Scheduling divisible loads with return messages on heterogeneous master-worker platforms

Olivier Beaumont, Loris Marchal, Yves Robert

Laboratoire de l'Informatique du Parallélisme

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

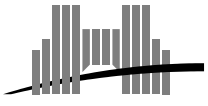
Scheduling divisible loads with return messages on heterogeneous master-worker platforms

Olivier Beaumont,
Loris Marchal,
Yves Robert

Mai 2005

Research Report N° 2005-21

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`



Scheduling divisible loads with return messages on heterogeneous master-worker platforms

Olivier Beaumont, Loris Marchal, Yves Robert

Mai 2005

Abstract

In this paper, we consider the problem of scheduling independent tasks, or divisible loads, onto an heterogeneous star platform, with both heterogeneous computing and communication resources. We consider the case where the workers, after processing the tasks, send back some results to the master processor. This corresponds to a more general framework than the one used in many divisible load papers, where only forward communications are taken into account. To the best of our knowledge, this paper constitutes the first attempt to derive optimality results under this general framework (forward and backward communications, heterogeneous processing and communication resources). We prove that it is possible to derive the optimal solution both for LIFO and FIFO distribution schemes. Nevertheless, the complexity of the general problem remains open: we also prove in the paper that the optimal distribution scheme may be neither LIFO nor FIFO.

Keywords: Scheduling, divisible load, master-worker platform, heterogeneous cluster.

Résumé

Nous nous intéressons ici au problème de l'ordonnancement de tâches divisibles sur une plate-forme de calcul hétérogène en étoile. L'hétérogénéité concerne les ressources de calcul comme les liens de communication. Nous considérons le cas où les esclaves, après avoir effectué des tâches, doivent envoyer les résultats obtenus au processeur maître. Ceci correspond à un schéma plus général que celui utilisé dans l'essentiel de la littérature sur les tâches divisibles, où seules les communications du maître vers les esclaves sont prises en compte. À notre connaissance, ceci est la première tentative pour obtenir des résultats d'optimalité pour ce problème général (communications dans les deux sens, hétérogénéité pour les puissances de calcul et de communication). Nous montrons qu'il est possible de concevoir des solutions optimales pour les schémas de communication FIFO et LIFO. Cependant, la complexité du problème reste ouverte : nous montrons également que le schéma de communication optimal peut n'être ni LIFO ni FIFO.

Mots-clés: Ordonnancement, tâches divisibles, plate-forme maître-esclave, grappe hétérogène.

1 Introduction

This paper deals with scheduling independent tasks on heterogeneous platforms. Quite naturally, we target a master-worker implementation where the master initially holds (or generates data for) a large collection of identical tasks which will be executed by the workers. In the end, results will be returned by the workers to the master. Each worker has a different computational speed, and each master-worker link has a different bandwidth, thereby making the platform fully heterogeneous. The master-worker platform is often referred to as a *star network* in the literature, and is said to reduce to a *bus network* when all master-worker links have the same bandwidth.

The scheduling problem is first to decide how many tasks the master sends to each worker, and in which order. After receiving its share of the data, each worker executes the corresponding tasks and returns the results to the master. Again, the ordering of the return messages must be decided by the scheduler.

Assume that there are N tasks to be distributed by the master. Each worker P_i will execute a fraction $\alpha_i N$ of these tasks. An important relaxation of the problem is to allow the α_i be rational numbers instead of integers. The main motivation for this relaxation is technical: the scheduling is better amenable to an analytical solution if rounding problems are ignored. From a practical point of view, this simplification is harmless. Indeed, we assume that a large number of tasks is processed (otherwise why deploy a parallel implementation?). The quality of the schedule is not affected if, say, a rational value of $\alpha_1 = 1027,3$ tasks for worker P_1 is transformed into $\alpha_1 = 1027$ tasks. We can randomly assign the few tasks that would remain to be executed after rounding the rational solution.

From a theoretical perspective, this relaxation is important, because it shows the equivalence between scheduling independent tasks and *divisible load scheduling*, or DLS for short. As their name suggests, divisible load applications can be divided among worker processes arbitrarily, i.e. 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. In practice, the DLS model is an approximation of applications that consist of large numbers of identical, low-granularity computations. The former number of independent tasks N corresponds in the DLS model to a total number of load units W to be distributed to the workers, where $W = Nu$ if each task represents u load-units. Allocating rational values of load-units is equivalent to distributing rational numbers of tasks.

The DLS model has been widely studied in the last several years, after having been popularized by the landmark book written in 1996 by Bharadwaj, Ghose, Mani and Robertazzi [7]. The DLS model provides a practical framework for the mapping on independent tasks onto heterogeneous platforms, and has been applied to a large spectrum of scientific problems: see Section 6 for further details.

From a theoretical standpoint, the success of the DLS model is mostly due to its analytical tractability. Optimal algorithms and closed-form formulas exist for important instances of the divisible load problem. A famous example is the closed-form formula given in [5, 7] for a bus network. The hypotheses are the following:

1. the master distributes the load to the workers, but no results are returned to the master
2. a linear cost model is assumed both for computations and for communications
3. all master-worker communication links have same bandwidth (but the workers have different processing speeds)

The proof to derive the closed-form formula proceeds in several steps: it is shown that in an optimal solution: (i) all workers participate in the computation, then that (ii) they never stop working after having received their data from the master, and finally that (iii) they all terminate the execution of their load simultaneously. These conditions give rise to a set of equations from which the optimal value α_i can be computed for each worker P_i.

Extending this result to a star network (with different master-worker link bandwidths), but still (1) without return messages and (2) with a linear cost model, has been achieved only recently [6].

The proof basically goes along the same steps as for a bus network, but the main additional difficulty was to find the optimal ordering of the messages from the master to the workers. It turns out that the best strategy is to serve workers with larger bandwidth first, independently of their computing power.

The next natural step is to include return messages in the picture. This is very important in practice, because in most applications the workers are expected to return some results to the master. When no return messages are assumed, it is implicitly assumed that the size of the results to be transmitted to the master after the computation is negligible, and hence has no (or very little) impact on the whole DLS problem. This may be realistic for some particular DLS applications, but not for all of them. For example suppose that the master is distributing files to the workers. After processing a file, the worker will typically return results in the form of another file, possibly of shorter size, but still non-negligible. In some situations, the size of the return message may even be larger than the size of the original message: for instance the master initially scatters instructions on some large computations to be performed by each worker, such as the generation of several cryptographic keys; in this case each worker would receive a few bytes of control instructions and would return longer files containing the keys.

Because it is very natural and important in practice, several authors have investigated the problem with return messages: see the papers [4, 13, 22, 3, 1], whose contents are surveyed in Section 6 on related work. However, all the results obtained so far are very partial. Intuitively, there are hints that suggest that the problem with return results is much more complicated. The first hint lies in the combinatorial space that is open for searching the best solution. There is no reason for the ordering of the initial messages sent by the master to be the same as the ordering for the messages returned to the master by the workers after the execution. In some situations a FIFO strategy (the worker first served by the master is the first to return results, and so on) may be preferred, because it provides a smooth and well-structured pipelining scheme. In other situations, a LIFO strategy (the other way round, first served workers are the last to return results) may provide better results, because faster workers would work a longer period if we serve them first. True, but what if these fast workers have slow communication links? In fact, and here comes the second hint, it is not even clear whether all workers should be enrolled in the computation by the master. This is in sharp contrast to the case without return messages, where it is obvious that all workers should participate.

To the best of our knowledge, the complexity of the problem remains open, despite the simplicity of the linear cost model. In [1], Adler, Gong and Rosenberg show that all FIFO strategies are equally performing on a bus network, but even the analysis of FIFO strategies is an open problem on a star network.

The main contributions of this paper are the characterization of the best FIFO and LIFO strategies on a star network, together with an experimental comparison of them. While the study of LIFO strategies nicely reduces to the original problem without return messages, the analysis of FIFO strategies turns out to be more involved; in fact, the optimal FIFO solution may well not enroll all workers in the computations.

Admittedly, the complexity of the DLS problem with return messages remains open: there is no a priori reason that either FIFO or LIFO strategies would be superior to solutions where the ordering of the initial messages and that of return messages are totally uncorrelated (and we give an example of such a situation in Section 2). Still, we believe that our results provide an important step in the understanding of this difficult problem, both from a theoretical and practical perspective. Indeed, we have succeeded in characterizing the best FIFO and LIFO solutions, which are the most natural and easy-to-implement strategies.

The rest of the paper is organized as follows. In Section 2, we state precisely the DLS problem, with all application and platform parameters, and we give two examples showing the difficulty of the problem: in the first example, not all processors participate in the optimal solution; in the second example, the optimal solution is obtained using neither a FIFO nor a LIFO strategy. Section 3 deals with the characterization of the best LIFO solution. Section 4 is the counterpart for FIFO strategies. We provide an experimental comparison of LIFO and FIFO strategies in Section 5. Section 6 is devoted to an overview of related work. Finally, we state some concluding

remarks in Section 7.

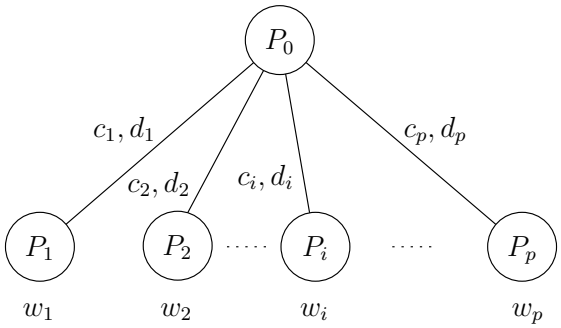


Figure 1: Heterogeneous star graph, with the linear cost model.

2 Framework

2.1 Problem parameters

As illustrated in Figure 1, a *star network* $\mathcal{S} = \{P_0, P_1, P_2, \ldots, P_p\}$ is composed of a master P_0 and of p workers P_i, $1 \leq i \leq p$. There is a communication link from the master P_0 to each worker P_i. In the linear cost model, each worker P_i has a (relative) computing power w_i: it takes $X.w_i$ time units to execute X units of load on worker P_i. Similarly, it takes $X.c_i$ time units to send the initial data needed for computing X units of load from P_0 to P_i, and $X.d_i$ time units to return the corresponding results from P_i to P_0. Without loss of generality we assume that the master has no processing capability (otherwise, add a fictitious extra worker paying no communication cost to simulate computation at the master). Note that a *bus network* is a star network such that all communication links have the same characteristics: $c_i = c$ and $d_i = d$ for each worker P_i, $1 \leq i \leq p$.

It is straightforward to specify the parameters in terms of independent tasks rather than in terms of DLS load units. Let u be the number of floating-point operations to process a task, δ_{init} be the size (number of bytes) of the input file needed to process a task, and δ_{return} be the size (number of bytes) of the output file generated by the processing of a task. Let s_i be the speed of processor P_i (in flops). Finally, let b_i denote the bandwidth (in bytes per second) of the communication link between P_0 and P_i. We have the relations

$$w_i = \frac{u}{s_i}, \quad c_i = \frac{\delta_{\text{init}}}{b_i}, \quad d_i = \frac{\delta_{\text{return}}}{b_i}$$

These relation show that the quantity

$$\frac{d_i}{c_i} = \frac{\delta_{\text{return}}}{\delta_{\text{init}}} = z$$

is a constant z that depends on the application but not on the selected worker. In other words, workers who communicate faster with the master for the initial message will also communicate faster for the return message. In the following, we keep using both values d_i and c_i, because many results are valid even without the relation $d_i = zc_i$, and we explicitly mention when we use this relation.

Finally, we use the standard model in DLS problem for communications: the master can only send data to, and receive data from, a single worker at a given time-step. A given worker cannot start execution before it has terminated the reception of the message form the master; similarly, it cannot start sending the results back to the master before finishing the computation. However, there is another classic hypothesis in DLS papers which we do not enforce, namely that there is

no idle time in the operation of each worker. Under this assumption, a worker starts computing immediately after having received its initial message, which is no problem, but also starts returning the results immediately after having finished its computation: this last constraint does reduce the solution space arbitrarily. Indeed, it may well prove useful for a worker P_i to stay idle a few steps before returning the results, waiting for the master to receive the return message of another worker $P_{q'}$. Of course we could have given more load to P_i to prevent him from begin idle, but this would have implied a longer initial message, at the risk of delaying the whole execution scheme. Instead, we will tackle the problem in its full generality and allow for the possibility of idle times (even if we may end by proving that there is no idle time in the optimal solution).

The objective function is to maximize the number of load units that are processed within T time-units. Let α_i be the number of load units sent to, and processed by, worker P_i within T time-units. Owing to the linear cost model, the quantity

$$\frac{\sum_{i=1}^{p} \alpha_i}{T} = \rho$$

is a constant (see Section 2.2 for a proof), and corresponds to the achieved throughput, which we aim at maximizing.

2.2 Linear program for a given scenario

Given a star platform with p workers, and parameters w_i, c_i, d_i, $1 \leq i \leq p$, how can we compute the optimal throughput?

First we have to decide which workers are enrolled. Next, given the set of participating workers, we have to decide for the ordering of the initial messages. Finally we have to decide for the ordering of the return messages. Altogether, there is a finite (although exponential) number of scenarios, where a *scenario* refers to a schedule with a given set of participating workers and a fixed ordering of initial and return messages. Then, the next question is: how can we compute the throughput for a given scenario?

Without loss of generality, we can always perform all the initial communications as soon as possible. In other words, the master sends messages to the workers without interruption. If this was not the case, we would simply shift ahead some messages sent by the master, without any impact on the rest of the schedule. Obviously, we can also assume that each worker initiates its computation as soon as it has received the message from the master. Finally, we can always perform all the return communications as late as possible. In other words, once the master starts receiving data back from the first worker, it receives data without interruption until the end of the whole schedule. Again, if this was not the case, we would simply delay the first messages received by the master, without any impact on the rest of the schedule. Note that idle times can still occur in the schedule, but only between the end of a worker's computation and the date at which it starts sending the return message back to the master.

The simplest approach to compute the throughput ρ for a given scenario is to solve a linear program. For example, assume that we target a LIFO solution involving all processors, with the ordering $P_1, P_2, \ldots, P_p$, as outlined in Figure 2. With the notations of Section 2.1 (parameters w_i, c_i, d_i and unknowns α_i, ρ), worker P_i:

- starts receiving its initial message at time $t_i^{\text{recv}} = \sum_{j=1}^{i-1} \alpha_j c_j$
- starts execution at time $t_i^{\text{recv}} + \alpha_i c_i$
- terminates execution at time $t_i^{\text{term}} = t_i^{\text{recv}} + \alpha_i c_i + \alpha_i w_i$
- starts sending back its results at time $t_i^{\text{back}} = T - \sum_{j=1}^{i} \alpha_j d_j$

Here T denotes the total length of the schedule. The idle time of P_i is $x_i = t_i^{\text{back}} - t_i^{\text{term}}$, and this quantity must be nonnegative. We derive a linear equation for worker P_i:

$$T - \sum_{j=1}^{i} \alpha_j d_j \geq \sum_{j=1}^{i-1} \alpha_j c_j + \alpha_i c_i + \alpha_i w_i$$

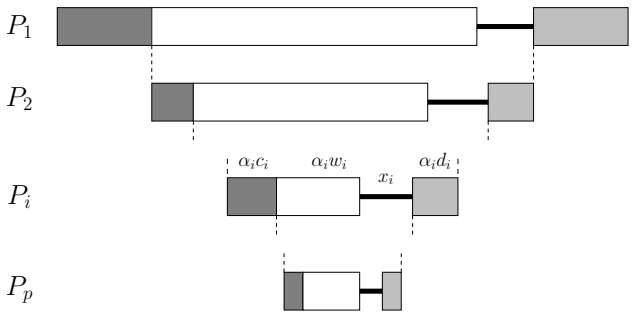


Figure 2: LIFO strategy. Dark grey rectangles (of length $\alpha_q c_q$) represent the initial messages destined to the workers. White rectangles (of length $\alpha_q w_q$) represent the computation on the workers. Light grey rectangles (of length $\alpha_q d_q$) represent the return messages back to the master. Bold lines (of length x_q) represent the idle time of the workers.

Together with the constraints $\alpha_i \geq 0$, we have assembled a linear program, whose objective function is to maximize $\rho(T) = \sum_{i=1}^{p} \alpha_i$. In passing, we check that the value of $\rho(T)$ is indeed proportional to T, and we can safely define $\rho = \rho(1)$ as mentioned before. We look for a rational solution of the linear program, with rational (not integer) values for the quantities α_i and ρ, hence we can use standard tools like Maple [11] or MuPAD [24].

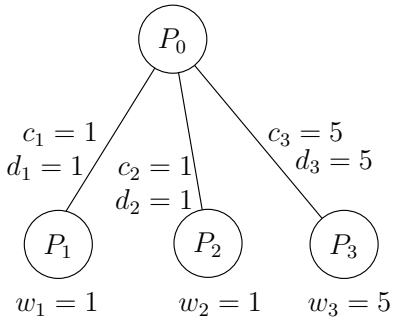
Obviously, this linear programming approach can be applied for any permutation of initial and return messages, not just LIFO solutions as in the above example. Note that it may well turn out that some α_i is zero in the solution returned by the linear program, which means that P_i is not actually involved in the schedule. This observation reduces the solution space: we can *a priori* assume that all processors are participating, and solve a linear program for each pair of message permutations (one for the initial messages, one for the return messages). The solution of the linear program will tell us which processors are actually involved in the optimal solution for this permutation pair.

For a given scenario, the cost of this linear programming approach may be acceptable. However, as already pointed out, there is an exponential number of scenarios. Worse, there is an exponential number of LIFO and FIFO scenarios, even though there is a single permutation to try in these cases (the ordering of the return messages is the reverse (LIFO) or the same (FIFO) as for the initial messages). The goal of Sections 3 and 4 is to determine the best LIFO and FIFO solution in polynomial time.

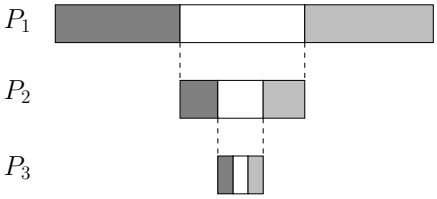
2.3 Counter-examples

In Figure 3 we outline an example where not all processors participate in the optimal solution. The platform has three workers, as shown in Figure 3(a). The best throughput that can be achieved using all the three workers is obtained via the LIFO strategy represented in Figure 3(b), and is $\rho = 61/135$. However, the FIFO solution which uses only the first two workers P_1 and P_2 achieves a better throughput $\rho = 1/2$. To derive these results, we have used the linear programming approach for each of the 36 possible permutation pairs. It is very surprising to see that the optimal solution does not involve all workers under a linear cost model (and to the best of our knowledge this is the first known example of such a behavior).

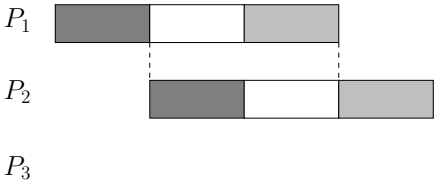
Next, in Figure 4, we outline an example where the best throughput is achieved using neither a FIFO nor a LIFO approach. Instead, the optimal solution uses the initial ordering (P_1, P_2, P_3) and the return ordering (P_2, P_1, P_3). Again, we have computed the throughput of all possible permutation pairs using the linear programming approach.



(a) Platform

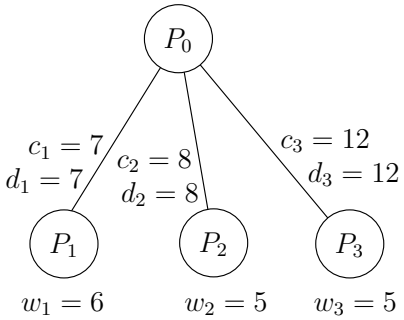


(b) LIFO, throughput $\rho = 61/135$

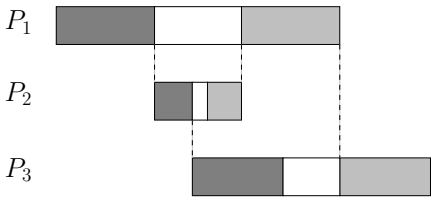


(c) FIFO with 2 processors, optimal throughput $\rho = 1/2$

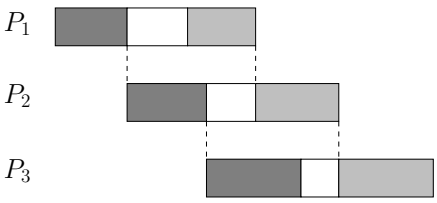
Figure 3: The best schedule with all the three workers (shown in (b)) achieves a lower throughput than when using only the first two workers (as shown in (c)).



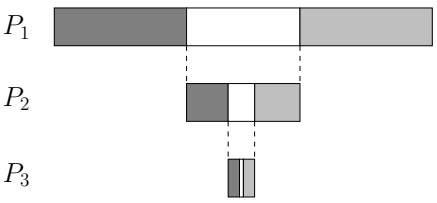
(a) Platform



(b) Optimal schedule ($\rho = 38/499 \approx 0.076$)



(c) FIFO schedule ($\rho = 47/632 \approx 0.074$)



(d) LIFO schedule ($\rho = 43/580 \approx 0.074$)

Figure 4: An example where the optimal solution is neither FIFO nor LIFO.

3 LIFO strategies

In this section, we concentrate on LIFO strategies, where the processor that receives the first message is also the last processor that sends its results back to the master, as depicted in Figure 2. We keep the notations used in Section 2.2, namely w_i, c_i, d_i and α_i for each worker P_i.

In order to determine the optimal LIFO ordering, we need to answer the following questions:

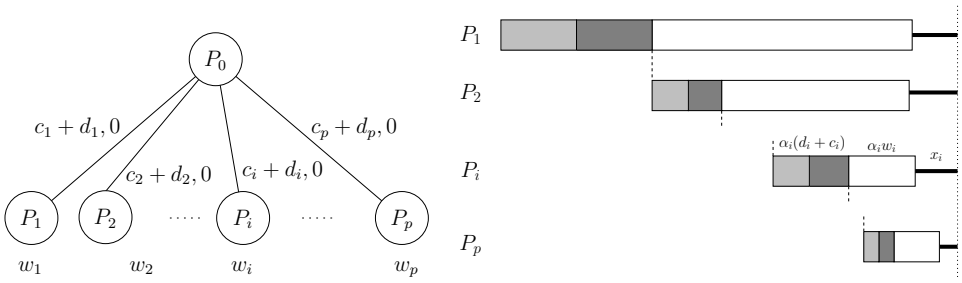
- What is the subset of participating processors?
- What is the ordering of the initial communications?
- What is the idle time x_i of each participating worker?

The following theorem answers these questions and provides the optimal solution for LIFO strategies:

Theorem 1. *In the optimal LIFO solution, then*

- *All processors participate to the execution*
- *Initial messages must be sent by non-decreasing values of* $c_i + d_i$
- *There is no idle time, i.e.* $x_i = 0$ *for all* i.

Proof. Let us assume that $c_1 + d_1 \leq c_2 + d_2 \leq \ldots c_p + d_p$. We will prove that the optimal solution for a LIFO strategy on the star network of Figure 1 has the same throughput as the optimal solution for a classical DLS problem, without return messages, on the platform represented in Figure 5(a). The key idea is to transform the LIFO schedule of Figure 2 into the schedule of Figure 5(b). Because the optimal schedule for the latter problem is well-known, this will lead to the desired result.



(a) Modified platform

(b) Modified schedule

Figure 5: Transforming the platform on Figure 1 and the LIFO schedule into a schedule without return transfers and with same performance.

Indeed, consider the optimal solution for the LIFO problem, and suppose that it sends α_{σ_1} tasks to P_{σ_1}, then α_{σ_2} tasks to P_{σ_2},..., and finally α_{σ_p} tasks to P_{σ_p}, where σ is a permutation of the integers between 1 and p. Note that since α_{σ_i} may be 0, this formulation is general and includes the case where some processors are not used in the optimal solution. Let us also denote by x_i the idle time on processor P_i in the optimal solution.

Then, from the discussion in Section 2.2 (see also Figure 2), we check that the α_i's are the solution of the following (triangular and non-singular) linear system:

$$\begin{pmatrix} c_{\sigma_1} + w_{\sigma_1} + d_{\sigma_1} & 0 & 0 & \ldots & 0 \\ c_{\sigma_1} & c_{\sigma_2} + w_{\sigma_2} + d_{\sigma_2} & 0 & \ldots & 0 \\ \vdots & c_{\sigma_2} & c_{\sigma_3} + w_{\sigma_3} + d_{\sigma_3} & \ddots & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ c_{\sigma_1} & c_{\sigma_2} & c_{\sigma_3} & \ldots & c_{\sigma_p} + w_{\sigma_p} + d_{\sigma_p} \end{pmatrix} \begin{pmatrix} \alpha_{\sigma_1} \\ \alpha_{\sigma_2} \\ \vdots \\ \\ \vdots \\ \alpha_{\sigma_p} \end{pmatrix} = \begin{pmatrix} T - x_{\sigma_1} \\ T - x_{\sigma_2} \\ \vdots \\ \\ \vdots \\ T - x_{\sigma_p} \end{pmatrix}$$

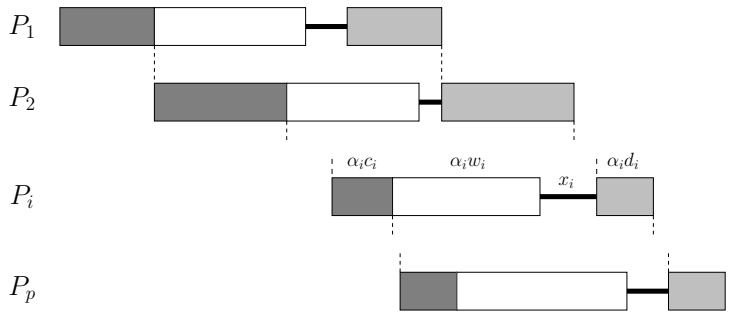


Figure 6: FIFO strategies.

Given the x_i's, the solution of this linear system also provides the schedule shown in Figure 5(b) for the classical DLS problem without return results on the platform depicted in Figure 5(a) This problem with linear costs and without return results has been extensively studied (see [6] and the references therein). It is known that $\sum \alpha_i$ is maximal if and only if

- All processors participate to the execution
- Messages must be sent by non-increasing values of $c_i + d_i$ (i.e. $\forall i, \quad \sigma_i = i$)
- There is no idle time, i.e. $x_i = 0$ for all i

This achieves the proof of the theorem. □

Theorem 1 shows that the optimal LIFO solution is obtained by the resolution of the following linear system:

$$\begin{pmatrix} c_1+w_1+d_1 & 0 & 0 & \dots & 0 \\ c_1 & c_2+w_2+d_2 & 0 & \dots & 0 \\ \vdots & c_2 & c_3+w_3+d_3 & \ddots & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ c_1 & c_2 & c_3 & \dots & c_p+w_p+d_p \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \vdots \\ \alpha_p \end{pmatrix} = T \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Rather than directly solving the system and computing $\rho = \sum_{i=1}^{p} \alpha_i$, which would require $O(p^2)$ operations, we can use the closed-form formula provided in [23, equations (4) and (5)]. We have proven the following result:

Proposition 1. *The optimal LIFO strategy (and the corresponding throughput) can be determined in linear time $O(p)$.*

Finally, we point out that we have not used the relation $d_i = zc_i$ in this section. All results for LIFO solutions apply to platforms with uncorrelated c_i and d_i parameters.

4 FIFO strategies

In this section, we concentrate on FIFO strategies, where the processor that receives the first message is also the first processor to send its results back to the master, as depicted in Figure 6.

We keep the notations used in Sections 2.2 and 3, namely w_i, c_i, d_i and α_i for each worker P_i. The analysis of FIFO strategies is more difficult than the analysis of LIFO strategies: we will show that not all processors are enrolled in the optimal FIFO solution. Throughout this section, we assume that $d_i = zc_i$ for $1 \leq i \leq p$, and also that $z \leq 1$. The case $z > 1$ is *symmetric* in some sense, and will be considered at the end of the section.

Theorem 2. *In the optimal FIFO solution, then*

- *Initial messages must be sent by non-decreasing values of* $c_i + d_i$
- *The set of participating processors is composed of the first* q *processors for the previous ordering, where* q *can be determined in linear time*
- *There is no idle time, i.e.* $x_i = 0$ *or all* i.

Theorem 2 will be proven via several Lemmas. The first one states that there exists an optimal solution where all participating processors work without idle times.

Lemma 1. *There exists an optimal FIFO solution such that*

$$\forall i, \quad \alpha_i x_i = 0.$$

Proof. Consider an optimal FIFO, where the number of participating processors is minimal. To simplify notations, we will assume that this minimal set of participating processors (i.e. the processors such that $\alpha_i > 0$) is $P_1, P_2, \ldots, P_k$. $x_i \geq 0$ denotes the idle time on processor P_i. The α_i's are solution of the following linear system (see Figure 6):

$$A\alpha = T\mathbb{1} - x,$$

where $\alpha = (\alpha_1, \ldots, \alpha_k)^t$, $x = (x_1, \ldots, x_k)^t$, $\mathbb{1} = (1, \ldots, 1)^t$ and

$$A = \begin{pmatrix} c_1 + w_1 + d_1 & d_2 & d_3 & \ldots & d_k \\ c_1 & c_2 + w_2 + d_2 & d_3 & \ldots & d_k \\ \vdots & c_2 & c_3 + w_3 + d_3 & \ddots & \vdots \\ \vdots & \vdots & & \ddots & d_k \\ c_1 & c_2 & c_3 & \ldots & c_k + w_k + d_k \end{pmatrix}$$

To show that A is non-singular, we use the decomposition $A = L + \mathbb{1}d^t$, where

$$L = \begin{pmatrix} c_1 + w_1 & 0 & 0 & \ldots & 0 \\ c_1 - d_1 & c_2 + w_2 & 0 & \ldots & 0 \\ \vdots & c_2 - d_2 & c_3 + w_3 & \ddots & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ c_1 - d_1 & c_2 - d_2 & c_3 - d_3 & \ldots & c_k + w_k \end{pmatrix} \text{ and } d = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_k \end{pmatrix}.$$

The matrix $\mathbb{1}d^t$ is a rank-one matrix, so we can use the Sherman-Morrison formula [16] (indeed, L is a triangular, non-singular matrix and we will prove in Lemma 2 that $1 + d^t L^{-1}\mathbb{1} \neq 0$):

$$A^{-1} = (L + \mathbb{1}d^t)^{-1} = L^{-1} - \frac{L^{-1}\mathbb{1}d^t L^{-1}}{1 + d^t L^{-1}\mathbb{1}}$$

Therefore, $\alpha = A^{-1}(T\mathbb{1} - x)$. More precisely, if we let $A^{-1} = a_{i,j}^{(-1)}$, we have $\alpha_i = \sum_j a_{i,j}^{(-1)}(T - x_j)$ and

$$\sum \alpha_i = b_0 + \sum_j b_j x_j,$$

where $b_0 = T \sum_i \sum_j a_{i,j}^{(-1)}$ and $b_j = -\sum_i a_{i,j}^{(-1)}$.

In order to complete the proof of the lemma, we will prove that if $x_j > 0$, then either we can find a solution processing more tasks, or we can find a solution processing exactly the same number of tasks but using strictly fewer processors. In both cases we would obtain a contradiction. Let us suppose then that there exists j such that $x_j > 0$. The proof depends upon the value of b_j:

- First case: $b_j > 0$
  Consider $x(\epsilon)$ defined by: $\forall j \neq i, \quad x_i(\epsilon) = x_i$ and $x_j(\epsilon) = x_j + \epsilon$. Let
  $$\alpha(\epsilon) = A^{-1}(T\mathbb{1} - x(\epsilon)).$$
  Since by assumption $\alpha_i(0) > 0$, then $\alpha_i(\epsilon) > 0$ if ϵ is sufficiently small, so that the set of $x(\epsilon)_j$'s defines a valid solution. But in this case
  $$\sum \alpha_i(\epsilon) = b_0 + \sum_j b_j x_j(\epsilon) = \sum \alpha_i(0) + b_j\epsilon > \sum \alpha_i(0),$$
  which contradicts our optimality assumption for the values $\alpha_i(0) = \alpha_i$.
- Second case: $b_j < 0$
  The proof is similar, except that we define $x(\epsilon)$ by: $\forall j \neq i, \quad x_i(\epsilon) = x_i$ and $x_j(\epsilon) = x_j - \epsilon$.
- Third case: $b_j = 0$
  We define $x(\epsilon)$ by: $\forall j \neq i, \quad x_i(\epsilon) = x_i$ and $x_j(\epsilon) = x_j - \epsilon$. Let
  $$\alpha(\epsilon) = A^{-1}(T\mathbb{1} - x(\epsilon)).$$
  As long as both $x_j(\epsilon) > 0$ and $\forall i, \quad \alpha_i(\epsilon) > 0$, the $x_i(\epsilon)$'s define a valid and optimal solution, since
  $$\sum \alpha_i(\epsilon) = b_0 + \sum_j b_j x_j(\epsilon) = \sum \alpha_i(0) - b_j\epsilon = \sum \alpha_i(0).$$
  Let us increase the value of ϵ until either $x_j(\epsilon) = 0$ or $\exists i, \quad \alpha_i(\epsilon) = 0$:
  - If $x_j(\epsilon) = 0$, we have found an optimal solution such that $x_j = 0$.
  - If $\exists i, \quad \alpha_i(\epsilon) = 0$, we have found an optimal solution where less than k processors participate to the execution.

□

In the following, we will therefore restrict our search to optimal solutions where no idle time occurs on any processor. In order to keep notations simple, we will assume that the participating processors are $P_1, \ldots, P_k$ and that load units are sent to these processors in the order $P_1, P_2, \ldots, P_k$. The following lemma gives an analytical expression for the total number of load units that can be processed using processors $P_1, \ldots, P_k$ in that order.

Lemma 2. *The total number of load units that can be processed in time $T = 1$, using processors $P_1, \ldots, P_k$ in that order, is*
$$\rho = \sum_{i=1}^{k} \alpha_i = \frac{\sum_{i=1}^{k} u_i}{1 + \sum_{i=1}^{k} u_i d_i},$$
where
$$u_i = \frac{1}{d_i + w_i} \prod_{j=1}^{i} \left(\frac{d_j + w_j}{c_j + w_j} \right).$$

Proof. Since we restrict our search to solutions such that $x = 0$, then with the notations of the proof of Lemma 1, we have
$$A\alpha = T\mathbb{1} = \mathbb{1}.$$
We have $\rho = \sum_{i=1}^{k} \alpha_i = \mathbb{1}^t A^{-1} \mathbb{1}$, We use the decomposition $A = L + \mathbb{1}d^t$ and the Sherman-Morrison formula, as in the proof of Lemma 1, to derive that
$$\rho = \mathbb{1}^t L^{-1} \mathbb{1} - \frac{(\mathbb{1}^t L^{-1} \mathbb{1})(d^t L^{-1} \mathbb{1})}{1 + d^t L^{-1} \mathbb{1}} = \frac{\mathbb{1}^t L^{-1} \mathbb{1}}{1 + d^t L^{-1} \mathbb{1}}.$$
In order to obtain an analytical expression for $\sum \alpha_i$, we need the following Lemma:

Lemma 3. *Let $u = L^{-1}\mathbb{1}$. Then,*

$$u_i = \frac{1}{d_i + w_i} \prod_{j=1}^{i} \left(\frac{d_j + w_j}{c_j + w_j} \right)$$

Proof. We use induction to prove that $\forall i, \quad e_i^t L u = 1$, where e_i is the i-th canonical vector:

- Clearly, $e_1^t L u = \frac{c_1+w_1}{d_i+w_i} \quad \frac{d_1+w_1}{c_1+w_1} = 1$ so that the lemma holds true for $i = 1$
- Let us assume now that the lemma holds true for $i-1$, i.e.

$$\sum_{l=1}^{i-2} \frac{c_l - d_l}{d_l + w_l} \prod_{j=1}^{l} \left(\frac{d_j + w_j}{c_j + w_j} \right) + \frac{c_{i-1} + w_{i-1}}{d_{i-1} + w_{i-1}} \prod_{j=1}^{i-1} \left(\frac{d_j + w_j}{c_j + w_j} \right) = 1.$$

Then,

$$\begin{aligned}
&\textstyle\sum_{l=1}^{i-1} \frac{c_l-d_l}{d_l+w_l} \prod_{j=1}^{l} \left(\frac{d_j+w_j}{c_j+w_j} \right) + \frac{c_i+w_i}{d_i+w_i} \prod_{j=1}^{i} \left(\frac{d_j+w_j}{c_j+w_j} \right) \\
&= \textstyle\sum_{l=1}^{i-2} \frac{c_l-d_l}{d_l+w_l} \prod_{j=1}^{l} \left(\frac{d_j+w_j}{c_j+w_j} \right) + \frac{c_{i-1}-d_{i-1}}{d_{i-1}+w_{i-1}} \prod_{j=1}^{i-1} \left(\frac{d_j+w_j}{c_j+w_j} \right) + \frac{c_i+w_i}{d_i+w_i} \prod_{j=1}^{i} \left(\frac{d_j+w_j}{c_j+w_j} \right) \\
&= 1 - \textstyle\frac{c_{i-1}+w_{i-1}}{d_{i-1}+w_{i-1}} \prod_{j=1}^{i-1} \left(\frac{d_j+w_j}{c_j+w_j} \right) + \frac{c_{i-1}-d_{i-1}}{d_{i-1}+w_{i-1}} \prod_{j=1}^{i-1} \left(\frac{d_j+w_j}{c_j+w_j} \right) + \frac{c_i+w_i}{d_i+w_i} \prod_{j=1}^{i} \left(\frac{d_j+w_j}{c_j+w_j} \right) \\
&= 1 + \textstyle\prod_{j=1}^{i-1} \left(\frac{d_j+w_j}{c_j+w_j} \right) \left(1 + \frac{c_{i-1}-d_{i-1}}{d_{i-1}+w_{i-1}} - \frac{c_{i-1}+w_{i-1}}{d_{i-1}+w_{i-1}} \right) \\
&= 1
\end{aligned}$$

which achieves the proof Lemma 3. □

Moreover, since

$$\sum \alpha_i = \frac{\mathbb{1}^t L^{-1} \mathbb{1}}{1 + d^t L^{-1} \mathbb{1}}$$

we obtain

$$\sum \alpha_i = \frac{\sum_{i=1}^{k} u_i}{1 + \sum_{i=1}^{k} u_i d_i},$$

which achieves the proof of Lemma 2. □

Lemma 2 provides an analytical expression for the throughput achieved using a given FIFO ordering and a given set of participating processors. The following lemma provides the optimal ordering for a given, fixed, set of participating processors (remember that we assume that $z = \frac{d_i}{c_i} \leq 1$):

Lemma 4. *Let $P_1, \ldots, P_k$ be the set of participating processors. We assume that $c_1 \leq c_2 \leq \ldots \leq c_k$. Then, the throughput is optimal if and only if initial messages are successively sent to $P_1, P_2, \ldots, P_k$.*

Proof. Let ρ^{opt} denote the optimal throughput (over the set of all possible communication orderings) that can be achieved using the set of processors $P_1, P_2, \ldots, P_k$, and let σ^{opt} be the corresponding optimal permutation. Then, using Lemma 2,

$$\rho^{\text{opt}} = \sum \alpha_i = \frac{\sum_{i=1}^{k} u_i}{1 + \sum_{i=1}^{k} u_i d_{\sigma_i^{\text{opt}}}},$$

where

$$u_i = \frac{1}{d_{\sigma_i^{\text{opt}}} + w_{\sigma_i^{\text{opt}}}} \prod_{j=1}^{i} \left(\frac{d_{\sigma_j^{\text{opt}}} + w_{\sigma_j^{\text{opt}}}}{c_{\sigma_j^{\text{opt}}} + w_{\sigma_j^{\text{opt}}}} \right).$$

Assume by contradiction that there exists an index m such that $c_{\sigma_m^{\text{opt}}} > c_{\sigma_{m+1}^{\text{opt}}}$ and let us denote by σ the permutation defined by $\sigma_m = \sigma_{m+1}^{\text{opt}}$, $\sigma_{m+1} = \sigma_m^{\text{opt}}$ and $\sigma_i = \sigma_i^{\text{opt}}$ otherwise. Let us also denote by α'_i and u'_i the values of α and u with the permutation σ.

Since

$$u'_i = \frac{1}{d_{\sigma_i} + w_{\sigma_i}} \prod_{j=1}^{i} \left(\frac{d_{\sigma_j} + w_{\sigma_j}}{c_{\sigma_j} + w_{\sigma_j}} \right),$$

we have that $\forall i \neq m, m+1, \quad u'_i = u_i$. Moreover, define

$$\begin{aligned} K &= \prod_{j=1}^{m-1} \left(\frac{d_{\sigma_j} + w_{\sigma_j}}{c_{\sigma_j} + w_{\sigma_j}} \right) \\ &= \prod_{j=1}^{m-1} \left(\frac{d_{\sigma_j^{\text{opt}}} + w_{\sigma_j^{\text{opt}}}}{c_{\sigma_j^{\text{opt}}} + w_{\sigma_j^{\text{opt}}}} \right). \end{aligned}$$

Then

$$u'_m = K \frac{1}{c_{\sigma_m} + w_{\sigma_m}} = K \frac{1}{c_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}}},$$

$$u'_{m+1} = K \frac{d_{\sigma_m} + w_{\sigma_m}}{(c_{\sigma_m} + w_{\sigma_m})(c_{\sigma_{m+1}} + w_{\sigma_{m+1}})} = K \frac{d_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}}}{(c_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}})(c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}})},$$

$$u_m = K \frac{1}{c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}}},$$

and

$$u_{m+1} = K \frac{d_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}}}{(c_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}})(c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}})}.$$

Let us now evaluate

$$\Delta = u'_m(1 - \rho^{\text{opt}} d_{\sigma_m}) + u'_{m+1}(1 - \rho^{\text{opt}} d_{\sigma_{m+1}}) - u_m(1 - \rho^{\text{opt}} d_{\sigma_m^{\text{opt}}}) - u_{m+1}(1 - \rho^{\text{opt}} d_{\sigma_{m+1}^{\text{opt}}}).$$

$$\begin{aligned} \Delta &= u'_m(1 - \rho^{\text{opt}} d_{\sigma_{m+1}^{\text{opt}}}) + u'_{m+1}(1 - \rho^{\text{opt}} d_{\sigma_m^{\text{opt}}}) - u_m(1 - \rho^{\text{opt}} d_{\sigma_m^{\text{opt}}}) - u_{m+1}(1 - \rho^{\text{opt}} d_{\sigma_{m+1}^{\text{opt}}}) \\ &= K(1 - \rho^{\text{opt}} d_{\sigma_{m+1}^{\text{opt}}}) \left(\frac{1}{d_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}}} - \frac{d_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}}}{(c_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}})(c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}})} \right) + \\ &\quad K(1 - \rho^{\text{opt}} d_{\sigma_m^{\text{opt}}}) \left(\frac{d_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}}}{(c_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}})(c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}})} - \frac{1}{c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}}} \right) \\ &= K(1 - \rho^{\text{opt}} d_{\sigma_{m+1}^{\text{opt}}}) \left(\frac{c_{\sigma_m^{\text{opt}}} - d_{\sigma_m^{\text{opt}}}}{(c_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}})(c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}})} \right) + K(1 - \rho^{\text{opt}} d_{\sigma_m^{\text{opt}}}) \left(\frac{d_{\sigma_{m+1}^{\text{opt}}} - c_{\sigma_{m+1}^{\text{opt}}}}{(c_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}})(c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}})} \right) \\ &= \frac{K(1-z)}{(c_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}})(c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}})} \left((1 - \rho^{\text{opt}} z c_{\sigma_{m+1}^{\text{opt}}}) c_{\sigma_m^{\text{opt}}} - (1 - \rho^{\text{opt}} z c_{\sigma_m^{\text{opt}}}) c_{\sigma_{m+1}^{\text{opt}}} \right) \\ &= \frac{K(1-z)}{(c_{\sigma_{m+1}^{\text{opt}}} + w_{\sigma_{m+1}^{\text{opt}}})(c_{\sigma_m^{\text{opt}}} + w_{\sigma_m^{\text{opt}}})} \left(c_{\sigma_m^{\text{opt}}} - c_{\sigma_{m+1}^{\text{opt}}} \right) \\ &> 0 \text{ by assumption} \end{aligned}$$

Therefore,

$$\begin{aligned} \textstyle\sum_i u'_i(1 - \rho^{\text{opt}} d_{\sigma_i}) &= \textstyle\sum_{i \neq m, m+1} u_i(1 - \rho^{\text{opt}} d_{\sigma_i^{\text{opt}}}) + \Delta + u_m(1 - \rho^{\text{opt}} d_{\sigma_m^{\text{opt}}}) + u_{m+1}(1 - \rho^{\text{opt}} d_{\sigma_{m+1}^{\text{opt}}}) \\ &= \textstyle\sum_i u_i(1 - \rho^{\text{opt}} d_{\sigma_i^{\text{opt}}}) + \Delta \\ &> \textstyle\sum_i u_i(1 - \rho^{\text{opt}} d_{\sigma_i^{\text{opt}}}) \\ &> \rho^{\text{opt}} \end{aligned}$$

Thus, $\sum_i u'_i > \rho^{\text{opt}}(1 + \sum_i u'_i d_{\sigma_i})$ and finally

$$\frac{\sum_i u'_i}{1 + \sum_i u'_i d_{\sigma_i}} > \rho^{\text{opt}}.$$

We have proved that if there exists an index m such that $c_{\sigma_m^{\text{opt}}} > c_{\sigma_{m+1}^{\text{opt}}}$, then it is possible, by switching those processors, to obtain a solution that processes strictly more load units. An easy induction shows that initial messages must be sent successively to processors $P_1, \ldots, P_k$ if $c_1 \leq \ldots \leq c_k$. □

Lemma 4 provides the communication ordering, once the set of participating processors is fixed. We still need to provide an algorithm for determining the optimal set of participating processors. This is the objective of the next two lemmas:

Lemma 5. *Let ρ^{opt} denote the optimal throughput that can be achieved. Then, only those workers P_i such that*

$$d_i \leq \frac{1}{\rho^{opt}}$$

should participate to the execution.

Proof. Suppose that the set of participating processors is $P_1, \ldots, P_k$ and that there exists a processor such that $(1 - \rho^{\text{opt}} d_i) < 0$. Since $c_i \leq c_k$, then $d_i \leq d_k$ and therefore

$$(1 - \rho^{\text{opt}} d_k) < 0.$$

In what follows, we prove that the throughput achieved without using P_k is larger than the throughput with P_k. Let us denote by u'_i the values of the u's when P_k does not participate to the execution. Since P_k is the last processor, given the definition of u, we have $\forall i < k, \quad u'_i = u_i$. Therefore,

$$\begin{array}{rcl} \sum_{i=1}^{k-1} u'_i(1 - \rho^{\text{opt}} d_i) & = & \sum_{i=1}^{k-1} u_i(1 - \rho^{\text{opt}} d_i) \\ & = & \sum_{i=1}^{k} u_i(1 - \rho^{\text{opt}} d_i) - u_k(1 - \rho^{\text{opt}} d_k) \\ & > & \sum_{i=1}^{k} u_i(1 - \rho^{\text{opt}} d_i) \text{ since } u_k > 0 \text{ and } (1 - \rho^{\text{opt}} d_k) < 0 \\ & > & \rho^{\text{opt}}. \end{array}$$

Therefore, $\sum_{i=1}^{k-1} u'_i(1 - \rho^{\text{opt}} d_i) > \rho^{\text{opt}}$ and finally

$$\frac{\sum_{i=1}^{k-1} u'_i}{1 + \sum_{i=1}^{k-1} u'_i d_i} > \rho^{\text{opt}}.$$

Therefore, we have proved that, the optimal throughput ρ^{opt} being known, if there exists a processor P_i such that $(1 - \rho^{\text{opt}} d_i) < 0$, then it cannot contribute to process any load, which achieves the proof of the lemma. □

Lemma 6. *Let ρ^{opt} denote the optimal throughput that can be achieved. Then, all those workers P_i such that*

$$d_i \leq \frac{1}{\rho^{opt}}$$

should participate to the execution.

Proof. Consider the set of processors $\mathcal{S}^{\text{opt}}$ that achieves the optimal throughput ρ^{opt}. To simplify notations, let $\mathcal{S}^{\text{opt}} = \{P_1, P_2, \ldots, P_r\}$. According to Lemma 5, we have $(1 - \rho^{\text{opt}} d_i) \geq 0$ for $P_i \in \mathcal{S}^{\text{opt}}$. Assume (by contradiction) that there exists another worker, say P_m, which does not belong to $\mathcal{S}^{\text{opt}}$ but which satisfies to $(1 - \rho^{\text{opt}} d_m) \geq 0$. Then we claim that adding P_m at the end of $\mathcal{S}^{\text{opt}}$ improves the throughput.

Indeed, we have $\rho^{\text{opt}} = \frac{U}{1+D}$, with $U = \sum_{i=1}^{r} u_i$ and $D = \sum_{i=1}^{r} u_i d_i$, and where the u_i's are given by Lemma 3. Because P_m is added at the end of the schedule, it does not change the value of the previous u_i. Adding P_m leads to the new throughput $\rho' = \frac{U+u_m}{1+D+u_m d_m}$. We have

$$\rho' - \rho^{\text{opt}} = \frac{u_m}{1 + D + u_m d_m}(1 - \rho^{\text{opt}} d_m) \geq 0,$$

hence the result. □

We are almost done! If we sort the communication parameters so that $d_1 \leq d_2 \leq \ldots \leq d_p$ (which is equivalent to sorting the c_i's or the $c_i + d_i$'s because $d_i = zc_i$ for all i), we know that the optimal throughput ρ^{opt} is achieved when we use *only* those workers P_i such that $d_i \leq \frac{1}{\rho^{\text{opt}}}$, and *all* of them, in the natural ordering. In other words, the optimal solution is obtained when using the first q processors in their natural order, for some value of q between 1 and p. The simplest algorithm to determine q is to try all the p possible values. It is possible to compute all the p corresponding throughputs in polynomial time, and to keep the best of them. Even better, owing to the formula given in Lemma 2, we can determine the optimal FIFO throughput in linear time, provided that we take care to update values on the fly in constant time when moving from q to $q+1$ workers. Indeed,

$$\rho_q = \sum_{i=1}^{q} \alpha_i = \frac{\sum_{i=1}^{q} u_i}{1 + \sum_{i=1}^{q} u_i d_i}$$

and

$$u_i = \frac{1}{d_i + w_i} \prod_{j=1}^{i} \left(\frac{d_j + w_j}{c_j + w_j} \right).$$

Therefore, we can determine ρ_{q+1} from ρ_q in constant time using the following equations:

$$u_{q+1} = \left(\frac{d_q + w_q}{c_q + w_q} \right) u_q, \quad N_{q+1} = N_q + u_{q+1}, \quad D_{q+1} = D_q + d_{q+1} u_{q+1}$$

and finally

$$\rho_{q+1} = \frac{N_{q+1}}{1 + D_{q+1}}.$$

This concludes the proof of Theorem 2. For the sake of completeness, we state the counterpart of Proposition 1:

Proposition 2. *The optimal FIFO strategy (and the corresponding throughput) can be determined in linear time $O(p)$.*

To conclude the study of FIFO solutions, we must deal with the case $z > 1$, where $d_i = zc_i$. Assume that we have a FIFO solution on a platform as in Figure 6, whose parameters are w_i, c_i, d_i and with $z > 1$. If we read the figure upside down, with time flying backwards, we have a FIFO solution for the platform whose parameters are w_i, d_i, c_i. This simple observation leads to the optimal FIFO solution when $z > 1$: solve the problem with w_i, d_i, c_i as explained in this section (because $c_i = \frac{1}{z} d_i$ with $\frac{1}{z} \leq 1$, Theorem 2 is applicable) and flip over the solution. Note that this implies that initial messages are sent in non-increasing order of the c_i's, rather than in non-decreasing order as was the case for $z < 1$. In passing, this also shows that when $z = 1$, i.e. $c_i = d_i$, the ordering of participating workers has no importance (but some workers may not be enrolled). This last statement can be checked directly with Lemma 2: if $z = 1$, then $u_i = \frac{1}{d_i + w_i}$, and these values are indeed independent of the ordering.

5 Simulations

In this section, we present the results of some simulations conducted with the LIFO and FIFO strategies. We cannot compare these results against the optimal schedule, since we are not able to determine the optimal solution as soon as the number of workers exceeds a few units. For instance, for a platform with 100 workers, we would need to solve $(100!)^2$ linear programs of 100 unknowns (one program for each permutation pair). Rather than computing the solution for all permutation pairs, we use the optimal FIFO algorithm as a basis for the comparisons.

The algorithms tested in this section are the following:

- optimal FIFO solution, as determined in Section 4, called OPT-FIFO

- optimal LIFO solution , as determined in Section 3, called OPT-LIFO
- a FIFO heuristic using all processors, sorted by non-decreasing values of c_i (faster communicating workers first), called FIFO-INC-C
- a FIFO heuristic using all processors, sorted by non-decreasing values of w_i (faster computing workers first, called FIFO-INC-W).

In the following, we present the relative performance of these heuristics on a master/worker platform with 100 workers. For these experiments, we chose $z = 0.8$, meaning that the returned data represents 80% of the input data. The performance parameters (communication and computation costs) of each worker may vary from 50% around an average value. The ratio of the average computation cost over the average communication cost is used to distinguish between the experiments, as the behavior of the heuristics highly depends on this parameter. This ratio is called the *w/c-ratio* in the following.

Figure 7 presents the throughput of the different heuristics for a w/c-ratio going from 1/10 to 100. These results are normalized so that the optimal FIFO algorithm always gets a throughput of 1. We see that both OPT-FIFO and FIFO-INC-C give good results. The other heuristics (FIFO-INC-W and OPT-LIFO) perform not so well, except when the w/c-ratio is high: in this case, communications have no real impact on the schedule, and almost all schedules may achieve good performances.

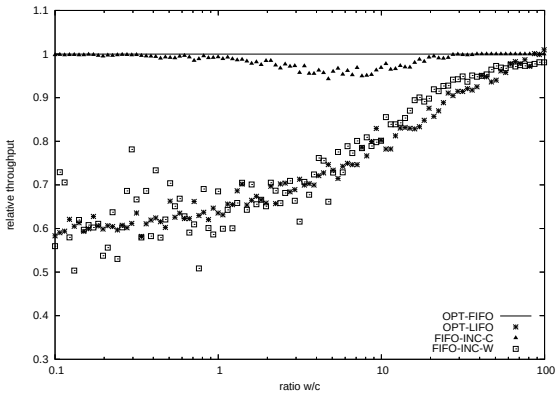


Figure 7: Performance of the heuristics relative to the optimal FIFO schedule, for different computation/communication costs ratios.

In Section 4, we showed that using all processors is not always a good choice. In Figure 8 we plot the number of processors used by the OPT-FIFO algorithm for the previous experiments: for small values of the w/c-ratio, a very small fraction of the workers is enrolled in the optimal schedule.

Finally, Figure 9 presents the relative performance of all heuristics when the size of the data returned is the same as the size of the input data ($z = 1$). With the exception of this new hypothesis ($z = 1$ instead of $z = 0.8$), the experimental settings are the same as for Figure 7. We show that for a w/c-ratio less than 10, only the OPT-FIFO algorithm gives good performance: the FIFO-INC-C heuristic is no longer able to reach a comparable throughput. We also observe what we have proved at the end of Section 4: when $z = 1$ the ordering of the workers has no importance, FIFO-INC-C and FIFO-INC-W are FIFO strategies involving all the workers but in different orders, and give exactly the same results.

6 Related work

In addition to the landmark book [7] quoted in Section 1, several sources to DLS literature are available: see the two introductory surveys [8, 21], the special issue of the Cluster Computing

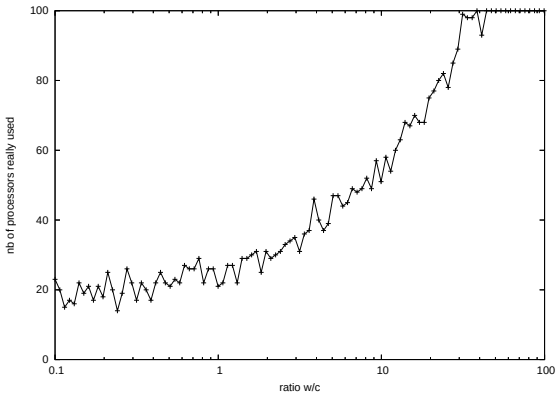


Figure 8: Number of workers enrolled in the optimal FIFO schedule, for different computation/communication cost ratios.

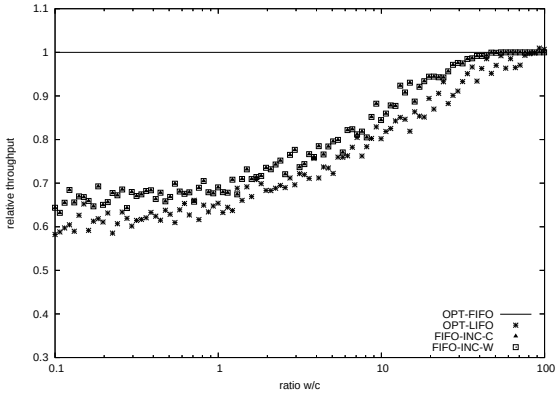


Figure 9: Performance of the heuristics relative to the optimal FIFO schedule, when $c_i = d_i$ ($z = 1$).

journal entirely devoted to divisible load scheduling [15], and the Web page collecting DLS-related papers is maintained [20].

DLS applications include linear algebra [10], image processing [17, 19], video and multimedia broadcasting [2, 3], database searching [12, 9], and the processing of large distributed files [25]. These applications are amenable to the simple master-worker programming model and can thus be easily implemented and deployed on computing platforms ranging from small commodity clusters to computational grids [14].

The DLS model comes in two flavors, with a linear cost model and with an affine cost model. The linear cost model is the original model, and has been widely adopted because of its simplicity: several closed-form formulas are available for star, tree, mesh networks among others [7, 20]. The affine cost model (which amounts to introduce a start-up overhead in the communication cost, and/or in the computation cost) has been advocated more recently, for two main reasons: (i) it is more realistic than the linear model; and (ii) it cannot be avoided when dealing with multiple-round scenarios, where the master is allowed to send data to the workers with several messages rather than with a single one. Multiple-round strategies are better amenable to pipelining than one-round approaches, but using a linear cost model would then favor sending a large collection of infinitely small messages, hence the need to add communication latencies. However, latencies render the problem more complex: the DLS problem has recently been proved NP-hard on a star network with the affine model [18].

When dealing with one-round scenarios, as in this paper, the linear model is more realistic, especially is the total work to be distributed to the slaves is large. From a theoretical perspective, one major question was to determine whether adding return messages, while retaining the linear model, would keep the DLS scheduling problem polynomially tractable. We failed to answer this question, but we have been able to characterize optimal solutions for LIFO and FIFO strategies.

Relatively few papers have considered adding return messages in the study of DLS problems. Pioneering results are reported by Barlas [4], who tackles the same problem as in this paper (one round, star platform) but with an affine framework model. Barlas [4] concentrates on two particular cases: one called *query processing*, where communication time (both for initial and return messages) is a constant independent of the message size, and the other called *image processing*, which reduces to linear communication times on a bus network, but with affine computation times. In both cases, the optimal sequence of messages is given, and a closed-form solution to the DLS problem is derived. In [13], the authors consider experimental validation of the DLS model for several applications (pattern searching, graph coloring, compression and join operations in databases). They consider both FIFO and LIFO distributions, but they do not discuss communication ordering.

Rosenberg [22] and Adler, Gong and Rosenberg [1] also tackle the DLS model with return messages, but they limit themselves to a bus network (same link bandwidth for all workers). They introduce a very detailed communication model, but they state results for affine communication costs and linear computation costs. They have the additional hypothesis that worker processors can be *slowed down* instead of working at full speed, with allows them not to consider no idle times between the end of the execution and the emission of the return messages. They state the very interesting result that all FIFO strategies are equivalent, and that they perform better than any other protocol. Note that our results, although not derived under the same model, are in accordance with these results: when the star platform reduces to a bus platform, the results of Section 4 show that all processors should be involved in the computation, and that their ordering has no impact on the quality of the solution.

Finally, we point out that Altilar and Paker [3] also investigate the DLS problem on a star network, but their paper is devoted to the asymptotic study of several multi-round strategies.

7 Conclusion

In this paper we have dealt with divisible load scheduling on a heterogeneous master-worker platform. We have shown that including return messages from the workers after execution, although

a very natural and important extension in practice, leads to considerable difficulties. These difficulties were largely unexpected, because of the simplicity of the linear model.

We have not been able to fully assess the complexity of the problem, but we have succeeded in characterizing the optimal LIFO and FIFO strategies, and in providing an experimental comparison of these strategies against simpler greedy approaches.

Future work must be devoted to investigate the general case, i.e. using two arbitrary permutation orderings for sending messages from, and returning messages to, the master. This seems to be a very combinatorial and complicated optimization problem.

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] D. Altilar and Y. Paker. An optimal scheduling algorithm for parallel video processing. In *IEEE Int. Conference on Multimedia Computing and Systems*. IEEE Computer Society Press, 1998.

[3] D. Altilar and Y. Paker. Optimal scheduling algorithms for communication constrained parallel processing. In *Euro-Par 2002*, LNCS 2400, pages 197–206. Springer Verlag, 2002.

[4] 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.

[5] S. Bataineh, T. Hsiung, and T.G.Robertazzi. Closed form solutions for bus and tree networks of processors load sharing a divisible job. *IEEE Transactions on Computers*, 43(10):1184–1196, Oct. 1994.

[6] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Trans. Parallel Distributed Systems*, 16(3):207–218, 2005.

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

[8] V. Bharadwaj, D. Ghose, and T. Robertazzi. Divisible load theory: a new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17, 2003.

[9] J. Blazewicz, M. Drozdowski, and M. Markiewicz. Divisible task scheduling - concept and verification. *Parallel Computing*, 25:87–98, 1999.

[10] S. Chan, V. Bharadwaj, and D. Ghose. Large matrix-vector products on distributed bus networks with communication delays using the divisible load paradigm: performance and simulation. *Mathematics and Computers in Simulation*, 58:71–92, 2001.

[11] B. W. Char, K. O. Geddes, G. H. Gonnet, M. B. Monagan, and S. M. Watt. *Maple Reference Manual*, 1988.

[12] M. Drozdowski. *Selected problems of scheduling tasks in multiprocessor computing systems*. PhD thesis, Instytut Informatyki Politechnika Poznanska, Poznan, 1997.

[13] M. Drozdowski and P. Wolniewicz. Experiments with scheduling divisible tasks in clusters of workstations. In *Proceedings of Euro-Par 2000: Parallel Processing*, LNCS 1900, pages 311–319. Springer, 2000.

[14] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1999.

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

[16] G. H. Golub and C. F. V. Loan. *Matrix computations*. Johns Hopkins, 1989.

[17] C. Lee and M. Hamdi. Parallel image processing applications on a network of workstations. *Parallel Computing*, 21:137–160, 1995.

[18] A. Legrand, Y. Yang, and H. Casanova. Np-completeness of the divisible load scheduling problem on heterogeneous star platforms with affine costs. Research Report CS2005-0818, GRAIL Project, University of California at San Diego, march 2005.

[19] X. Li, V. Bharadwaj, and C. Ko. Distributed image processing on a network of workstations. *Int. J. Computers and Applications (ACTA Press)*, 25(2):1–10, 2003.

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

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

[22] A. L. Rosenberg. Sharing partitionable workloads in heterogeneous NOws: greedier is not better. In *Cluster Computing 2001*, pages 124–131. IEEE Computer Society Press, 2001.

[23] J. Sohn, T. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on parallel and distributed systems*, 9(3):225–234, Mar. 1998.

[24] The MuPAD Group (B. Fuchssteiner et al.). *MuPAD User's Manual*. John Wiley and sons, 1996.

[25] R. Wang, A. Krishnamurthy, R. Martin, T. Anderson, and D. Culler. Modeling communication pipeline latency. In *Measurement and Modeling of Computer Systems (SIGMETRICS'98)*, pages 22–32. ACM Press, 1998.