figure 3 provides a solution to COMPACT-BOUNDED-THROUGHPUT (b_i, c_i, w_i, ρ, S).

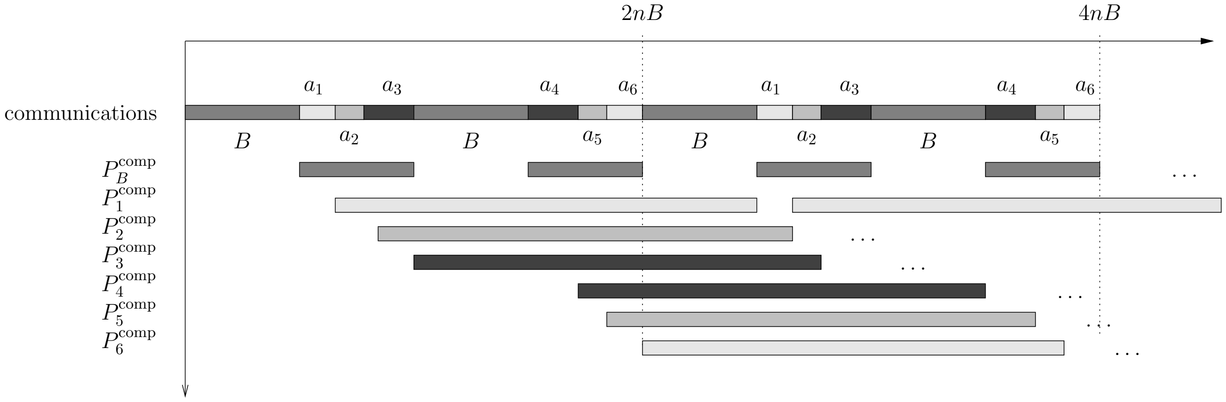

Figure 3: solution to COMPACT-BOUNDED-THROUGHPUT

$\boxed{\Leftarrow}$ Let us now suppose that there is a solution to the instance of COMPACT-BOUNDED-THROUGHPUT (b_i, c_i, w_i, ρ, S) we have built.

Lemma 3. *In a solution of* COMPACT-BOUNDED-THROUGHPUT (b_i, c_i, w_i, ρ, S)*, then* $K = 4n$*,* $T = 2nB$*, each processor* P_i *processes exactly 1 task during each period, and* P_B *processes exactly* n *tasks during one period.*

Proof. Since $K \leqslant \log S = 4n$ then $K \leqslant 4n$ and since $\frac{K}{T} \geqslant \rho = \frac{2}{B}$, then $T \leqslant 2nB$. Let us suppose that $T < 2nB$. Then, since it takes $2nB$ time units to P_i to process one task, then the number of tasks allocated to P_i during one period is necessarily 0, and P_B is unable to process more than $\frac{T}{2B}$ during T time units and $K \leqslant \frac{T}{2B}$. Thus,

$$\frac{K}{T} \leqslant \frac{1}{2B},$$

and the throughput cannot be achieved. Therefore, in any solution, $T = 2nB$ and $K = 4n$. Moreover, during $2nB$ time units, P_i cannot process more than 1 task, and P_B cannot process more than n tasks, so that in any solution, P_i processes exactly 1 task during each period, and P_B processes exactly n tasks during each period of duration $2nB$. ∎

We will not detail the end of the proof of the completeness of COMPACT-BOUNDED-THROUGHPUT (b_i, c_i, w_i, ρ, S) since it very close to the proof of the completeness of BOUNDED-MAKESPAN. Indeed, since n tasks have to be sent to P_B during $2nB$ time units and that communications and processing cannot overlap in P_B, then the communications to the P_i's must take place in the n disjoint holes left free by processing on P_B, thus inducing a solution to 3-Partition. ∎

Theorem 3. *BOUNDED-THROUGHPUT* (b_i, c_i, w_i, ρ) *is complete in the strong sense.*

Proof. The demonstration is very similar to the COMPACT-BOUNDED-THROUGHPUT NP-completeness demonstration. We use exactly the same instance consisting of a master processor P_0, $3n$ slave processors $P_1, \ldots, P_{3n}$ with the following characteristics:

- $b_i = 1$. Thus, processor P_i cannot start receiving a new task since it has processed the task it holds
- $c_i = a_i, w_i = 2nB - a_i$. It takes a_i time units to P_i to receive a task from P_0 and $2nB - a_i$ time units to process it,

and one processor P_B with $b_B = 1, c_B = B, w_B = B$. Moreover we set $\rho = \frac{2}{B}$. It is easy to see that if there is a solution to the original instance of 3-Partition, the schedule depicted in Figure 3 provides a solution to BOUNDED-THROUGHPUT (b_i, c_i, w_i, ρ).

Let us prove the converse by supposing that we have a solution to BOUNDED-THROUGHPUT (b_i, c_i, w_i, ρ) and consider the corresponding pattern (and the associated emission order). Since none of the processors can overlap their communications and their computations (due to the buffer limitation), the throughput of P_i is bounded by $1/2nB$ and the throughput of P_B is bounded by $1/2B$. Moreover, the throughput of the whole platform is bounded by the sum of the throughput of all the processors, i.e. $2/B$. Therefore, to reach the desired throughput, each processor works at full speed. Note that to reach this throughput, the communication medium has to be used without any interruption.

We can freely consider that P_b is the first processor to receive its task. Since P_b has to work at full speed, the master has to send a task to P_b a time $k.2B$ for all k. Therefore, the master has to send its tasks to the other processors in the time intervals $[(2k+1)B, (2k+2)B]$ of length B. Since the communication medium is used without any interruption, exactly three P_i's (say P_{i_1}, P_{i_2}

and P_{i_3}) are sent data during the time interval $[B, 2B]$. All indices i_1, i_2 and i_3 are distinct and since the P_i's should work at full speed to reach the desired throughput, the same communication order (i.e. P_{i_1}, P_{i_2} and P_{i_3}) will occur in time intervals $[B + k.2nB, 2B + k.2nB]$. Using the same argument, exactly three P_i's (say P_{i_4}, P_{i_5} and P_{i_6}) are sent data during the time interval $[3B + k.2nB, 4B + k.2nB]$ and $\{i_1, \ldots, i_6\}$ are all distinct due to the computation times of the P_i's. By recurrence, we can thus prove that $i_1, \ldots, i_{3n}$ define a solution to the original instance of 3-Partition. ■

It is possible to prove a stronger result when considering non-periodic schedules (see [6] or the survey paper of Hanen and Munier [22] for an introduction to cyclic scheduling). A cyclic schedule is an infinite (not necessarily periodic) schedule and its throughput is defined, provided that this limit exists, as

$$\lim_{N\to\infty} \frac{N}{T_N}, \text{ where } T_N \text{ is the completion time of the } N^{th} \text{ task.}$$

Definition 9 (APERIODIC-BOUNDED-THROUGHPUT $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, (b_i)_{1\leqslant i\leqslant p}, \rho)$). *Consider the reference platform, and assume that each slave processor P_i is equipped with a bounded buffer that can hold at most b_i tasks. Is there a cyclic schedule whose throughput is larger than ρ?*

Theorem 4. *APERIODIC-BOUNDED-THROUGHPUT (b_i, c_i, w_i, ρ) is complete in the strong sense.*

Proof. Again, we use exactly the same instance consisting of a master processor P_0, $3n$ slave processors $P_1, \ldots, P_{3n}$ with the following characteristics:

- $b_i = 1$. Thus, processor P_i cannot start receiving a new task since it has processed the task it holds
- $c_i = a_i, w_i = 2nB - a_i$. It takes a_i time units to P_i to receive a task from P_0 and $2nB - a_i$ time units to process it,

and one processor P_B with $b_B = 1, c_B = B, w_B = B$. Moreover we set $\rho = \frac{2}{B}$. It is easy to see that if there is a solution to the original instance of 3-Partition, the schedule depicted in Figure 3 provides a solution to APERIODIC-BOUNDED-THROUGHPUT (b_i, c_i, w_i, ρ).

Let us prove the converse by supposing that there exists a cyclic schedule whose throughput is larger than $2/B$. Since we are only interested with the asymptotic behavior, we can freely assume that the first task is sent to P_B. We first introduce a few notations. Let us denote by $\tau_B(l)$ the date of the emission of the l^{th} task to P_B. In the following, we consider the time intervals $I_k = [\tau_B((k-1)n+1), \tau_B(kn+1)]$. Δ_k is the length of these intervals and n_k is the number of task sent to the P_i's during the time interval I_k. Since n tasks are sent to P_B in time interval I_k, for all k we have $\Delta_k \geqslant 2nB$.

Lemma 4. *For all $\varepsilon > 0$, there exists a k such that $\Delta_k < 2nB + \varepsilon$ and $n_k \geqslant 3n$.*

Proof. We aim at proving that, to reach the desired throughput, the Δ_k's should be close to $2nB$ and the n_k should be mostly equal to $3n$. Therefore, we also introduce δ_k that denotes the "deviation" of Δ_k (i.e. $\Delta_k - 2nB$).

Let us suppose that there exists an $\varepsilon > 0$ such that for all k we have

$$\begin{cases} \Delta_k \geqslant 2nB + \varepsilon \text{ or} \\ n_k < 3n \end{cases}$$

1. If $\Delta_k \geqslant 2nB+\varepsilon$, since the P_i's cannot overlap their communications and their computations, we have $n_k^{(i)} \times 2nB \leqslant \Delta_k$, where $n_k^{(i)}$ denotes the number of tasks sent to P_i in the time interval I_k. Therefore
$$n_k \leqslant \frac{3\Delta_k}{2B}.$$
We also have
$$n = \frac{\Delta_k - (\Delta_k - 2nB)}{2B} = \frac{\Delta_k - \delta_k}{2B}$$
and then
$$n + n_k \leqslant \frac{2\Delta_k}{B} - \frac{\delta_k}{2B}, \text{ where } \delta_k \geqslant \varepsilon.$$
2. If $\Delta_k < 2nB+\varepsilon$, then we have $n_k < 3n$ and thus $n+n_k \leqslant 4n-1$. Last, since $\Delta_k \geqslant 2nB$, we have
$$n + n_k \leqslant \frac{2\Delta_k}{B} - 1.$$

Let us denote by $\mathcal{K}_1$ (resp. K_1) the set (resp. the number) of I_k's such that $\Delta_k \geqslant 2nB+\varepsilon$ and $\mathcal{K}_2$ (resp. K_2) the set (resp. the number) of I_k's such that $\Delta_k < 2nB+\varepsilon$.

First note that on one hand we have
$$N = \sum_{\mathcal{K}_1\cup\mathcal{K}_2} n + n_k \leqslant \frac{2}{B} \underbrace{\sum_{\mathcal{K}_1\cup\mathcal{K}_2} \Delta_k}_{T_N} - \sum_{\mathcal{K}_1} \frac{\delta_k}{2B} - \sum_{\mathcal{K}_2} 1$$
and thus
$$\frac{N}{T_N} \leqslant \frac{2}{B} - \frac{S}{2BT_N} - \frac{K_2}{T_N}, \text{ where } S = \sum_{\mathcal{K}_1} \delta_k.$$

On the other hand we have
$$T_N = \sum_{\mathcal{K}_1\cup\mathcal{K}_2} \Delta_k \leqslant K_1 2nB + S + K_2(2nB+\varepsilon),$$
and therefore,
$$\frac{N}{T_N} \leqslant \frac{2}{B} - \frac{1}{2B} \underbrace{\frac{2BK_2 + S}{K_1 2nB + K_2(2nB+\varepsilon) + S}}_{loss(K_1,K_2,S)}, \text{ with } K_1, K_2 \in \mathbb{N} \text{ and } S \geqslant K_1\varepsilon.$$

$loss(K_1,K_2,S)$ is a bilinear transformation in all its variables and is therefore monotonic in each variable. Therefore we have:
$$\begin{aligned} loss(K_1,K_2,S) &\geqslant \min\left(\frac{2B}{2nB+\varepsilon}, \frac{S}{K_1 2nB + S}\right), \text{ using bilinearity in } K_2 \\ &\geqslant \min\left(\frac{2B}{2nB+\varepsilon}, 1, \frac{K_1\varepsilon}{K_1 2nB + K_1\varepsilon}\right), \text{ using bilinearity in } S \\ &\geqslant \min\left(\frac{2B}{2nB+\varepsilon}, 1, \frac{\varepsilon}{2nB+\varepsilon}\right) \end{aligned}$$

Therefore, this schedule cannot reach the desired throughput, which is absurd. ∎

Using previous lemma, we know that there exists a k such that $2nB \leqslant \Delta_k \leqslant 2nB + 1/2$ and $n_k \geqslant 3n$. In such a short interval, there cannot be more than one communication to a given P_i. Therefore there is exactly one communication to each P_i in I_k and n communications to P_B. Let us now consider the intervals $J_j = [\tau_B(n(k-1)+j-1)+B, \tau_B(n(k-1)+j)]$ for $j \in [\![1,n]\!]$. The length of these intervals is always comprised between B and $B+1/2$. All the remaining communications have to fit in the empty slots and, since the communication times are integers, all communications in a row should be made in a time smaller than B. Therefore, the communication order defines a solution to the original instance of 3-Partition. ∎

4.3 BOUNDED-DIVISIBLE

Theorem 5. *BOUNDED-DIVISIBLE* $(G_i, g_i, w_i, w_i, W, T)$ *is NP-Complete in the strong sense.*

Proof. We have to polynomially transform the instance of 3-Partition into an instance of BOUNDED-DIVISIBLE which has a solution if and only if the original instance of 3-Partition has a solution.

Let us consider the following instance of BOUNDED-MAKESPAN, consisting of a master processor P_0, $n+1$ slave processors $P_1, \ldots, P_{n+1}$ with the following characteristics (N and ε are respectively a large and a small value that will be explicitly given later).

- $G_i = NB$ and $g_i = 0$. Thus sending α units of load to P_i requires a time NB.
- $w_i = 1$ and $b_i = (n+1-i)(N+1)B + n^2B$.

and $3n$ slave processors $Q_1, \ldots, Q_{3n}$ with the following characteristics

- $G_i = a_i$ and $g_i = 0$. Thus sending α units of load to Q_i requires a time a_i.
- $w_i = \varepsilon.a_i$ and $b_i = \infty$.

The time bound is equal to

$$T = n(N+1)B + NB + n^2B$$

and the amount of load to distribute is equal to

$$\begin{aligned}
W_{\text{tot}} &= \underbrace{\left(\frac{n(n+1)}{2}(N+1)B\right) + \left(n^2(n+1)B\right)}_{W_{\text{tot}}^P,\text{ contribution of processors } P_i} + \underbrace{\varepsilon\left(\frac{n(n+1)}{2}(N+1)B + n(n^2-1)B\right)}_{W_{\text{tot}}^Q,\text{ contribution of processors } Q_i} \\
&= \underbrace{\left(\frac{n(n+1)}{2}B\right)}_{W_1} N + \underbrace{\left(\frac{n(n+1)}{2}B + n^2(n+1)B\right)}_{W_2} + \underbrace{\left(\frac{n(n+1)}{2}B^2\right)}_{W_3} N\varepsilon \\
&\quad + \underbrace{\left(\frac{n(n+1)}{2}B^2 + n(n^2-1)B\right)}_{W_4}\varepsilon
\end{aligned}$$

The idea of the following proof is to use the decomposition of this workload into four parts: $W_{\text{tot}} = W_1N + W_2 + W_3N\varepsilon + W_4\varepsilon$ where the values of N and ε are such that W_2 is negligible in front of W_1N, and $W_4\varepsilon$ is negligible in front of W_2 or $W_3N\varepsilon$. Thus, we have to choose N big enough and ε small enough such that if a small part of the contribution of, for example, W_1N is missing, this loss cannot be circumvented by a rise of W_2 in a feasible schedule. The following values of N and

ε can be used in the proof, but it is often easier to remember that N is as big and ε as small as necessary.

$$\begin{aligned} N &= (n+1)^3 \\ \varepsilon &= \frac{1}{2B^2(n+2)^4(n+3)^2} \end{aligned}$$

$\boxed{\Rightarrow}$ Let us first suppose that there is a solution to the original instance of 3-Partition and let us suppose, without loss of generality, that

$$\forall 0 \leqslant j \leqslant n-1, \quad a_{3j+1} + a_{3j+2} + a_{3j+3} = B.$$

Then, the schedule depicted in Figure 4 provides a solution to BOUNDED-DIVISIBLE $((P_i, Q_i), W, T)$.

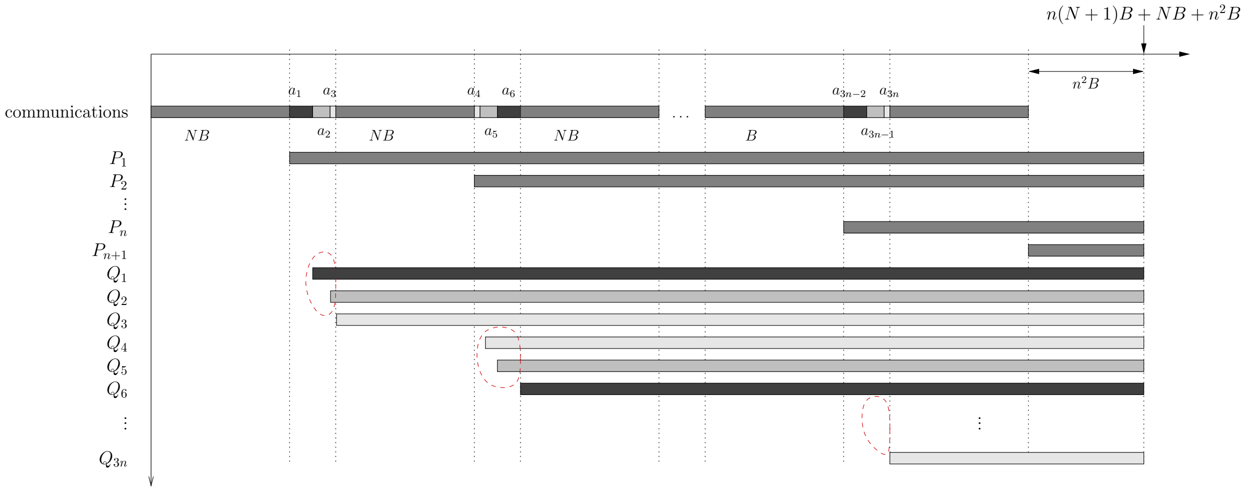

Figure 4: solution to BOUNDED-DIVISIBLE

Indeed, processor P_i processes exactly b_i units of load. Therefore the overall load processed by the P_i's is given by

$$\begin{aligned} W_P = \sum_{i=1}^{n+1} b_i = \sum_{i=1}^{n+1} (n+1-i)(N+1)B + n^2B = n^2(n+1)B + \sum_{i=1}^{n} i(N+1)B \\ = W_1 N + W_2. \end{aligned}$$

Even when forgetting the small fraction of load processed by the Q_i's in the dashed areas (remember $NB >> B$), the load processed by the Q_i's is given by

$$\begin{aligned} W_Q &= \sum_{i=1}^{n} (a_{3i-2} + a_{3i-1} + a_{3i})\varepsilon\big((n+1-i)(N+1)B - B + n^2B\big) \\ &= B\varepsilon\left((N+1)B\sum_{i=1}^{n} i + n((n^2-1)B)\right) \\ &= B\varepsilon\left(\frac{n(n+1)}{2}(N+1)B + n(n^2-1)B\right) \end{aligned}$$

Therefore, when summing both previous expressions (W_P and W_Q), we can check that the overall load processed by the P_i's and Q_i's is larger than W_{tot}, thus providing a solution to BOUNDED-DIVISIBLE.

$\boxed{\Leftarrow}$ Let us now suppose that it is possible to distribute the W_{tot} units of load within T units of time.

Lemma 5. *Such a solution is necessarily a one-round distribution (i.e. where the master communicates at most once with each processor) of the W_{tot} units of load. Moreover, in this distribution, all the P_i's are used.*

Proof. Let us suppose that one of the P_i is not used in this solution. We may assume without loss of generality that this processor is the one with the smallest buffer, i.e. P_{n+1}. The amount of load processed within our time-bound is then smaller than

$$\begin{aligned}
W_{\text{sup}} &= \underbrace{\sum_{i=1}^{n}(T - iNB)}_{\text{when forgetting } P_i\text{'s buffers}} + \underbrace{\sum_{i=1}^{3n}\varepsilon a_i T}_{\text{when forgetting } Q_i\text{'s latency}} \\
&= nT - \frac{n(n+1)}{2}NB + Bn\varepsilon T \\
&= n^2(N+1)B + nNB + n^3B - \frac{n(n+1)}{2}NB + Bn\varepsilon T \\
&= N\left(\frac{n(n+1)}{2}B\right) + \left(n^2B + n^3B\right) + Bn\varepsilon T \\
&= W_1N + W_2 - \frac{n(n+1)}{2}B + Bn\varepsilon T
\end{aligned}$$

Since $\frac{n(n+1)}{2}B > Bn\varepsilon T + \varepsilon(W_3N + W_4)$ (thanks to the choice of ε), we have $W_{\text{sup}} < W_{\text{tot}}$. Therefore all the P_i's have to be used.

Let us now prove that each P_i is used exactly once. In the following, n_i denotes the number of times when the master sends data to P_i (so we want to prove that $n_i = 1$ for each P_i).

$$\begin{aligned}
\sum_{i=1}^{n+1} n_i(NB) \leqslant T = (n+1)NB + nB + n^2B, &\text{ therefore} \\
\sum_{i=1}^{n+1} n_i \leqslant (n+1) + \frac{n+1}{N}n &\text{ and since } N < n(n+1), \\
0 \leqslant \sum_{i=1}^{n+1} n_i \leqslant (n+1) &\text{ (because all these quantities are integers).}
\end{aligned}$$

Therefore, at most $n+1$ communications with the P_i's can be initiated.

Let us now suppose that the master communicates more than once with one of the Q_i's. Since the bandwidth between the master and Q_i is infinite and since the Q_i's have no buffer limitations, we can freely suppress all communications to Q_i but the first one, while processing exactly the same amount of work within the same time bound. ■

Lemma 6. *For all $i \in [\![1, n+1]\!]$, P_i starts computing before time step $iNB + (i-1)B$.*

Proof. In a one-round distribution, we can freely assume that communications are started as soon as possible. If P_i starts its processing later than $iNB+(i-1)B$ then he will not be able to process b_i tasks. Since we start communications as soon as possible and w_i's and c_i's are integers, communications and computations starting dates are all integers. Therefore, such a P_i would start at least one unit of time "too late" and the amount of load processed would be smaller than

$$\begin{aligned} W_{\text{sup}} &= \underbrace{\left(\sum_{i=1}^{n+1} b_i\right) - 1}_{\text{because one } P_i \text{ starts too late}} \qquad + \qquad \underbrace{\sum_{i=1}^{3n} a_i \varepsilon T}_{\text{when forgetting } Q_i\text{'s latency}} \\ &= n^2(n+1)B + \frac{n(n+1)}{2}(N+1)B - 1 + Bn\varepsilon T \qquad = W_1 N + W_2 - 1 + Bn\varepsilon T \end{aligned}$$

Due to our choice of ε, we have $-1 + Bn\varepsilon T < \varepsilon(W_3 N + W_4)$ and therefore it would be impossible to process all W_{tot} tasks. ∎

Previous lemma proves that the master has to start sending data to processor P_i before time $l_i = (i-1)(N+1)B$. However during the time interval $[0, l_i]$ the master can perform at most $(i-1)$ communications targeting processors P_j's. Therefore, we can prove using a straightforward induction that these communications have to be done in the following order: $P_1, P_2, \dots P_{n+1}$.

Let us denote by s_i the time between the end of the communication to P_i and the beginning of the communication to P_{i+1}. Since the all communications to $P_1, \dots, P_i$ take place before l_i, we have:

$$\forall i, \quad \sum_{j=1}^{i-1} s_j + \sum_{j=1}^{i-1} NB \quad \leqslant \quad l_i = (i-1)(N+1)NB$$

and therefore

$$\begin{cases} s_1 \leqslant B \\ s_1 + s_2 \leqslant 2B \\ \quad \vdots \\ s_1 + s_2 + \cdots + s_n \leqslant nB \end{cases}$$

In each of these *holes*, it is possible to communicate with some Q_j's and therefore, s_i is the sum of some a_j's. The rest of the proof consists in proving that that if the communications to the Q_j's are not organized so as to define a solution to 3-Partition, the distribution will fail to distribute the W_{tot} units of load.

Since s_i is the sum of some a_j's, the the contribution of corresponding Q_j's to the processing of tasks is given by W_i, where

$$W_i = s_i \varepsilon \Big(\underbrace{(n+1-i)NB - B + n^2 B}_{\text{non dashed areas}} + \underbrace{W_i^d}_{\text{the dashed area}} \Big)$$

Since the number of tasks processed the Q_j's in dashed area is not larger than $2B$, then

$$W_i \leqslant s_i \varepsilon B \big((n+1-i)N + n^2 + 1 \big)$$

When summing all previous expressions for all $i \in [\![1, n]\!]$, we get that the overall amount of load processed by the Q_j's satisfies

$$\begin{aligned} W_Q &\leqslant NB\varepsilon\left(\sum_{i=1}^{n} s_i(n+1-i)\right) + (n^2+1)B\varepsilon\left(\sum_{i=1}^{n} s_i\right) \\ &\leqslant NB\varepsilon\left(\sum_{i=1}^{n}\sum_{j=1}^{i} s_j\right) + (n^2+1)B\varepsilon\left(\sum_{i=1}^{n} s_i\right) \end{aligned}$$

We have proved that $\forall i,\ \sum_{j=1}^{i} s_j \leqslant iB$, and $\left(\sum_{i=1}^{n}\sum_{j=1}^{i} s_j\right)$ can be developed as follows

$$\begin{aligned} W_Q \leqslant\ & NB\varepsilon\left(\underbrace{s_1}_{\leqslant B} + \underbrace{s_1+s_2}_{\leqslant 2B} + \underbrace{(s_1+s_2+s_3)}_{\leqslant 3B} + \cdots + \underbrace{(s_1+s_2+\cdots+s_n)}_{\leqslant nB}\right) \\ & + (n^2+1)B\varepsilon \cdot nB \end{aligned}$$

If at least one of the inequalities $\sum_{j=1}^{i} s_j \leqslant iB$ is strict, then $\exists i,\ \sum_{j=1}^{i} s_j \leqslant iB - 1$ and

$$\begin{aligned} W_Q &\leqslant NB\varepsilon\left(\frac{n(n+1)B}{2} - 1\right) + (n^2+1)nB^2\varepsilon \\ &\leqslant (W_3N\varepsilon + W_4\varepsilon) - NB\varepsilon. \end{aligned}$$

Thus, this would lead to $W < W_{\text{tot}}$. Therefore, in order to process W_{tot} then necessarily $\forall i,\ s_i = B$. Finally, the communications to the Q_i's must take in the n disjoint holes of size $s_i = B$, thus providing a solution to 3-Partition. ■

Note that in above reduction, we proved that any solution to our instance is necessarily a one-round distribution. Therefore, the result holds true for both one-round and multi-round distributions.

4.4 Approximation Algorithms

In this section, we give an approximation algorithm in presence of limited memory for star graphs when we aim at minimizing the makespan. The approximation algorithm we propose is designed for processors that are able to hold only one task (and thus where tasks have to be distributed one by one and where communications and processing cannot be overlapped), which is the most restrictive case in presence of limited buffer. Therefore, the approximation ratio holds true a fortiori for larger buffers.

The approximation algorithm we propose is a list based scheduling algorithm, whose makespan is not larger than twice the optimal makespan on the platform where all memory constraints have been removed.

The sketch of the list algorithm is depicted in Figure 5. At any time, IdleProc[i] is the next smallest date when processor P_i becomes idle (and thus the smallest date when a task can be sent to P_i since the algorithm makes use of only one buffer), NbTasksSent is the overall number of tasks already sent by P_0, and NbTasksProc[i] is the overall number of tasks already sent to P_i, and NbCommEvent denotes the date of the next communication event. The algorithm we propose requires some pre-processing. We need to know the number n_i of tasks that are sent to P_i in the optimal solution without memory limitation. The n_i's are given by the solution of UNBOUNDED-MAKESPAN, which is described in [7]. In the algorithm described in Figure 5, a task is sent to P_i as soon as

- the communication medium is free
- P_i is idle
- P_i has not processed yet the number of tasks allocated to it in the optimal solution without limited memory.

```
STAR-BOUNDED-BUFFER(c_i, w_i, N)
(n_1, ..., n_p) =STAR-UNBOUNDED-BUFFER(c_i, w_i, N);
NbTasksSent=0;
NextCommEvent=0;
∀i,    IdleProc[i]=0; NbTasksProc[i]=0;
while  NbTasksSent ⩽ N do
   Find P_i, such that IdleProc[i] is minimal and NbTasksProc[i] ⩽ n_i
   if  IdleProc[i] ⩽ NextCommEvent then
      NbTasksProc[i] ++; NbTasksSent++;
      IdleProc[i]=NextCommEvent+c_i + w_i;
      NextCommEvent=NextCommEvent+c_i;
   else
      NexCommEvent=IdleProc[i];
```

Figure 5: List scheduling approximation algorithm

Surprisingly, this very simple heuristic builds a schedule whose makespan cannot be larger than twice the makespan of the optimal schedule without memory limitations:

Theorem 6. *Let us denote by T_{alg} the makespan of the schedule built with limited buffers by* STAR-BOUNDED-BUFFER*(c_i, w_i, N) (defined in Figure 5), and by T_{opt} the makespan of the (optimal) schedule built with unlimited buffers by* STAR-UNBOUNDED-BUFFER*(c_i, w_i, N) (defined in [7]), then*

$$T_{alg} \leqslant 2T_{opt}.$$

Proof. The proof of this theorem is adapted from Graham's proof for list based scheduling [14]. Let us consider the schedule built by STAR-BOUNDED-BUFFER(c_i, w_i, N) and let us denote by $[t_1, t_1 + \alpha_1], \ldots, [t_k, t_k + \alpha_k], [t_l, T_{\text{alg}}]$ the intervals when the communication medium is idle. Clearly $t_l = \sum_1^k \alpha_i + \sum_1^p c_i n_i$ since at each time step, either the communication medium is idle or a task is being sent, and the overall number of tasks sent to P_i is n_i by construction. Let us also denote by P_{last} the processor that finishes its processing at time T_{alg} in the schedule built by STAR-BOUNDED-BUFFER(c_i, w_i, N). The situation is depicted in Figure 6.

Let us consider the case of P_{last}. An idle time in the communication medium is generated by STAR-BOUNDED-BUFFER if and only if all the processors that have not processed all their tasks yet are working. Thus, since P_{last} processes the last task, it has been working at least during all the time intervals $[t_1, t_1 + \alpha_1], \ldots, [t_k, t_k + \alpha_k], [t_l, T_{\text{alg}}]$, of overall size $T_{\text{alg}} - \sum_{i=1}^p c_i n_i$. Therefore, the overall processing time of P_{last} is given by $w_{\text{last}} n_{\text{last}}$, so that

$$T_{\text{alg}} - \sum_{i=1}^{p} c_i n_i \leqslant w_{\text{last}} n_{\text{last}} \text{ and thus } T_{\text{alg}} \leqslant \sum_{i=1}^{p} c_i n_i + w_{\text{last}} n_{\text{last}}.$$

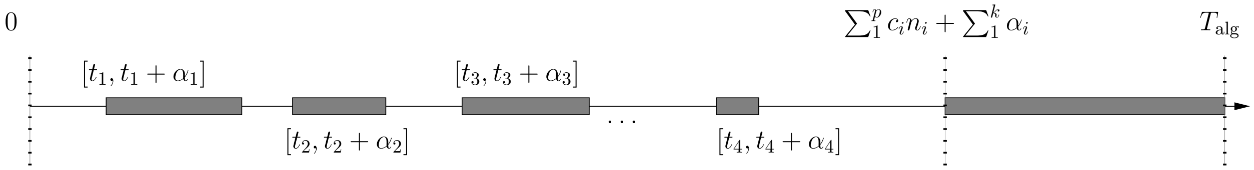

Figure 6: Schedule built by Star-Bounded-Buffer

Moreover, $T_{\text{opt}} \leqslant \sum_{i=1}^{p} c_i n_i$ since $\sum_{i=1}^{p} c_i n_i$ represents the time necessary to send all the tasks to the different slaves in the optimal solution (remember that the numbers of tasks sent to P_i by Star-Bounded-Buffer and Star-Unbounded-Buffer are the same), and $T_{\text{opt}} \leqslant w_{\text{last}} n_{\text{last}}$ since $w_{\text{last}} n_{\text{last}}$ represents the overall processing time on slave P_{last} (again, either with Star-Bounded-Buffer or Star-Unbounded-Buffer). Therefore,

$$T_{\text{alg}} \leqslant 2T_{\text{opt}}.$$

■

5 Simulation

5.1 Heuristics

In this section, we present several heuristics for the model with independent, equal-sized tasks. We recall that c_i is the cost of a communication to processor P_i, w_i is the time needed by P_i to process one task and b_i is the size of its buffer. We introduce some notations:

- IdleProc$[i]$ is the date when P_i has processed all the tasks that have already been assigned to it,
- Available$[i]$ is the date when P_i can receive some tasks (i.e. its buffer is no longer full),
- IdleComm is the date when the communication medium become free,
- NbTasksProc$[i]$ is the number of tasks sent to processor P_i,
- NbTasksSent is the total number of tasks sent to the slaves.

Note that it is possible to derive Available$[i]$ from the current date t and from IdleProc$[i]$, if we assume that no extra task will be sent to P_i after t:

$$\mathsf{Available}[i] = \max\left\{t, \mathsf{IdleProc}[i] - b_i \cdot w_i\right\}$$

Indeed, either the buffer is not full at time t and P_i is available, or it is full and we have to wait. In the latter case, the simplest way to express the availability of P_i is to use IdleProc$[i]$, the time when it becomes idle, and to subtract the time needed to process the b_i tasks.

All the heuristics are list-based heuristics, and they follow the scheme depicted in Figure 7. Only the selection function Select differs from one heuristic to the other.

The selection function Select plays a key role in this scheme: it selects the next target processor (the one that will execute the next task) among all the different processors that are available at a given time step, as soon as the communication medium becomes free. In the following, we present different selection functions.

```
LIST-BASED-HEURISTIC(c_i, w_i, b_i, N)
NbTasksSent = 0;
CurrentDate = 0;
for all P_i do
    IdleProc[i] = 0;
    NbTasksProc[i] = 0;
while NbTasksSent ⩽ N do
    CurrentDate = max {min_i {Available[i]}, IdleComm}
    {Choose a target P_i and a date to start the communication using the selection function}
    (P_i, start_comm_date) = SELECT(CurrentDate, IdleProc, Available, c, w, b)
    NbTasksProc[i] + +;
    NbTasksSent + +;
    IdleProc[i] = max {start_comm_date + c_i + w_i, IdleProc[i] + w_i} ;
    IdleComm = start_comm_date + c_i ;
```

Figure 7: Sketch of list-based heuristics.

5.1.1 Simple selection functions

A natural idea for choosing among available processors, is to select the one with the highest computing speed, or the one with the smallest communication cost. This is formally defined in the following naive heuristics:

```
MAX-SPEED-SEL(...)
    t = i_0 such that w_{i_0} = min {w_i, i such that Available[i] ⩽ CurrentDate}
    Return(P_t, CurrentDate)
```

```
MAX-BW-SEL(...)
    t = i_0 such that c_{i_0} = min {c_i, i such that Available[i] ⩽ CurrentDate}
    Return(P_t, CurrentDate)
```

These selection functions lead to the heuristics denoted by min_c and min_w in the following. Note that there is always at least one processor P_i such that $\mathsf{Available}[i] \leqslant \mathsf{CurrentDate}$ because we set $\mathsf{CurrentDate} = \max\{\min_i\{\mathsf{Available}[i]\}, \mathsf{IdleComm}\}$ just before the call to the selection function.

5.1.2 Loss evaluation

This heuristic is based on the evaluation of the gain and loss of a decision to schedule a task a processor. This kind of approach is very close to the commonly used *sufferage* heuristic [13, 25] and may avoid misleading choices like the one depicted on Figure 8. This heuristic may not communicate a task as soon as possible and wait for a better available slave instead, so it is not a "real" list heuristic, even if it fits in the sketch depicted in Figure 7.

Assume that we decide, at a given time step t when the communication medium is free, to schedule the next communication as a transfer to P_i. We earn one task, but it is possible that another processor P_j becomes starving between t and the end of the communication to P_i ($\mathsf{Available}[i] + c_i$).

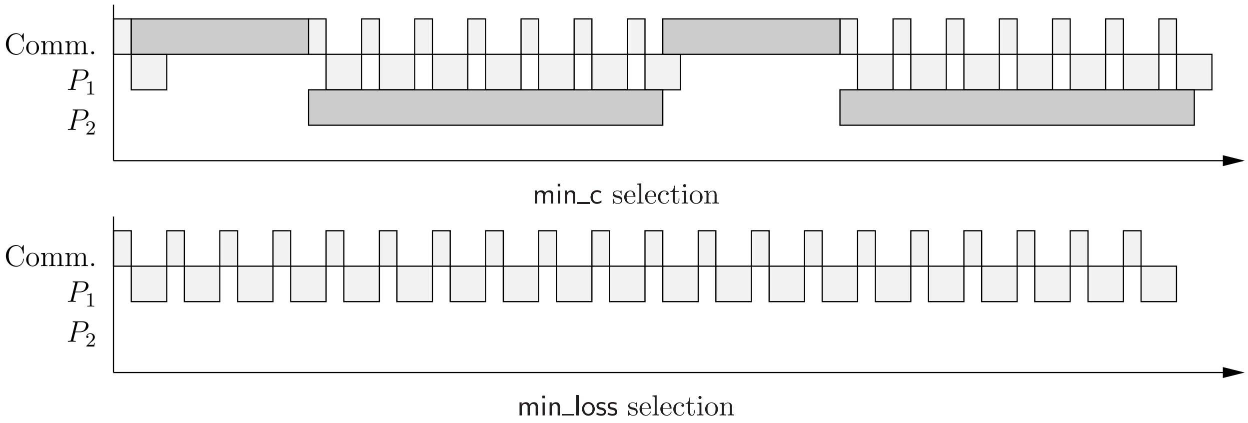

Figure 8: Simple instance with two processors ($b_1 = b_2 = 1$, $c_1 = 1$, $c_2 = 10$, $w_1 = 2$, $w_2 = 20$). min_c may perform a very bad choice. Leaving the communication medium idle for a while may lead to much better results.

We can compute the number of tasks that could have been performed by P_j during this interval in such a case:

$$\mathsf{loss}_{i,j} = \frac{\mathsf{Available}[i] + c_i - \mathsf{IdleProc}[j]}{w_j}, \qquad \text{if } \mathsf{Available}[i] + c_i - \mathsf{IdleProc}[j] \geqslant 0$$

We suppose here that $\mathsf{IdleProc}[j]$ is at least the current date t for all processors: there is no reason to take into account the starvation of a processor due to previous scheduling decisions. So the total loss due to the decision to send a task to P_i is:

$$\mathsf{loss}_i = \sum_{j, \mathsf{Available}[i]+c_i-\mathsf{IdleProc}[j]>0} \mathsf{loss}_{i,j} = \sum_j \max\left\{\frac{\mathsf{Available}[i] + c_i - \mathsf{IdleProc}[j]}{w_j}, 0\right\}$$

This leads to the heuristic min_loss based on the following selection function.

MIN-LOSS-SEL(...)
Return($P_{i_0}, \max\{\mathsf{Available}[i_0], \mathsf{CurrentDate}\}$) such that $\mathsf{loss}(i_0) = \min_i \{\mathsf{loss}(i)\}$

5.1.3 Minimum Completion Time

We adapt here a widely used scheme for list-based heuristics: we choose the processor which will finish to process the task the earliest, given previous scheduling decisions. The completion time of a new task for processor P_i is:

$$\mathsf{CT}_i = \max\left\{\mathsf{Available}[i] + c_i, \mathsf{IdleProc}[i]\right\} + w_i$$

The heuristic based on the following selection function is denoted by mct in the following.

MCT-SEL(...)
Return($P_{i_0}, \max\{\mathsf{Available}[i_0], \mathsf{CurrentDate}\}$) such that $\mathsf{CT}(i_0) = \min\{\mathsf{CT}_i\}$

5.1.4 Using the optimal schedule for unbounded buffers

For this heuristic, denoted by 1D, the selection function is made up using the optimal schedule without buffer constraints, described in [5, 1]. This optimal schedule is pre-computed during an initialization phase. The framework of this heuristic is the following:

1. Choose the resources that will participate to the computation (some processors may be discarded), according to the formulas in [5, 1].
2. Choose the size of the period.
3. Compute the number of tasks sent to each slave during the period, according to the formulas in [5, 1].
4. Order the emission of these tasks according to a 1D load-balancing distribution (see example on Figure 9 or [4]).
5. Follow this emission order repeatedly.

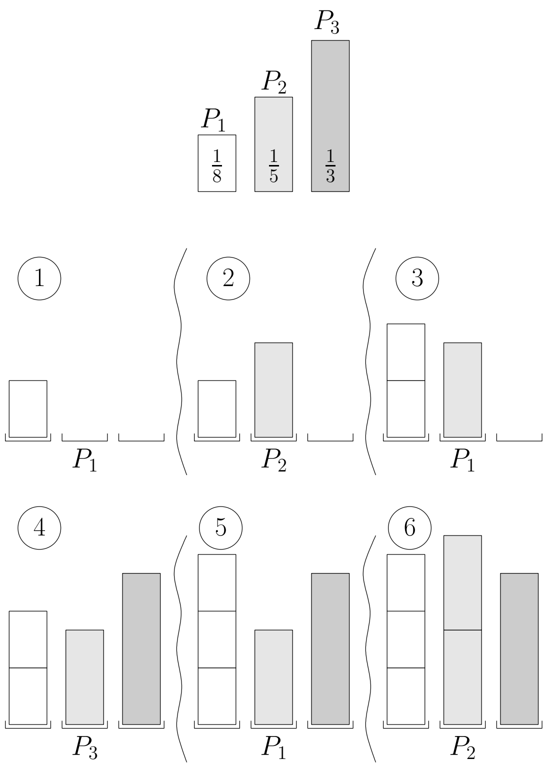

Figure 9: Example of 1D load-balancing with three processors ($w_1 = \frac{1}{8}, w_2 = \frac{1}{5}, w_3 = \frac{1}{3}$). At each step, we choose the target processor (the one which will process the following task) such that the maximum computing time of every processors ($M = \max_i\{n_i \cdot w_i\}$ where n_i is the number of tasks assigned to processor P_i) is minimized. For the first step, it is obviously the one with the smallest w_i (here P_1). For the second task, if we choose P_1 again, this gives $M = \frac{1}{4}$, if we choose P_2, we get $M = \frac{1}{5}$ and with P_3, $M = \frac{1}{3}$, so the next target is P_2... This gives the emission order which is used in the 1D heuristic.

5.2 Simulation platforms

The platform consists of a master and several slaves. The different parameters to take into account are the following:

Number of slaves We performed the experiments with a small number of processors (5) and with a larger number (20). As the results are similar, we only present them in the latter case, where heterogeneity is more likely to play a part.

Sending and computing speeds These parameters were chosen randomly with a Gaussian distribution in the interval [50,150].

Number of tasks As we simulate the scheduling of a fixed number of tasks on a platform, we have to choose the number of tasks to be scheduled. We let this number vary from 10 to 20,480. Note that a small number of tasks is more significant for makespan minimization, while a large number of tasks is relevant for throughput optimization.

Size of the buffers We perform experiments for buffer sizes going from 1 (no overlap between communication and computation) to 32 (almost no limitation on memory).

5.3 Simulation results

To compare the performance of the different heuristics, we compute their performance ratio, defined as the ratio of the number of tasks processed by the slaves over the total time spent to process these tasks. We cannot compute the optimal performance ratio in the case of bounded buffers, but we normalize results by plotting the performance ration of the heuristics over the optimal performance ratio in the absence of memory limitation. The latter performance ration is known from [5, 1] and can be expressed as $\sum_i 1/w_i$ in the case where $\sum_i c_i/w_i \leqslant 1$ (for other cases, the expression is more complicated, because some processors are discarded: see [5, 1]).

Figure 10 presents the results in the case of a single buffer, for a varying number of tasks. Figure 11 shows the results for a fixed number of tasks (20,480), for a varying size of buffer.

Most scheduling heuristics try to greedily minimize the completion time of each task. Even if some variants exist to cope with task affinity or misleading greedy decisions (like *sufferage*), none of these heuristics is efficient in the situation where communications form the master are the bottleneck. Surprisingly, the simplest heuristic (min_c) outperforms the more involved ones (like min_loss and mct), and achieves very good results in all situations: min_c always has the best performances when trying to minimize the makespan in the single buffer case; it reaches 90% of the optimal throughput in the single buffer case, and more than 99% of this bound when the size of the buffer is greater than 2.

Another surprising conclusion is that min_c reaches the optimal unbounded throughput with only a few buffers. When the constraint on buffers are less tight, the 1D heuristic performs very good. This is not surprising as it is an algorithm leading to the best throughput in the unconstrained case, and the way in which we shuffle the emissions aims at balancing the load of the slaves at each time-step.

The good performances of min_c can be explained as follows: if we send a task to a processor P_i with a small c_i and a big w_i, the communication medium will be busy during a short time, and P_i spends a lot of time processing the task; we are able to perform many other communications during this computation. Conversely, if we send a task to a processor with a small c_i and a small w_i, this processor is likely to process the task quickly and to be chosen again soon as a future target: this leads to a larger share of the communication medium for P_i but since it has a small

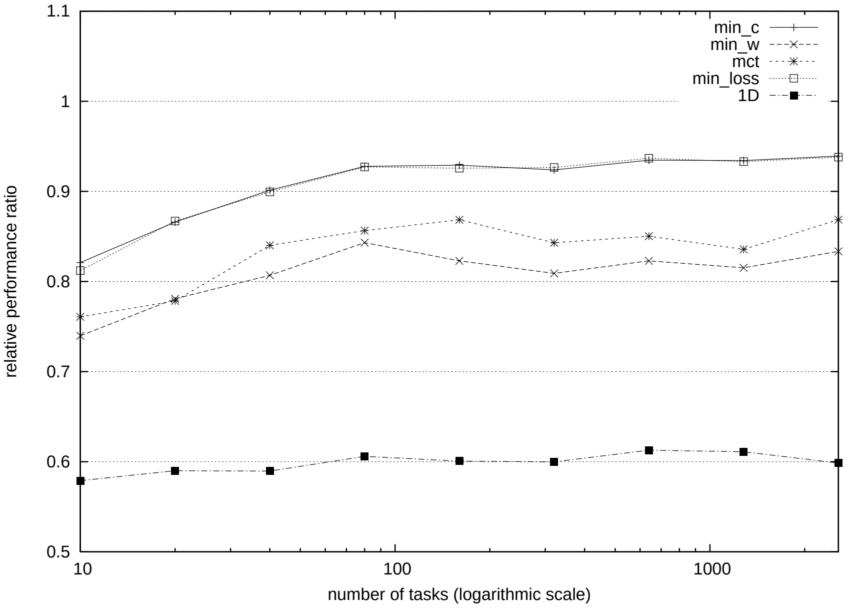

Figure 10: Performance for a single buffer

w_i, it contributes to a big fraction of the total throughput of the platform. In conclusion, sending a task to a slave processor with a small c_i is never a bad choice, regardless of its computing power.

6 Conclusion

In this paper, we have studied the allocation of a large number of independent, equal-sized tasks, on simple star platforms, under different application models and different objective functions. We have also studied the same problem in the context of divisible tasks. In all these situations memory limitations lead to NP-completeness results. We believe that the classification of these scheduling problems will prove useful to the community, and will foster more work on the open problems listed in Table 1.

For the objective of makespan minimization, we have been able to derive an approximation algorithm. We have introduced several heuristics which have been evaluated and compared by performing extensive simulations. Unexpectedly, classical list-based scheduling heuristics that aim at greedily minimizing the completion time of each task are outperformed by the simplest heuristic that consists in delegating data to the available processor that has the smallest communication time, regardless of its computation power.

References

[1] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Trans. Parallel Distributed Systems*, 15(4):319–330, 2004.

[2] G. Barlas. Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees. *IEEE Trans. Parallel Distributed Systems*, 9(5):429–441, 1998.

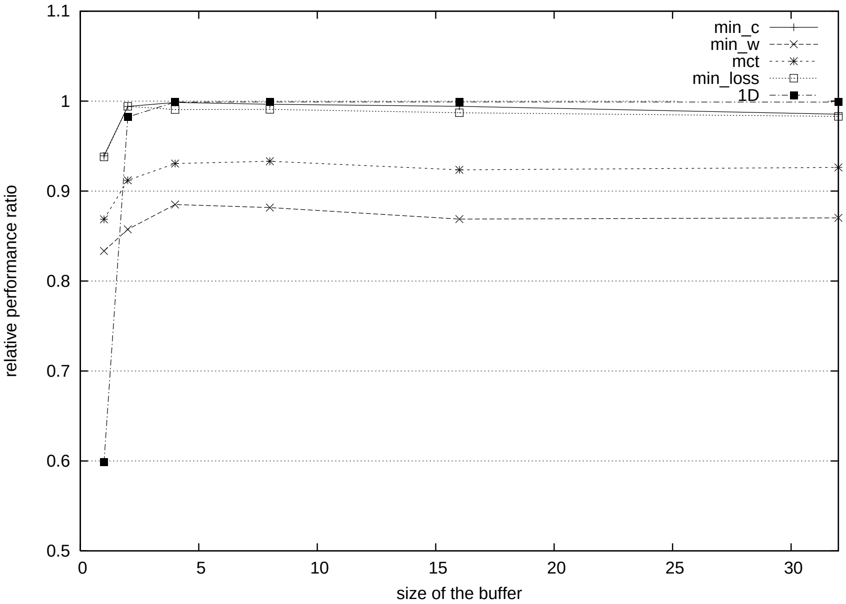

Figure 11: Performance as a function of the buffer size.

[3] A. Batat and D. G. Feitelson. Gang scheduling with memory considerations. In *14th International Parallel and Distributed Processing Symposium (IPDPS'2000)*. IEEE Computer Society Press, 2000.

[4] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert. A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). *IEEE Trans. Computers*, 50(10):1052–1070, 2001.

[5] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium (IPDPS'2002)*. IEEE Computer Society Press, 2002.

[6] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms. Technical Report 2004-20, LIP, ENS Lyon, France, Apr. 2004.

[7] O. Beaumont, A. Legrand, and Y. 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.

[8] O. Beaumont, A. Legrand, and Y. Robert. Scheduling divisible workloads on heterogeneous platforms. *Parallel Computing*, 29:1121–1152, 2003.

[9] V. Bharadwaj, D. Ghose, and V. Mani. Optimal Sequencing and Arrangement in Single-Level Tree Networks with Communication Delays. *IEEE transactions on parallel and distributed systems*, 5(9), 1994.

[10] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.

[11] V. Bharadwaj, D. Ghose, and T. Robertazzi. A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–18, 2003.

[12] L. Carter, H. Casanova, J. Ferrante, and B. Kreaseck. Autonomous protocols for bandwidth-centric scheduling of independent-task applications. In *International Parallel and Distributed Processing Symposium IPDPS'2003*. IEEE Computer Society Press, 2003.

[13] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Ninth Heterogeneous Computing Workshop*, pages 349–363. IEEE Computer Society Press, 2000.

[14] E. G. Coffman. *Computer and job-shop scheduling theory.* John Wiley & Sons, 1976.

[15] C. D. P. Dimitrios S. Nikolopoulos. Adaptive scheduling under memory pressure on multiprogrammed clusters. In *Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'02)*. IEEE Computer Society, May 2002.

[16] M. Drozdowski and P. Wolniewicz. Divisible Load Scheduling in Systems with Limited Memory. *Cluster Computing*, 6(1):19–29, 2003.

[17] P.-F. Dutot. Master-slave tasking on heterogeneous processors. In *International Parallel and Distributed Processing Symposium IPDPS'2003*. IEEE Computer Society Press, 2003.

[18] P.-F. Dutot. Complexity of master-slave tasking on heterogeneous trees. *European Journal of Operational Research*, 2004. Special issue on the Dagstuhl meeting on Scheduling for Computing and Manufacturing systems (to appear).

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1991.

[20] D. Ghose and T. Robertazzi, editors. *Special issue on* Divisible Load Scheduling. Cluster Computing, 6, 1, 2003.

[21] J. P. Goux, S. Kulkarni, J. Linderoth, and M. Yoder. An enabling framework for master-worker applications on the computational grid. In *Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*. IEEE Computer Society Press, 2000.

[22] C. Hanen and A. Munier. Cyclic scheduling on parallel processors: an overview. In P. Chrétienne, E. G. Coffman, J. K. Lenstra, and Z. Liu, editors, *Scheduling Theory and its Applications*, pages 193–226. John Wiley & Sons, 1994.

[23] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive scheduling for master-worker applications on the computational grid. In R. Buyya and M. Baker, editors, *Grid Computing - GRID 2000*, pages 214–227. Springer-Verlag LNCS 1971, 2000.

[24] B. Kreaseck. *Dynamic autonomous scheduling on Heterogeneous Systems.* PhD thesis, University of California, San Diego, 2003.

[25] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Eight Heterogeneous Computing Workshop*, pages 30–44. IEEE Computer Society Press, 1999.

[26] T. Robertazzi. Divisible Load Scheduling. URL: `http://www.ece.sunysb.edu/~tom/dlt.html`.

[27] T. Robertazzi. Ten reasons to use divisible load theory. *IEEE Computer*, 36(5):63–68, 2003.

[28] G. Shao, F. Berman, and R. Wolski. Master/slave computing on the grid. In *Heterogeneous Computing Workshop HCW'00*. IEEE Computer Society Press, 2000.

[29] J. B. Weissman. Scheduling multi-component applications in heterogeneous wide-area networks. In *Heterogeneous Computing Workshop HCW'00*. IEEE Computer Society Press, 2000.