Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems

Olivier Beaumount,
Henri Casanova ,
Arnaud Legrand,
Yves Robert,
Yang Yang

September 2003

Research Report N^o 2003-41

Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems

Olivier Beaumount, Henri Casanova , Arnaud Legrand, Yves Robert, Yang Yang

September 2003

Abstract

Many applications in scientific and engineering domains are structured as large numbers of independent tasks with low granularity. These applications are thus amenable to straightforward parallelization, typically in master-worker fashion, provided that efficient scheduling strategies are available. Such applications have been called *divisible loads* because a scheduler may *divide* the computation among worker processes arbitrarily, both in terms of number of tasks and of task sizes. Divisible load scheduling has been an active area of research for the last twenty years. A vast literature offers results and scheduling algorithms for various models of the underlying distributed computing platform. Broad surveys are available that report on accomplishments in the field. By contrast, in this paper we propose a unified theoretical perspective that synthesizes previously published results, several novel results, and open questions, in a view to foster novel divisible load scheduling research. Specifically, we discuss both one-round and multi-round algorithms, and we restrict our scope to the popular star and tree network topologies, which we study with both linear and affine cost models for communication and computation.

Keywords: parallel computing, scheduling, divisible load

Résumé

De nombreuses applications scientifiques se découpent naturellement en un grand nombre de tâches indépendantes avec une faible granularité. Ces applications se parallélisent naturellement à l'aide d'une approche maître/esclave. De telles applications relèvent du modèle des *tâches divisibles* car un ordonnanceur peut *diviser* les calculs sur les différents processeurs disponibles, à la fois en terme de nombre de tâches mais également en terme de taille des tâches. L'ordonnancement de tâches divisibles a été un domaine de recherche actif durant les vingts dernières années. On trouve donc dans la littérature de nombreux résultats et algorithmes d'ordonnancement pour différents modèles de plates-formes. À la différence des états de l'art déja existant sur le sujet, ce rapport propose une nouvelle approche permettant d'unifier et de retrouver les résultats de la littérature, de proposer de nouveaux résultats et d'ouvrir de nouveaux problèmes. Plus précisément, nous présentons les distributions en une seule tournée et en plusieurs tournées et nous restreignons aux topologies populaires en étoile et en arborescence, que nous nous étudions à l'aide de coût de calculs et de communications linéaires puis affines.

Mots-clés: calcul parallèle, ordonnancement, tâches divisibles

1 Introduction

Scheduling the tasks of a parallel application on the resources of a distributed computing platform efficiently is critical for achieving high performance. The scheduling problem has been studied for a variety of application models, such as the well-known directed acyclic task graph model for which many scheduling heuristics have been developed [39]. Another popular application model is that of independent tasks with no task synchronizations and no inter-task communications. Applications conforming to this admittedly simple model arise in most fields of science and engineering. A possible model for independent tasks is one for which the number of tasks and the task sizes, i.e. their computational costs, are set in advance. In this case, the scheduling problem is akin to bin-packing and a number of heuristics have been proposed in the literature (see [18, 30] for surveys). Another flavor of the independent tasks model is one in which the number of tasks and the task sizes can be chosen arbitrarily. This corresponds to the case when the application 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. In practice, this model is an approximation of an application that consists of large numbers of identical, low-granularity computations. This *divisible load* model has been widely studied in the last several years, and *Divisible Load Theory* (DLT) has been popularized by the landmark book written in 1996 by Bharadwaj, Ghose, Mani and Robertazzi [10].

DLT provides a practical framework for the mapping on independent tasks onto heterogeneous platforms, and has been applied to a large spectrum of scientific problems, including Kalman filtering [40], image processing [32], video and multimedia broadcasting [1, 2], database searching [19, 13], and the processing of large distributed files [41]. 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 [24]. From a theoretical standpoint, the success of the divisible load model is mostly due to its analytical tractability. Optimal algorithms and closed-form formulas exist for the simplest instances of the divisible load problem. This is in sharp contrast with the theory of task graph scheduling, which abounds in NP-completeness theorems [25, 23] and in inapproximability results [18, 3].

There exists a vast literature on DLT. In addition to the landmark book [10], two introductory surveys have been published recently [11, 37]. Furthermore, a special issue of the Cluster Computing journal is entirely devoted to divisible load scheduling [26], and a Web page collecting DLT-related papers is maintained [36]. Consequently, the goal of this paper is not to present yet another survey of DLT theory and its various applications. Instead, we focus on relevant theoretical aspects: we aim at synthesizing some important results for realistic platform models. We give a new presentation of several previously published results, and we add a number of new contributions. The material in this paper provides the level of detail and, more importantly, the unifying perspective that are necessary for fostering new research in the field.

We limit our discussion star-shaped and tree-shaped logical network topologies, because they often represent the solution of choice to implement master-worker computations. Note that the star network encompasses the case of a bus, which is a homogeneous star network. The extended version of this paper [6] reviews works that study other network topologies. We consider two types of model for communication and computation: linear or affine in the data size. In most contexts, this is more accurate than the fixed cost model, which assumes that

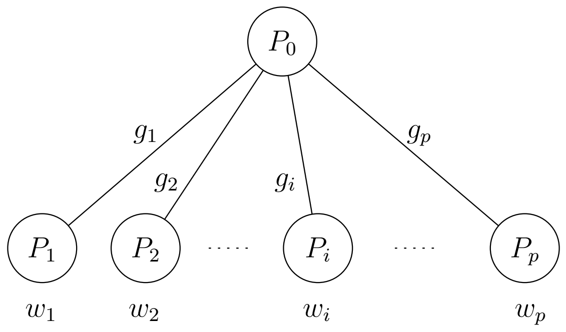

Figure 1: Heterogeneous star graph, with the linear cost model.

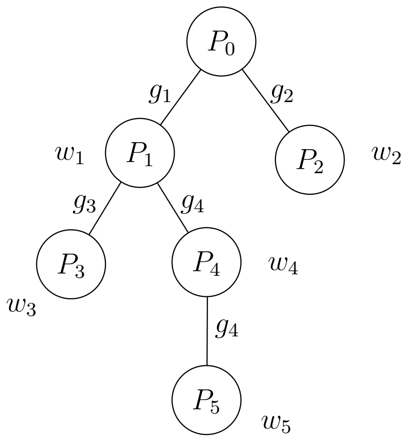

Figure 2: Heterogeneous tree graph.

the time to communicate a message is independent of the message size. Works considering fixed cost models are reviewed in [6].

The rest of this paper is organized as follows. In Section 2, we detail our platform and cost models. We also introduce the algorithmic techniques that have been proposed to schedule divisible loads: one-round and multi-round algorithms. One-round algorithms are described in detail in Section 3 and multi-round algorithms in Section 4. Finally, we conclude in Section 6.

2 Framework

2.1 Target architectures and cost models

We consider either star-graphs or tree-graphs, and either linear or affine costs, which leads to four different platform combinations.

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_q. In the linear cost model, each worker P_q has a (relative) computing power w_q: it takes $X.w_q$ time units to execute X units of load on worker P_q. Similarly, it takes $X.g_q$ time unites to send X units of load from P_0 to P_q. 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).

In the affine cost model, a latency is added to computation and communication costs: it takes $W_q + X.w_q$ time units to execute X units of load on worker P_q, and $G_q + X.g_q$ time units to send X units of load from P_0 to P_q. It is acknowledged that these latencies make the model more realistic.

For communications, the one-port model is used: the master can only communicate with a single worker at a given time-step. We assume that communications can overlap computations on the workers: a worker can compute a load fraction while receiving the data necessary for the execution of the next load fraction. This corresponds to workers *equipped with a front end* as in [10]. A *bus network* is a star network such that all communication links have the same characteristics: $g_i = g$ and $G_i = G$ for each worker P_i, $1 \leq i \leq p$.

Essentially, the same one-port model, with overlap of communication with computation, is used for tree-graph networks. A tree-graph $\mathcal{T} = \{P_0, P_1, P_2, \dots, P_p\}$ (see Figure 2) simply is an arborescence rooted at the master P_0. We still call the other resources *workers*, even though non-leaf workers have other workers (their children in the tree) to which they can delegate work. In this model, it is assumed that a worker in the tree can simultaneously perform some computation, receive data from its parent, and communicate to at most one of its children (sending previously received data).

2.2 Algorithmic strategies: one-round versus multi-round

We denote by W_{total} the total load to be executed. The key assumption of DLT is that this load is perfectly divisible into an arbitrary number of pieces, or *chunks*. The master can distribute the chunks to the workers in a single *round* (also called "installment" in [10]), so that there is a single communication between the master and each worker. The problem is to determine the size of these chunks and the order in which they are sent to the workers. We review one-round algorithms in Section 3. For large loads, the single round approach is not efficient due to the idle time incurred by the last workers to receive chunks. To reduce the makespan, i.e. the total execution time, the master can send chunks to the workers in multiple rounds so that communication is pipelined and overlapped with computation. Additional questions in this case are: "How many rounds should be scheduled?"; and "What are the best chunk sizes at each round?" We discuss multi-round algorithms in Section 4.

3 One-round algorithms

For one-round algorithms, the first problem is to determine in which order the chunks should be sent to the different workers (or equivalently to sort the 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.

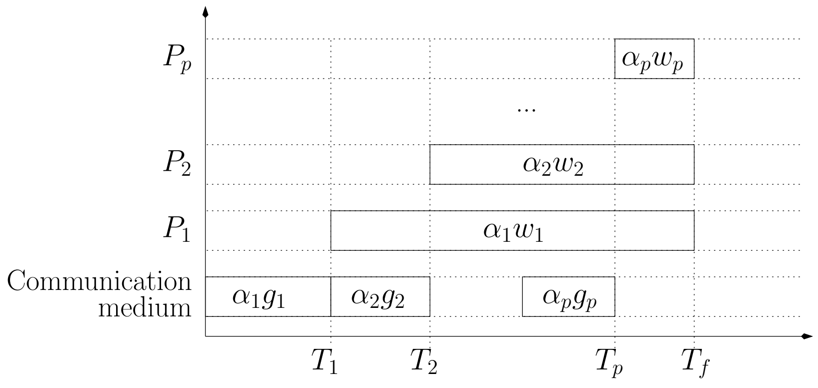

Figure 3: Pattern of a solution for dispatching a divisible load, using a star network and the linear cost model. All workers complete execution at the same time-step T_f.

3.1 Star network and linear cost model

This is the simplest platform combination, denoted as StarLinear. Let α_i denote the number of units of load sent to worker P_i, such that $\sum_{i=1}^{p} \alpha_i = W_{\text{total}}$. Figure 3 depicts the execution, where T_i denotes idle time of P_i, i.e. the time elapsed before P_i begins its processing. The goal is to minimize the total execution time, $T_f = \max_{1 \leq i \leq p}(T_i + \alpha_i w_i)$, according to the linear model defined in Section 2. In Figure 3, all the workers participate in the computation, and they all finish computing at the same time (i.e. $T_i + \alpha_i w_i = T_f,\ \forall i$). This is a general result:

Proposition 1. *In any optimal solution of the* StarLinear *problem, all workers participate in the computation, and they all finish computing simultaneously.*

Note that Proposition 1 has been proved for the case of a bus in [10]. To the best of our knowledge, this is a new result for the case of a heterogeneous star network.

Proof. We first prove that in an optimal solution all workers participate to the computation. Then, we prove that in any optimal solution, all workers finish computing simultaneously.

Lemma 1. *In any optimal solution, all workers participate in the computation.*

Proof. Suppose that there exists an optimal solution where at least one worker is kept fully idle. In this case, at least one of the α_i, $1 \leq i \leq P$, is zero. Let us denote by k the largest index such that $\alpha_k = 0$.

Case $k < n$. Consider a solution of StarLinear, where the ordering

$$P_1, \ldots, P_{k-1}, P_{k+1}, \ldots, P_n, P_k$$

is used. This solution is clearly optimal since P_k did not process any load in the initial solution. By construction, $\alpha_n \neq 0$, so that the communication medium is not used during at least the last $\alpha_n w_n$ time units. Therefore, it would be possible to process at least $\frac{\alpha_n w_n}{g_k + w_k} > 0$ additional units of load with worker P_k, which contradicts the assumption that the original solution was optimal.

Case $k = n$. Consider the original solution of StarLinear, i.e. with the ordering $P_1, \ldots, P_n$.Moreover, let k' be the largest index such that $\alpha_{k'} \neq 0$. By construction, the communication medium is not used during at least the last $\alpha_{k'} w_{k'} > 0$ time units. Thus, as previously, it would be possible to process at least $\frac{\alpha_{k'} w_{k'}}{g_n + w_n} > 0$ additional units of load with worker P_n, which leads to a similar contradiction.

Therefore, in any optimal solution, all workers participate in the computation. □

It is worth pointing out that the above property does not hold true if we consider solutions in which the communication ordering is fixed *a priori*. For instance, consider a platform comprising two workers: P_1 (with $g_1 = 4$ and $w_1 = 1$) and P_2 (with $g_2 = 1$ and $w_2 = 1$). If the first chunk has to be sent to P_1 and the second chunk to P_2, the optimal number of units of load that can be processed within 10 time units is 5, and P_1 is kept fully idle in this solution. On the other hand, if the communication ordering is not fixed, then 6 units of load can be performed within 10 time units (5 units of load are sent to P_2, and then 1 to P_1). In the optimal solution, both workers perform some computation, and both workers finish computing at the same time, which is stated in the following lemma.

Lemma 2. *In the optimal schedule, all workers finish computing simultaneously.*

Proof. Consider an optimal solution. All the α_i's have strictly positive values (Lemma 1). Consider the following linear program:

$$\begin{array}{l} \text{MAXIMIZE } \sum \beta_i, \\ \text{SUBJECT TO} \\ \left\{ \begin{array}{lll} \text{LB}(i) & \forall i, & \beta_i \geq 0 \\ \text{UB}(i) & \forall i, & \sum_{k=1}^{i} \beta_k g_k + \beta_i w_i \leq T \end{array} \right. \end{array}$$

The α_i's satisfy the set of constraints above, and from any set of β_i's satisfying the set of inequalities, we can build a valid solution of the StarLinear problem that process exactly $\sum \beta_i$ units of load. Therefore, if we denote by $(\beta_1, \ldots, \beta_n)$ an optimal solution of the linear program, then $\sum \beta_i = \sum \alpha_i$.

It is known that one of the extremal solutions $\mathcal{S}_1$ of the linear program is one of the convex polyhedron $\mathcal{P}$ induced by the inequalities [38, chapter 11]: this means that in the solution $\mathcal{S}_1$, at least n inequalities among the $2n$ are equalities. Since we know that for any optimal solution of the StarLinear problem, all the β_i's are strictly positive (Lemma 1), then this vertex is the solution of the following (full rank) linear system

$$\forall i, \quad \sum_{k=1}^{i} \beta_k g_k + \beta_i w_i = T.$$

Thus, we derive that there is an optimal solution where all workers finish their work at the same time.

Let us denote by $\mathcal{S}_2 = (\alpha_1, \ldots, \alpha_n)$ another optimal solution, with $\mathcal{S}_1 \neq \mathcal{S}_2$. As already pointed out, $\mathcal{S}_2$ belongs to the polyhedron $\mathcal{P}$. Now, consider the following function f:

$$f : \left\{ \begin{array}{lll} \mathbb{R} & \rightarrow & \mathbb{R}^n \\ x & \mapsto & \mathcal{S}_1 + x(\mathcal{S}_2 - \mathcal{S}_1) \end{array} \right.$$

By construction, we know that $\sum \beta_i = \sum \alpha_i$. Thus, with the notation $f(x) = (\gamma_1(x), \ldots, \gamma_n(x))$:

$$\forall i, \gamma_i(x) = \beta_i + x(\alpha_i - \beta_i),$$

and therefore

$$\forall x, \quad \sum \gamma_i(x) = \sum \beta_i = \sum \alpha_i.$$

Therefore, all the points $f(x)$ that belong to $\mathcal{P}$ are extremal solutions of the linear program.

Since $\mathcal{P}$ is a convex polyhedron and both $\mathcal{S}_1$ and $\mathcal{S}_2$ belong to $\mathcal{P}$, then $\forall 0 \leq x \leq 1, \quad f(x) \in \mathcal{P}$. Let us denote by x_0 the largest value of $x \geq 1$ such that $f(x)$ still belongs to $\mathcal{P}$: at least one constraint of the linear program is an equality in $f(x_0)$, and this constraint is not satisfied for $x > x_0$. Could this constraint be one of the UB(i)'s? the answer is no, because otherwise this constraint would be an equality along the whole line $(\mathcal{S}_2 f(x_0))$, and would remain an equality for $x > x_0$. Hence, the constraint of interest is one of the LB(i)'s. In other terms, there exists an index i such that $\gamma_i(x_0) = 0$. This is a contradiction since we have proved that the γ_i's correspond to an optimal solution of the StarLinear problem. Therefore $\mathcal{S}_1 = \mathcal{S}_2$, the optimal solution is unique, and in this solution, all workers finish computing simultaneously. $\square$

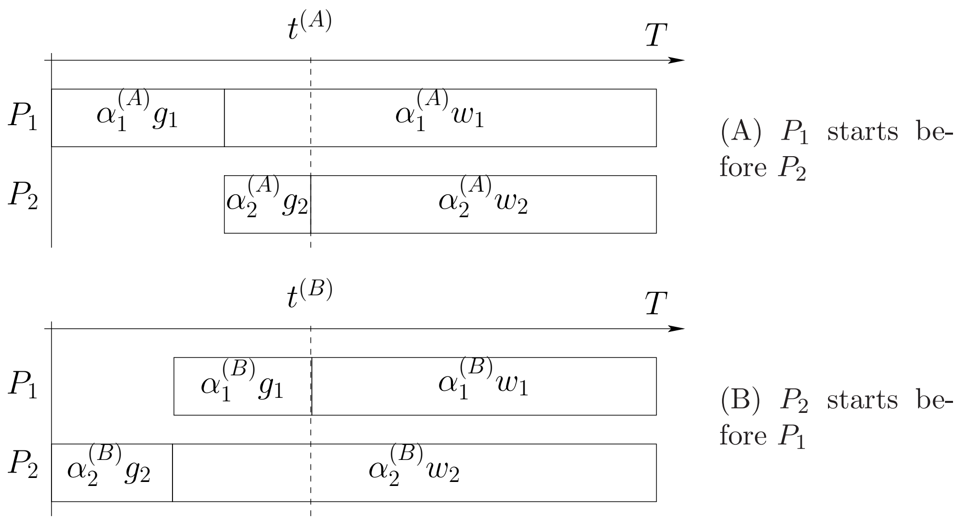

Figure 4: Comparison of the two possible orderings.

Altogether, this concludes the proof of Proposition 1. □

To be able to characterize the optimal solution, there remains to determine the best ordering for the master P_0 to send work to the workers:

Proposition 2. *An optimal ordering for the* StarLinear *problem is obtained by serving the workers in the ordering of non decreasing link capacities g_i.*

To the best of our knowledge, Proposition 2 is a new result. Although closed-from solutions to the heterogeneous StarLinear problem are given in [15], they require that (i) the optimal ordering be known, and (ii) that all workers finish computing simultaneously. Note that we have shown that this latter property indeed holds for the optimal schedule (as characterized by Proposition 2).

Proof. The proof is based on the comparison of the amount of work that is performed by the first two workers, and then proceeds by induction. To simplify notations, assume that P_1 and P_2 have been selected as the first two workers. There are two possible orderings, as illustrated in Figure 4. For each ordering, we determine the total number of units of load $\alpha_1+\alpha_2$ that are processed in T time-units, and the total occupation time, t, of the communication medium during this time interval. We denote with upper-script (A) (resp. (B)) all the quantities related to the first (resp. second) ordering.

Let us first determine the different quantities $\alpha_1^{(A)}$, $\alpha_2^{(A)}$, and $t^{(A)}$ for the upper ordering in Figure 4:

- From the equality $\alpha_1^{(A)}(g_1 + w_1) = T$, we get:

$$\alpha_1^{(A)} = \frac{T}{g_1 + w_1}. \tag{1}$$

- Using the equality $\alpha_1^{(A)} g_1 + \alpha_2^{(A)}(g_2 + w_2) = T$, we obtain (from equation (1)):

$$\alpha_2^{(A)} = \frac{T}{g_2 + w_2} - \frac{T g_1}{(g_1 + w_1)(g_2 + w_2)}. \tag{2}$$

Therefore, the overall number of processed units of load is equal to (by (1) and (2)):

$$\alpha_1^{(A)} + \alpha_2^{(A)} = \frac{T}{g_1 + w_1} + \frac{T}{g_2 + w_2} - \frac{Tg_1}{(g_1 + w_1)(g_2 + w_2)}. \tag{3}$$

and the overall occupation time of the network medium is equal to (using the previous equalities and $t^{(A)} = \alpha_1^{(A)} g_1 + \alpha_2^{(A)} g_2$):

$$t^{(A)} = \frac{Tg_1}{g_1 + w_1} + \frac{Tg_2}{g_2 + w_2} - \frac{Tg_1 g_2}{(g_1 + w_1)(g_2 + w_2)}. \tag{4}$$

A similar expression can be obtained for scenario (B) and we derive that:

$$(\alpha_1^{(A)} + \alpha_2^{(A)}) - (\alpha_1^{(B)} + \alpha_2^{(B)}) = \frac{T(g_2 - g_1)}{(g_1 + w_1)(g_2 + w_2)}, \tag{5}$$

and

$$t^{(A)} = t^{(B)}. \tag{6}$$

Thanks to these expressions, we know that the occupation of the communication medium does not depend on the communication ordering. Therefore, we only need to consider the number of processed units of load in both situations. Equation (5) indicates that one should send chunks to the worker with the smallest g_i first.

We now proceed to the general case. Suppose that the workers are already sorted so that $g_1 \le g_2 \le \ldots \le g_p$. Consider an optimal ordering of the communications σ, where chunks are sent successively to $P_{\sigma(1)}, P_{\sigma(2)}, \ldots P_{\sigma(p)}$. Let us assume that there exists an index i such that $\sigma(i) > \sigma(i+1)$. Furthermore, let us consider the smallest such index if multiple ones exist. Consider now the following ordering:

$$P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}, P_{\sigma(i+1)}, P_{\sigma(i)}, P_{\sigma(i+2)}, \ldots P_{\sigma(p)}.$$

Then, $P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}, P_{\sigma(i+2)}, \ldots P_{\sigma(p)}$ perform exactly the same number of units of load, since the exchange does not affect the overall communication time, but together, $P_{\sigma(i+1)}$ and $P_{\sigma(i)}$ perform $\frac{T(g_{\sigma(i)} - g_{\sigma(i+1)})}{(g_{\sigma(i+1)} + w_{\sigma(i+1)})(g_{\sigma(i)} + w_{\sigma(i)})}$ more units of load, where T denotes the remaining time after communications to $P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}$. Therefore, the initial ordering σ is not optimal, which is a contradiction. Therefore index i does not exist, which proves that in an optimal ordering the workers are sorted by non-decreasing values of the g_i's. $\square$

According to Proposition 2, we now re-order the workers so that $g_1 \le g_2 \le \ldots \le g_p$. The following linear program aims at computing the optimal distribution of the load:

$$\begin{array}{l}
\textsc{Minimize } T_f, \\
\textsc{subject to} \\
\left\{\begin{array}{lll}
(1) & \alpha_i \ge 0 & 1 \le i \le p \\
(2) & \sum_{i=1}^{p} \alpha_i = W_{\text{total}} & \\
(3) & \alpha_1 g_1 + \alpha_1 w_1 \le T_f & \text{(first communication)} \\
(4) & \sum_{j=1}^{i} \alpha_j g_j \; + \alpha_i w_i \le T_f & (i\text{-th communication})
\end{array}\right.
\end{array}$$

Theorem 1. *The optimal solution for the* StarLinear *problem is given by the solution of the linear program above.*

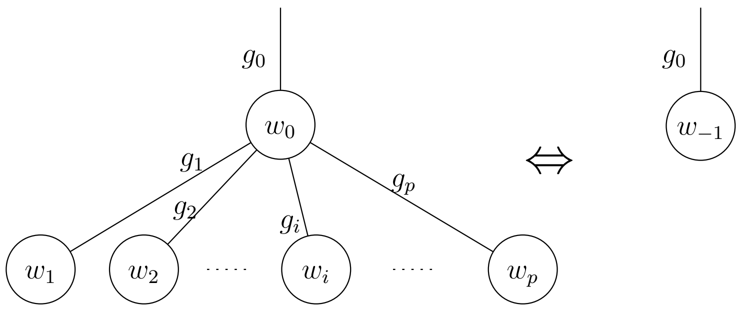

Figure 5: Replacing a single-level tree by an equivalent node.

Proof. Direct consequence of Propositions 1 and 2. Note that inequalities (3) and (4) will be in fact equalities in the solution of the linear program, so that we can easily derive a closed-form expression for T_f. □

We point out that this is linear programming with rational numbers, hence of polynomial complexity. Finally, we consider the variant where the master is capable of processing chunks (with computing power w_0) while communicating to one of its children. It is easy to see that the master is kept busy at all times (otherwise more units of load could be processed). The optimal solution is therefore given by the following linear program (where $g_1 \leq g_2 \leq \ldots \leq g_p$ as before):

$$
\begin{array}{l}
\textsc{Minimize } T_f,\\
\textsc{subject to}\\
\left\{\begin{array}{ll}
(1)\ \alpha_i \geq 0 & 0 \leq i \leq p\\
(2)\ \sum_{i=0}^{p} \alpha_i = W_{\text{total}} & \\
(3)\ \alpha_0 w_0 \leq T_f & \text{(computation of the master)}\\
(4)\ \alpha_1 g_1 + \alpha_1 w_1 \leq T_f & \text{(first communication)}\\
(5)\ \sum_{j=1}^{i} \alpha_j g_j \ + \alpha_i w_i \leq T_f & \text{(i-th communication)}
\end{array}\right.
\end{array}
$$

3.2 Tree network and linear cost model

All the results in the previous section can be extended to a tree-shaped network. There is however a key difference with the beginning of Section 3.1: each worker now is capable of computing and communicating to one of its children simultaneously. However, because of the one-round hypothesis, no overlap can occur with the incoming communication from the node's parent.

We use a recursive approach, which replaces any set of leaves and their parent by a single worker of equivalent computing power:

Lemma 3. *A single-level tree network with parent P_0 (with input link of capacity g_0 and cycle-time w_0) and p children P_i, $1 \leq i \leq p$ (with input link of capacity g_i and cycle-time w_i), where $g_1 \leq g_2 \leq \ldots \leq g_p$, is equivalent to a single node with same input link capacity g_0 and* *cycle-time* $w_{-1} = 1/W$ *(see Figure 5), where* W *is the solution to the linear program:*

$$\begin{array}{l}\textsc{Maximize } W,\\ \textsc{subject to}\\ \left\{\begin{array}{lr}(1)\ \alpha_i \geq 0 & 0 \leq i \leq p\\ (2)\ \sum_{i=0}^{p} \alpha_i = W & \\ (3)\ Wg_0 + \alpha_0 w_0 \leq 1 & \\ (4)\ Wg_0 + \alpha_1 g_1 + \alpha_1 w_1 \leq 1 & \\ (5)\ Wg_0 + \sum_{j=1}^{i} \alpha_j g_j \ + \alpha_i w_i \leq 1 & \end{array}\right.\end{array}$$

Proof. Here, instead of minimizing the time T_f required to execute load W, we aim at determining the maximum amount of work W that can be processed in one time-unit. Obviously, after the end of the incoming communication, the parent should be constantly computing . We know that all children (i) participate in the computation and (ii) terminate execution at the same-time. Finally, the optimal ordering for the children is given by Proposition 2. This completes the proof. Note that inequalities (3), (4) and (5) will be in fact equalities in the solution of the linear program, so that we can easily derive a closed-form expression for $w_{-1} = 1/W$. □

Lemma 3 provides a constructive way of solving the problem for a general tree. First we traverse it from bottom to top, replacing each single-level tree by the equivalent node. We do this until there remains a single star. We solve the problem for the star, using the results of Section 3.1. Then we traverse the tree from top to bottom, and undo each transformation in the reverse ordering. Going back to a reduced node, we know which amount of time it is working. Knowing the ordering, we know which amount of time each of the children is working. If one of this children is a leaf node, we have computed its load. If it is a reduced node, we apply the transformation recursively.

Instead of this pair of tree traversals, we could write down the linear program for the whole tree: when it receives something, a given node knows exactly what to do: compute itself all the remaining time, and feed its children in decreasing bandwidth order. However, the size of the linear program would grow proportionally to the size of the tree, hence the recursive solution is to be preferred.

3.3 Star network and affine cost model

To the best of our knowledge, the complexity of the StarAffine problem is open. The main difficulty arises from resource selection: contrarily to the linear case where all workers participate in the optimal solution, it seems difficult to decide which resources to use when latencies are introduced. However, the second property proved in Proposition 1, namely simultaneous termination, still holds true:

Proposition 3. *In an optimal solution of the* StarAffine *problem, all participating workers finish computing at the same time.*

Proof. The proof is very similar to the StarLinear case. Details can be found in Appendix A. □

Proposition 4. *If the load is large enough, then for any optimal solution (i) all workers participate and (ii) chunks must be sent in the order of non decreasing link capacities* g_i.

Proof. Consider a valid solution of the StarAffine problem with time bound T. Suppose, without loss of generality, that $\alpha_{\sigma(1)}$ units of load are sent to $P_{\sigma(1)}$, then $\alpha_{\sigma(2)}$ to $P_{\sigma(2)}$, ... and finally $\alpha_{\sigma(k)}$ to P_k, where $\mathcal{S} = \{P_{\sigma(1)}, \dots, P_{\sigma(k)}\}$ is the set of workers that participate to the computation. Here, σ represents the communication ordering and is a one-to-one mapping from $(1 \dots k]$ to $[1 \dots n]$. Moreover, let n^{TASK} denote the optimal number of units of load that can be processed using this set of workers and this ordering.

- Consider the following instance of the StarLinear problem, with k workers $P'_{\sigma(1)}, \dots, P'_{\sigma(k)}$, where $\forall i, \quad G'_i = 0, W'_i = 0, g'_i = g_i, w'_i = w_i$ and $T' = T$. Since all computation and communication latencies have been taken out, the optimal number of units of load n_1^{TASK} processed by this instance is larger than the number of units of load n^{TASK} processed by the initial platform. From Theorem 1, the value of n_1^{TASK} is given by a formula
$$n_1^{\text{TASK}} = f(\mathcal{S}, \sigma) \cdot T,$$
where $f(\mathcal{S}, \sigma)$ is either derived from the linear program, or explicitly given by a closed form expression in [15]. What matters here is that the value of n_1^{TASK} is proportional to T.

- Consider now the following instance of the StarLinear problem, with k workers $P'_{\sigma(1)}, \dots, P'_{\sigma(k)}$, where $\forall i, \; G'_i = 0, W'_i = 0, g'_i = g_i, w'_i = w_i$ and $T' = T - \sum_{i \in \mathcal{S}}(G_i + W_i)$. Clearly, the optimal number of units of load n_2^{TASK} processed by this instance of the StarLinear problem is lower than n^{TASK}, since it consists in adding all the communication and computation latencies before the beginning of the processing. Moreover, as previously n_2^{TASK} is given by the formula
$$n_2^{\text{TASK}} = f(\mathcal{S}, \sigma)(T - \sum_{i \in \mathcal{S}}(G_i + W_i)).$$

Therefore, we have
$$f(\mathcal{S}, \sigma) \left(1 - \frac{\sum_{i \in \mathcal{S}}(G_i + W_i)}{T}\right) \leq \frac{n^{\text{TASK}}}{T} \leq f(\mathcal{S}, \sigma).$$

Hence, when T becomes arbitrarily large, then the throughput of the platform, $\frac{n^{\text{TASK}}}{T}$, becomes arbitrarily close to $f(\mathcal{S}, \sigma)$, i.e. the optimal throughput if there were no communication and computation latencies. Moreover, we have proved that if there are no latencies, then $f(\mathcal{S}, \sigma)$ is maximal when $\mathcal{S}$ is the set of all the workers, and when σ satisfies
$$g_j > g_i \Longrightarrow \sigma(i) > \sigma(j).$$

Therefore, when T is sufficiently large, then all the workers should be used and the chunks should be sent to workers in the ordering of non decreasing link capacities g_i. In this case, if $g_1 \leq \dots \leq g_n$, then the following linear system provides an asymptotically optimal solution
$$\forall i, \quad \sum_{k=1}^{i}(G_k + g_k \alpha_k) + W_i + g_i w_i = T.$$

This solution is optimal if all g_i are different. Determining the best way to break ties among workers having the same bandwidth is an open question. □

In the general case, we do not know whether there exists a polynomial-time algorithm to solve the StarAffine problem. However, we can provide the solution (with potentially exponential cost) as follows: we start from the mixed linear programming formulation of the problem proposed by Drozdowski [19], and we extend it to include resource selection. In the following program, y_j is a boolean variable that equals 1 if P_j participates in the solution, and $x_{i,j}$ is a boolean variable that equals 1 if P_j is chosen for the i-th communication from the master:

Minimize T_f,
subject to

$$\left\{\begin{array}{llllll}
(1)\ \alpha_i \geq 0 & 1 \leq i \leq p & (2)\ \sum_{i=1}^{p} \alpha_i = W_{\text{total}} & & (3)\ y_j \in \{0,1\} & 1 \leq j \leq p \\
(4)\ x_{i,j} \in \{0,1\} & 1 \leq i,j \leq p & (5)\ \sum_{i=1}^{p} x_{i,j} = y_j & 1 \leq j \leq p & & \\
(6)\ \sum_{j=1}^{p} x_{i,j} \leq 1 & 1 \leq i \leq p & (7)\ \alpha_j \leq W y_j & 1 \leq j \leq p & & \\
\multicolumn{6}{l}{(8)\ \sum_{j=1}^{p} x_{1,j}(G_j + \alpha_j g_j + W_j + \alpha_j w_j) \leq T_f \qquad \text{(first communication)}} \\
\multicolumn{6}{l}{(9)\ \sum_{k=1}^{i-1} \sum_{j=1}^{p} x_{k,j}(G_j + \alpha_j g_j) + \sum_{j=1}^{p} x_{i,j}(G_j + \alpha_j g_j + W_j + \alpha_j w_j) \leq T_f} \\
\multicolumn{6}{r}{2 \leq i \leq p \qquad (i\text{-th communication})}
\end{array}\right.$$

Equation (5) implies that P_j is involved in exactly one communication if $y_j = 1$, and in no communication otherwise. Equation (6) states that at most one worker is activated for the i-th communication; if $\sum_{j=1}^{p} x_{i,j} = 0$, the i-th communication disappears. Equation (7) states that no work is given to non participating workers (those for which $y_j = 0$) but is automatically fulfilled by participating ones. Equation (8) is a particular case of equation (9), which expresses that the worker selected for the i-th communication (where $i = 1$ in equation (8) and $i \geq 2$ in equation (9)) must wait for the previous communications to complete before starting its own communication and computation, and that all this quantity is a lower bound of the makespan. Contrarily to the formulation of Drozdowski [19], this mixed linear program always has a solution, even if a strict subset of the resources are participating. We state this result formally:

Proposition 5. *The optimal solution for the* StarAffine *problem is given by the solution of the mixed linear program above (with potentially exponential cost).*

3.4 Tree network and affine cost model

This is the most difficult platform/model combination, and very few results are known. However, we point out that Proposition 4 can be extended to arbitrary tree networks: when T becomes arbitrarily large, latencies become negligible, and an asymptotically optimal behavior is obtained by involving all resources and by having each parent communicate with its children in order of non decreasing link capacities.

4 Multi-round algorithms

Under the one-port communication model described in Section 2.1, one-round algorithms lead to poor utilization of the workers. As seen in Figure 3, worker P_i remains idle from time 0 to time T_i. To alleviate this problem, *multi-round* algorithms have been proposed. These algorithms dispatch the load in multiple rounds of work allocation and thus improve overlap of communication with computation. By comparison with one-round algorithms, work on

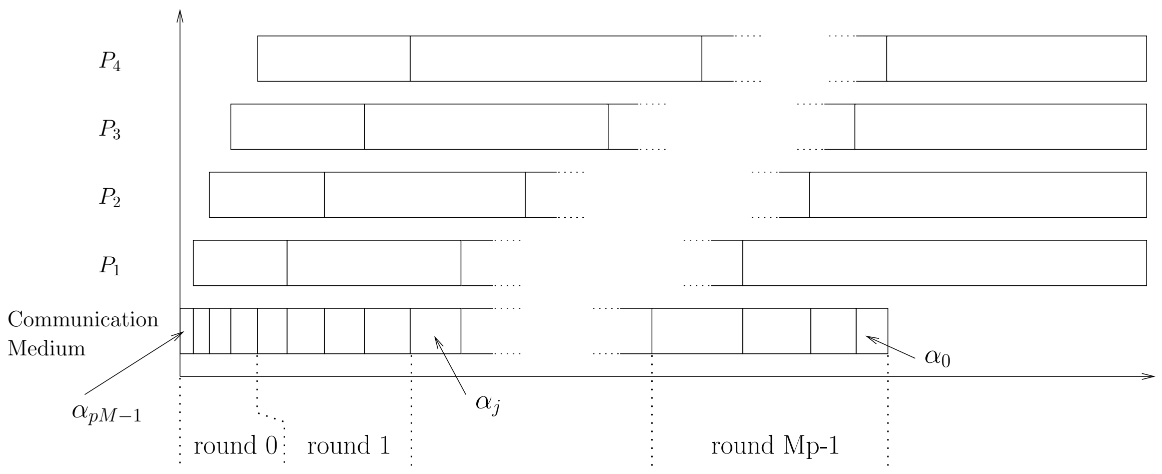

Figure 6: Pattern of a solution for dispatching the load of a divisible job, using a bus network ($g_i = g$), in multiple rounds, for 4 workers. All 4 workers complete execution at the same time. Chunk sizes increase during each of the first $M - 1$ rounds and decrease during the last round.

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? The majority of works on multi-round algorithms assume that the number of rounds is fixed and we review corresponding results and open questions in Section 4.1. In Section 4.2 we describe recent work that attempts at answering question (ii). Finally, we deal with asymptotic results in Section 4.3, which of course are of particular interest when the total load W_{total} is very large.

4.1 Fixed number of rounds, homogeneous star network, affine Costs

As for one-round algorithms, a key question is that of the order in which chunks should be sent to the workers. However, to the best of our knowledge, all previous work on multi-round algorithms with fixed number of rounds only offer solution for homogeneous platforms, in which case worker ordering is not an issue. Given a fixed number of rounds M, the load is divided into $p \times M$ chunks, each corresponding to a α_j ($j = 0, \ldots, pM - 1$) units of load such that $\sum_{j=0}^{pM-1} \alpha_j = W_{\text{total}}$. The objective is to determine the α_j values that minimize the overall makespan.

Intuitively, the chunk size should be small in the first rounds, so as to start all workers as early as possible and thus maximize overlap of communication with computation. It has be shown that the chunk sizes should then increase to optimize the usage of the total available bandwidth of the network and to amortize the potential overhead associated with each chunk. In the last round, chunk sizes should be decreasing so that all workers finish computing at the same time (following the same principle as in Section 3). Such a schedule is depicted in Figure 6 for four workers.

Bharadwaj et al. were the first to address this problem with the multi-installment scheduling algorithm described in [9]. They reduce the problem of finding an optimal schedule to that of finding a schedule that has essentially the following three properties: (i) there is no idle time between consecutive communications on the bus; (ii) there is no idle time between consecutive computation on each worker; and (iii) all workers should finish computing at the same time. These properties guarantee that the network and compute resources are at maximum utilization.

In [9], the authors consider only linear costs for both communication and computation. The three conditions above make it possible to obtain a recursion on the α_j series. This recursion must then be solved to obtain a close form expression for the chunk sizes. One method to solve the recursion is to use generating functions and the rational expansion theorem [28].

We recently extended the multi-installment approach to account for affine costs [43]. This was achieved by rewriting the chunk size recursion in a way that is more amenable to the use of generating functions when fixed latencies are incurred for communications and computations. Since it is more general but similar in spirit, we only present the affine case here.

For technical reasons, as in [9], we number the chunks in the reverse order in which they are allocated to workers: the last chunk is numbered 0 and the first chunk is numbered $Mp-1$. Instead of developing a recursion on the α_j series directly, we define $\gamma_j = \alpha_j * w$, i.e. the time to compute a chunk if size α_j on a worker not including the W latency. Recall that in this section we only consider homogeneous platforms and thus $w_q = w$, $G_q = G$, $g_q = g$, and $G_q = G$ for all workers $q = 1, \ldots, p$. The time to communicate a chunk of size α_j to a worker is $G + \gamma_i/R$, where R is the computation-communication ratio of the platform: w/g. We can now write the recursion on the γ_j series:

$$\begin{aligned}
\forall\ j \geq P \qquad & W + \gamma_j = (\gamma_{j-1} + \gamma_{j-2} + \gamma_{j-3} + \cdots + \gamma_{j-N})/R + P \times G && (7)\\
\forall\ 0 \leq j < P \quad & W + \gamma_j = (\gamma_{j-1} + \gamma_{j-2} + \gamma_{j-3} + \cdots + \gamma_{j-N})/R + j \times G + \gamma_0 && (8)\\
\forall\ j < 0 \qquad & \gamma_j = 0 && (9)
\end{aligned}$$

Eq. 7 ensures that there is no idle time on the bus and at each worker in the first $M-1$ rounds. More specifically, Eq. 7 states that a worker must compute a chunk in exactly the time required for all the next P chunks to be communicated, including the G latencies. This equation is valid only for $j \geq P$. For $j < P$, i.e. the last round, the recursion must be modified to ensure that all workers finish computing at the same time, which is expressed in Eq. 8. Finally, Eq. 9 ensures that the two previous equations are correct by taking care of out-of-range α_j terms. This recursion describes an infinite α_j series, and the solution to the scheduling problems is given by the first pM values.

As in [9], we use generating functions as they are convenient tools for solving complex recursions elegantly. Let $\mathcal{G}(x)$ be the generating function for the series γ_j, that is $\mathcal{G}(x) = \sum_{j=0}^{\infty} \gamma_j x^j$. Multiplying Eq. 7 and Eq. 8, manipulating the indices, and summing the two gives:

$$\mathcal{G}(x) \quad = \quad \frac{(\gamma_0 - P \times G)(1-x^P) + (P \times G - W) + G(\frac{x(1-x^{P-1})}{1-x} - (P-1)x^P)}{(1-x) - x(1-x^P)/R}.$$

The rational expansion method [28] can then be used to determine the coefficients of the above polynomial fraction, given the roots of the denominator polynomial, $Q(x)$. The values of the γ_j series, and thus of the α_j series, follow directly. If $Q(x)$ has only roots of degree 1 then the simple rational expansion theorem can be used directly. Otherwise the more complex general rational expansion theorem must be used. In [43] we show that if $R \neq P$ then $Q(x)$ has only roots of degree one. If $R = P$, then the only root of degree higher than 1 is root $x = 1$ and it is of degree 2, which makes the application of the general theorem straightforward. Finally,

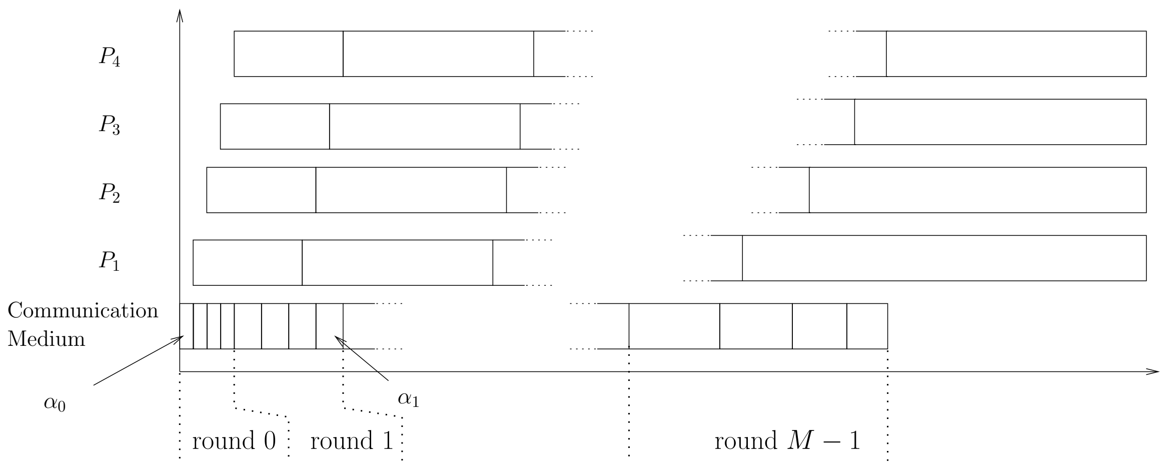

Figure 7: Pattern of a solution for dispatching the load of a divisible job, using a bus network ($g_i = g$), in multiple **uniform** rounds, for 4 workers. All workers complete execution at the same time. Chunk sizes a fixed within the first $M - 1$ rounds but increase from round to round. Chunk sizes decrease during the last round.

the value of γ_0 can be computed by writing that $\sum_{j=0}^{Mp-1} \gamma_j = W_{\text{total}} \times w$. All technical details on the above derivations are available in a technical report [43]. We have thus obtained a closed-form expression for optimal multi-installment schedule on a homogeneous star network with affine costs.

4.2 Computed number of rounds, star network, affine costs

The work presented in the previous section assumes that the number of rounds is fixed and provided as input to the scheduling algorithm. In the case of linear costs, the authors in [10] recognize that infinitely small chunks would lead to an optimal multi-round schedule, which implies an infinite number of rounds. When considering more realistic affine costs there is a clear trade-off. While using more rounds leads to better overlap of communication with computation, using fewer rounds reduces the overhead due to the fixed latencies. Therefore, an key question is: What is the optimal number of rounds for multi-round scheduling on a star network with affine costs?

While this question is still open for the recursion described in Section 4.1, our work in [45] proposes a scheduling algorithm, Uniform Multi-Round (UMR), that uses a restriction on the chunk size: all chunks sent to workers during a round are identical. This restriction limits the ability to overlap communication with computation, but makes it possible to derive an optimal number of rounds due to a simpler recursion on chunk sizes. Furthermore, this approach is applicable to both homogeneous and heterogeneous platforms. We only describe here the algorithm in the homogeneous case. The heterogeneous case is similar but involves more technical derivations and we refer the reader to [42] for all details.

As seen in Figure 7, chunks of identical size are sent out to workers within each round. Because chunks are uniform it is not possible to obtain a schedule with no idle time in which each worker finishes receiving a chunk of load right when it can start executing it. Note in Figure 7 that workers can have received a chunk entirely while not having finished to compute the previous chunk. The condition that a worker finishes receiving a chunk right when it can start computing is only enforced for the worker P_p, which is also seen in the figure. Finally, the uniform round restriction is removed for the last round. As in the multi-installment approach described in Section 4.1, chunks of decreasing sizes are sent to workers in the last round so that they can all finish computing at the same time.

Let α_j be the chunk size at round j, which is used for all workers during that round. We derive a recursion on the chunk size. To maximize bandwidth utilization, the master must finish sending work for round $j+1$ to all workers right when worker P finishes computing for round j. This can be written as

$$W + \alpha_j w = P(G + \alpha_{j+1} g), \tag{10}$$

which reduces to

$$\alpha_j = \left(\frac{g}{Pw}\right)^j (\alpha_0 - \gamma) + \gamma, \tag{11}$$

where $\gamma = \frac{1}{w-Pg} \times (PG - W)$. The case in which $w - Pg = 0$ leads to a simpler recursion and we do not consider it here for the sake of brevity.

Given this recursion on the chunk sizes, it is possible to express the scheduling problem as a constrained minimization problem. The total makespan, $\mathcal{M}$, is:

$$\mathcal{M}(M, \alpha_0) = \frac{W_{\text{total}}}{P} + MW + \frac{1}{2} \times P(G + g\alpha_0),$$

where the first term is the time for worker P to perform its computations, the second term the overhead incurred for each of these computations, and the third term is the time for the master to dispatch all the chunks during the first round. Note that the $\frac{1}{2}$ factor in the above equation is due to the last round during which UMR does not keep chunk sizes uniform so that all workers finish computing at the same time (see [45] for details).

Since all chunks must satisfy the constraint that they add up to the entire load, one can write that:

$$\mathcal{G}(M, \alpha_0) = \sum_{j=0}^{M-1} P\alpha_j - W_{\text{total}} = 0. \tag{12}$$

The scheduling problem can now be expressed as the following constrained optimization problem: minimize $\mathcal{M}(M, \alpha_0)$ subject to $\mathcal{G}(M, \alpha_0) = 0$. An analytical solution using the Lagrange Multiplier method [7] is given in [45], which leads to a single equation for the optimal number of round, M^*. This equation cannot be solved analytically but is eminently amenable to a numerical solution, e.g. using a bisection method.

The UMR algorithm is a heuristic and has been evaluated in simulation for a large number of scenarios [42]. In particular, a comparison of UMR with the multi-installment algorithm discussed in Section 4.1 demonstrates the following. The uniform chunk restriction minimally degrades performance compared to multi-installment when latencies are small (i.e. when costs are close to being linear). However, as soon as latencies become significant, this performance degradation is offset by the fact that an optimal number of rounds can be computed and UMR outperforms multi-installment consistently. Finally, note that a major benefit of UMR is that, unlike multi-installment, it is applicable to heterogeneous platforms. In this case the question of worker ordering arises and UMR uses the same criterion as that given in Proposition 2: workers are ordered by non-decreasing link capacities.

4.3 Asymptotic performance, star network, affine costs

In this section, we derive asymptotically optimal algorithms for the multi-round distribution of divisible loads. As in previous sections, we use a star network with affine costs.

The sketch of the algorithm that we propose is as follows: the overall processing time T is divided into k regular periods of duration T_p (hence $T = kT_p$, but k (and T_p) are yet to be determined). During a period of duration T_p, the master sends α_i units of load to worker P_i. It may well be the case that not all the workers are involved in the computation. Let $\mathcal{I} \subset \{1, \dots, p\}$ represent the subset of indices of participating workers. For all $i \in \mathcal{I}$, the α_i's must satisfy the following inequality, stating that communication resources are not exceeded:

$$\sum_{i \in \mathcal{I}} (G_i + \alpha_i g_i) \leq T_p. \tag{13}$$

Since the workers can overlap communications and processing, the following inequalities also hold true:

$$\forall i \in \mathcal{I}, \quad W_i + \alpha_i w_i \leq T_p.$$

Let us denote by $\frac{\alpha_i}{T_p}$ the average number of units of load that worker P_i processes during one time unit, then the system becomes

$$\left\{ \begin{array}{lll} \forall i \in \mathcal{I}, \quad \dfrac{\alpha_i}{T_p} w_i & \leq 1 - \frac{W_i}{T_p} & \text{(no overlap)} \\ \displaystyle\sum_{i \in \mathcal{I}} \frac{\alpha_i}{T_p} g_i & \leq 1 - \frac{\sum_{i \in \mathcal{I}} G_i}{T_p} & \text{(1-port model)} \end{array} \right. ,$$

and our aim is to maximize the overall number of units of load processed during one time unit, i.e. $n = \sum_{i \in \mathcal{I}} \frac{\alpha_i}{T_p}$.

Let us consider the following linear program:

$$\begin{array}{l} \textsc{Maximize} \sum_{i=1}^{p} \frac{\alpha_i}{T_p}, \\ \text{SUBJECT TO} \\ \left\{ \begin{array}{l} \forall 1 \leq i \leq p, \quad \dfrac{\alpha_i}{T_p} w_i \leq 1 - \dfrac{\sum_{i=1}^{p} G_i + W_i}{T_p} \\ \displaystyle\sum_{i=1}^{p} \frac{\alpha_i}{T_p} g_i \leq 1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p} \end{array} \right. \end{array}$$

This linear program is more constrained than the previous one, since $1 - \frac{W_i}{T_p}$ and $1 - \frac{\sum_{i \in \mathcal{I}} G_i}{T_p}$ have been replaced by $1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}$ in p inequalities. The linear program can be solved using a package similar to Maple [14] (we have rational numbers), but it turns out that the technique developed in [5] enables us to obtain the solution in closed form. We refer the reader to [5] for the complete proof. Let us sort the g_i's so that $g_1 \leq g_2 \leq \ldots \leq g_p$, and let q be the largest index so that $\sum_{i=1}^{q} \frac{g_i}{w_i} \leq 1$. If $q < p$, let ϵ denote the quantity $1 - \sum_{i=1}^{q} \frac{g_i}{w_i}$. If $p = q$, we set $\epsilon = g_{q+1} = 0$, in order to keep homogeneous notations. This corresponds to the case where the full use of all the workers does not saturate the 1-port assumption for out-going communications from the master. The optimal solution to the linear program is obtained with

$$\forall 1 \leq i \leq q, \quad \frac{\alpha_i}{T_p} = \frac{1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}}{g_i}$$

and (if $q < p$):

$$\frac{\alpha_{q+1}}{T_p} = \left(1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}\right)\left(\frac{\epsilon}{g_{q+1}}\right),$$

and $\alpha_{q+2} = \alpha_{q+3} = \ldots = \alpha_p = 0$.

With these values, we obtain:

$$n \geq \sum_{i=1}^{p} \frac{\alpha_i}{T_p} = \left(1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}\right)\left(\sum_{i=1}^{q} \frac{1}{w_i} + \frac{\epsilon}{g_{p+1}}\right).$$

Let us denote by n_{opt} the optimal number of units of load that can be processed within one unit of time. If we denote by β_i^* the optimal number of units of load that can be processed by worker P_i within one unit of time, the β_i^*'s satisfy the following set of inequalities, in which the G_i's have been removed:

$$\begin{cases} \forall 1 \leq i \leq p, \quad \beta_i^* w_i \leq 1 \\ \sum_{i=1}^{p} \beta_i^* g_i \leq 1 \end{cases}$$

Here, because we have no latencies, we can safely assume that all the workers are involved (and let $\beta_i^* = 0$ for some of them). We derive that:

$$n_{\text{opt}} \leq \left(1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}\right)\left(\sum_{i=1}^{q} \frac{1}{w_i} + \frac{\epsilon}{g_{q+1}}\right).$$

If we consider a large number B of units of load to be processed and if we denote by T_{opt} the optimal time necessary to process them, then

$$T_{\text{opt}} \geq \frac{B}{n_{\text{opt}}} \geq \frac{B}{\left(\sum_{i=1}^{q} \frac{1}{w_i} + \frac{\epsilon}{g_{q+1}}\right)}.$$

Let us denote by T the time necessary to process all B units of load with the algorithm that we propose. Since the first period is lost for processing, then the number k of necessary periods satisfies $nT_p(k-1) \geq B$ so that we choose

$$k = \left\lceil \frac{B}{nT_p} \right\rceil + 1.$$

Therefore,

$$T \leq \frac{B}{n} + 2T_p \leq \frac{B}{\left(\sum_{i=1}^{q} \frac{1}{w_i} + \frac{\epsilon}{g_{q+1}}\right)}\left(\frac{1}{1 - \sum_{i=1}^{p} \frac{G_i + W_i}{T_p}}\right) + 2T_p,$$

and therefore, if $T_p \geq 2\sum_{i=1}^{p} G_i + W_i$,

$$T \leq T_{\text{opt}} + 2\sum_{i=1}^{p}(G_i + W_i)\frac{T_{\text{opt}}}{T_p} + 2T_p.$$

Finally, if we set $T_p = \sqrt{T_{\text{opt}}}$, we check that

$$T \leq T_{\text{opt}} + 2\left(\sum_{i=1}^{p}(G_i + W_i) + 1\right)\sqrt{T_{\text{opt}}} = T_{\text{opt}} + O(\sqrt{T_{\text{opt}}}),$$

and

$$\frac{T}{T_{\text{opt}}} \leq 1 + 2\left(\sum_{i=1}^{p}(G_i + W_i) + 1\right)\frac{1}{\sqrt{T_{\text{opt}}}} = 1 + O\left(\frac{1}{\sqrt{T_{\text{opt}}}}\right),$$

which completes the proof of the asymptotic optimality of our algorithm.

Note that resource selection is part of our explicit solution to the linear program: to give an intuitive explanation of the analytical solution, workers are greedily selected, fast-communicating workers first, as long as the communication to communication-added-to-computation ratio is not exceeded.

We formally state our main result:

Theorem 2. *For arbitrary values of G_i, g_i, W_i and w_i and assuming communication-computation overlap, the previous periodic multi-round algorithm is asymptotically optimal. Closed-form expressions for resource selection and task assignment are provided by the algorithm, whose complexity does not depend upon the total amount of work to execute.*

5 Extensions

5.1 Other Platform Topologies

The divisible load scheduling problem has been studied for a variety of platforms. Although in this paper we have focused on star and tree networks, because we feel they are the most relevant to current practice, we briefly review here work on a broader class of topologies.

The earliest divisible work load scheduling work studied Linear Network [17], and Bus/Star Networks [16]. Linear Network refers to scenarios in which each worker has two neighbors, and data is relayed from one worker to the next. The works in [17, 34, 8] give four divisible load scheduling on Linear Networks, and [34] compares these strategies. While Linear networks are not very common in practice, they serves as a good basis for studying more complex architectures such as 3-D Mesh and Hypercube.

The work in [20, 21] targets a circuit-switched 3-D Mesh network. The nodes in the network are essentially divided layers. The layers are then equivalent to nodes in a Linear Network. This layer concept is further formalized in [12, 20, 33] and used for Ging, Tree, Mesh, and Hypercube network. In this context the work in [27, 22] proposes and compares two data distribution methods: LLF(Largest Layer First) and NLF (Nearest Layer First). Finally, the work in [34, 35] targets k-dimensional Meshes, by reducing them to Linear Network of $(k-1)$-dimensional Meshes. Finally note that Hypercubes have also been studied without a layer models but via recursive sub-division [20, 35].

5.2 Factoring

Divisible load scheduling has also been studied when there is some degree of uncertainty regarding chunk computation or communication times. Such uncertainty can be due to the use of non-dedicated resources and/or to applications with data-dependent computational complexity. In these cases, a scheduling algorithm must base its decisions on performance predictions that have some error associated to them. In such a scenario the scheduling algorithms that we have surveyed in this paper would not be effective as schedules would lead to potentially long periods of idle times due to mispredictions of communication or computation times. The multi-round Factoring algorithms has been proposed that address the issue of chunk computation time uncertainty [31]. Instead of increasing chunk sizes throughout application execution, these algorithms start with large chunks and *decrease* chunk sizes, typically exponentially, at each round throughout application execution. Chunks are dispatched to workers in a greedy fashion to avoid the "wait for the last task" problem. Many flavors of factoring have been proposed [31], including adaptive ones in which chunk sizes are determined based on feedback from workers [4]. All these approaches have in common the use of large initial chunk sizes, which presents a major disadvantage: poor overlap of communication with computation at the beginning of application execution, as for the one-round algorithms described in Section 3. Note that this issue was not discussed in [31, 29, 4] as the authors assumes a fixed communication costs, as opposed to a cost that is linear or affine in the chunk size. Recently, has proposed strategies that initially increase and then decrease chunk sizes throughout execution to achieve good overlap of communication with computation as well as robustness to uncertainties [44].

6 Conclusion

The goal of this paper was to present a unified discussion of divisible load scheduling results for star and tree networks. In Section 3 we have discussed one-round algorithms for which the two main issues are: (i) selection and ordering of the workers, (ii) computation of the chunk sizes. Section 4 focused on multi-round algorithms, with the two main issues: (i) computation of chunk sizes at each round, and (ii) choice of the number of rounds. Section 4 also discussed multi-round scheduling for maximizing asymptotic application performance. For both classes of algorithms, we have revisited previously published results, presented novel results, and clearly identified open questions. Our overall goal was to identify promising research directions and foster that research thanks to our unified and synthesized framework.

We have discussed affine cost models and have seen that they often lead to significantly more complex scheduling problems than when linear models are assumed. These models are generally considered more realistic, and we even contend that, given current trends, linear models are quickly becoming increasingly inappropriate. In terms of communication, technology trends indicate that available network bandwidth is rapidly augmenting. Therefore, latencies account for an increasingly large fraction of communication costs. A similar observation can be made in terms of computation. Due to the absence of stringent synchronization requirements, divisible workload applications are amenable to deployment on widely distributed platforms. For instance, computational grids [24] are attractive for deploying large divisible workloads. However, initiating computation on these platforms incurs potentially large latencies (i.e., due to resource discovery, authentication, creation of new processes, etc.). Consequently, it is clear that divisible workload research should focus on affine cost models for both communication and computation.

References

[1] 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.

[2] 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.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer, Berlin, Germany, 1999.

[4] I. Banicescu and V. Velusamy. Load Balancing Highly Irregular Computations with the Adaptive Factoring. In *Proceedings of the Heterogeneous Computing Workshop (HCW'03), Fort Lauderdale, Florida*, April 2002.

[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, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads for star and tree networks: main results and open problems. Technical Report RR-2003-41, LIP, ENS Lyon, France, September 2003.

[7] D. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods.* Athena Scientific, Belmont, Mass., 1996.

[8] V. Bharadwaj, D. Ghose, and V. Mani. An Efficient Load Distribution Strategy for a Distributed Linear Network of Processors with Communication Delays. *Computer and Mathematics with Applications*, 29(9):95–112, 1995.

[9] V. Bharadwaj, D. Ghose, and V. Mani. Multi-installment load distribution in tree networks with delays,. *IEEE Trans. on Aerospace and Electronc Systems*, 31(2):555–567, 1995.

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

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

[12] J. Blazewicz and M. Drozdowski. Scheduling Divisible Jobs on Hypercubes. *Parallel Computing*, 21, 1995.

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

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

[15] S. Charcranoon, T.G. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on computers*, 49(9):987–991, September 2000.

[16] Y-C. Cheng and T.G. Robertazzi. Distributed Computation for a Tree-Network with Communication Delay. *IEEE transactions on aerospace and electronic systems*, 26(3), 1990.

[17] Y-C. Cheng and T.G. Robertazzi. Distributed Computation with Communication Delay. *IEEE transactions on aerospace and electronic systems*, 24(6), 1998.

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

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

[20] M. Drozdowski. *Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems*. PhD thesis, Poznan University of Technology, Poznan, Poland, 1998.

[21] M. Drozdowski and W. Glazek. Scheduling Divisible Loads in a Three-dimensional Mesh of Processors. *Parallel Computing*, 25(4), 1999.

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

[23] H. El-Rewini, T. G. Lewis, and H. H. Ali. *Task scheduling in parallel and distributed systems*. Prentice Hall, 1994.

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

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

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

[27] W. Glazek. A Multistage Load Distribution Strategy for Three-Dimensional Meshes. *Cluster Computing*, 6(1):31–39, 2003.

[28] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics*. Wiley, 1994.

[29] T. Hagerup. Allocating independent tasks to parallel processors: an experimental study. *J. Parallel and Distributed Computing*, 47:185–197, 1997.

[30] D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[31] S. Flynn Hummel. Factoring: a method for scheduling parallel loops. *Communications of the ACM*, 35(8):90–101, 1992.

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

[33] K. Li. Scheduling Divisible Tasks on Heterogeneous Linear Arrays with Applications to Layered Networks. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[34] K. Li. Improved Methods for Divisible Load Distribution on k-dimensional Meshes Using Pipelined Communications. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2003)*, page 81b, April 2003.

[35] K. Li. Parallel Processing of Divisible Loads on Partitionable Static Interconnection Networks. *Cluster Computing*, 6(1):47–55, 2003.

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

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

[38] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, New York, 1986.

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

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

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

[42] Y. Yang and H. Casanova. Multi-round algorithm for scheduling divisible workload applications: analysis and experimental evaluation. Technical Report CS2002-0721, Dept. of Computer Science and Engineering, University of California, San Diego, 2002.

[43] Y. Yang and H. Casanova. Extensions to the multi-installment algorithm: affine costs and output data transfers. Technical Report CS2003-0754, Dept. of Computer Science and Engineering, University of California, San Diego, July 2003.

[44] Y. Yang and H. Casanova. RUMR: Robust Scheduling for Divisible Workloads. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03).* IEEE Computer Society Press, 2003.

[45] Y. Yang and H. Casanova. UMR: A multi-round algorithm for scheduling divisible workloads. In *International Parallel and Distributed Processing Symposium (IPDPS'2003), Nice, France.* IEEE Computer Society Press, April 2003.

A Star network and affine cost model

In an optimal solution of the StarAffine problem, all participating workers terminate the execution at the same time.

Proof of Proposition 3. Let us consider an optimal solution of the StarAffine problem, and let us suppose, without loss of generality, that α_1 units of load are sent to P_1, then α_2 to P_2, ... and finally α_j to P_j, where $P_1, \dots, P_j$ denotes the set of workers that participate in the computation. By construction, all the α_i's are non zero. Consider the following linear program

$$\begin{array}{l} \textsc{Maximize } \sum \beta_i, \\ \textsc{subject to} \\ \left\{ \begin{array}{lll} \mathrm{LB}(i) & \forall i \leq j, & \beta_i \geq 0 \\ \mathrm{UB}(i) & \forall i \leq j, & \sum_{k=1}^{i} (G_k + \beta_k g_k) + W_i + \beta_i w_i \leq T \end{array} \right. \end{array}$$

Clearly, the α_i's satisfy the set of constraints above, and from any set of β_i's satisfying the set of inequalities, we can build a valid solution of the StarAffine problem that process exactly $\sum \beta_i$ units of load. Therefore, if we denote by $(\beta_1, \dots, \beta_j)$ an optimal solution of the linear program, we have $\sum \beta_i = \sum \alpha_i$.

Lemma 4. *For any optimal solution $(\beta_1, \dots, \beta_j)$ of the linear program, we have*

- $\beta_k > 0$ *for all* $k < j$
- $\mathrm{UB}(j)$ *is an equality, even if* P_j *does not process any task (i.e.* $\beta_j = 0$*)*

Proof. Suppose that there exists an index i such that $\beta_i = 0$, and denote by k the largest index such that $\beta_k = 0$. We have to distinguish between two cases:

Case $k < j$**.** Consider the number of units of load processed by workers P_k and P_{k+1}, and calculate the number of units of load that could be processed by P_{k+1} if P_k was removed from the set of participating workers. When both P_k and P_{k+1} are used, the communication medium is used by P_k and P_{k+1} during exactly $G_k + G_{k+1} + \beta_{k+1} g_{k+1}$ time-units. If we remove P_k from the set of participating workers, and let β'_{k+1} denote the number of processed units of load by worker P_{k+1}, then the condition

$$G_{k+1} + \beta'_{k+1} g_{k+1} \leq G_k + G_{k+1} + \beta_{k+1} g_{k+1}$$

ensures that the communication medium is not used longer than previously , and the condition

$$G_{k+1} + W_{k+1} + \beta'_{k+1}(g_{k+1} + w_{k+1}) \leq G_k + W_k + G_{k+1} + W_{k+1} + \beta_{k+1}(g_{k+1} + w_{k+1})$$

ensures that P_{k+1} finishes its processing before the time bound. Both condition are in fact equivalent to

$$\beta'_{k+1} \leq \beta_{k+1} + \min \left(\frac{G_k + W_k}{g_{k+1} + w_{k+1}}, \frac{G_k}{g_{k+1}} \right).$$

Therefore, if we set $\beta'_{k+1} = \beta_{k+1} + + \min \left(\frac{G_k + W_k}{g_{k+1} + w_{k+1}}, \frac{G_k}{g_{k+1}} \right)$, then the number of units of load processed by the platform where P_k has been removed is strictly larger, what is in contradiction with the optimality of the solution where each P_i processes β_i units of load. Thus, in an optimal solution involving workers $P_1, \dots, P_j$, none of first $j-1$ workers can be kept fully idle.

Case $k = j$. We use the same proof as above to prove that there is no other worker P_i with $i < j$ that is kept fully idle. Moreover, let us denote by t_j the time step when the first $j-1$ workers have received their work and therefore, at which the communication medium is free. Then, $T = t_j + G_j + W_j$, otherwise, P_j would be able to process some units of load. Therefore, UB(j) is an equality

Finally, if all β_i's are non zero, then the last worker P_j finishes at time T, otherwise it could proceed more units of load. Therefore, UB(j) is an equality, whether $\beta_j = 0$ or not. □

To prove that all participating workers complete their processing at time step T, we use, as previously, a few results on linear programming. It is known that an optimal solution (that may not be unique a priori) is obtained at some vertex $\mathcal{S}_1 = (\beta_1, \dots, \beta_j)$ of the convex polyhedron $\mathcal{P}$ defined by the set of inequalities. In solution $\mathcal{S}_1$, at least j constraints among the $2j$ inequalities are equalities. We show that these equalities are in fact all the UB(i) constraints:

Lemma 5. *In solution $\mathcal{S}_1$, all the constraints* UB(i), $\forall 1 \leq i \leq j$ *are equalities.*

Proof. We know from Lemma 4 that all the constraints LB(i), $\forall i < j$ are tight for an optimal solution. Therefore, if the constraint LB(j) is tight too for solution $\mathcal{S}_1$, then the property holds true.

Suppose now that the constraint LB(j) is not tight for solution $\mathcal{S}_1$, i.e. $\beta_j = 0$. Then, consider the other linear program corresponding to the case where only workers $P_1, \dots, P_{j-1}$ are used. Both linear program have the same optimal value. Reasoning with the other linear program just as we did previously, we know that none of the first $j-2$ workers may be kept fully idle during the processing. If there existed an optimal solution where P_{j-1} is fully idle, then it would be possible to derive a solution where workers $P_1, \dots, P_{j-2}, P_j$ would be in use and would process strictly more units of load. Thus, all the workers P_1 to P_{j-1} do process some units of load. Therefore, the vertex of the polyhedron $\mathcal{S}_1$ where $j-1$ constraints among $2(j-1)$ are equalities is such that all UB(i), $1 \leq i \leq j-1$, are equalities. Together with the second part of Lemma 4, we derive the desired result. □

We still have to prove that the result (simultaneous termination) is true for all optimal solutions, not just the extremal solution $\mathcal{S}_1$. Let $\mathcal{S}_2 = (\alpha_1, \dots, \alpha_j)$ denote another optimal solution, and suppose that some constraints UB(i) are tight for $i \leq j$. As already noticed, P_2 belongs to the convex polyhedron $\mathcal{P}$. Consider the following function f:

$$f : \left\{ \begin{array}{lll} \mathbb{R} & \rightarrow & \mathbb{R}^j \\ x & \mapsto & \mathcal{S}_1 + x(\mathcal{S}_2 - \mathcal{S}_1) \end{array} \right.$$

By construction, all the points $f(x)$ that belong to $\mathcal{P}$ are extremal solutions of the linear program. Let x_0 denote the largest value of $x \geq 1$ such that $f(x)$ does belong to $\mathcal{P}$, so that at least one of the constraints of the linear program is an equality in $f(x_0)$. This constraint has to be one of the LB(i) and, as a consequence of Lemma 4, it has to be LB(j). Therefore, we know that the j-th component of $f(x_0)$ is equal to zero and that the first $j-1$ components of $f(x_0)$, $\mathcal{S}'_2 = (\alpha'_1, \dots, \alpha'_{j-1})$, constitute an optimal solution of the linear program where only the first $j-1$ workers are involved. Therefore, if we denote by $\mathcal{S}'_1 = (\beta'_1, \ldots, \beta'_{j-1})$ an optimal solution obtained at some vertex, $(\beta'_1, \ldots, \beta'_{j-1}, 0)$ is a solution of the original linear program. Therefore, all the LB(i) are tight for $i < j$ (by Lemma 4); using the same reasoning as before, we know that the UB(i) are equalities in solution $\mathcal{S}'_1$ for $i < j$.

Suppose that $\mathcal{S}'_1 \neq \mathcal{S}'_2$. Consider the following function f':

$$f' : \left\{ \begin{array}{ccc} \mathbb{R} & \rightarrow & \mathbb{R}^{j-1} \\ x & \mapsto & \mathcal{S}'_1 + x(\mathcal{S}'_2 - \mathcal{S}'_1) \end{array} \right.$$

By construction, all the points $f'(x)$ that belong to $\mathcal{SP}$ are extremal solution of the linear program. Therefore, if we denote by x'_0 the largest value of $x \geq 1$ such that $f'(x)$ does belong to $\mathcal{P}$, so that at least one of the constraints of the linear program is not tight in $f'(x'_0)$. This constraint has to be one of the LB(i) and we know by Lemma 4 that it is not possible. Therefore, we have $\mathcal{S}'_1 = \mathcal{S}'_2$. Thus, as the UB$(i)$ are not tight at $\mathcal{S}'_1$ and at $\mathcal{S}'_2$ for $i < j$, they are not tight either at $f(x_0)$. Therefore, as the UB(i) are not tight at $\mathcal{S}_1$ and at $f(x_0)$ for $i \leq j$, they are not tight either at $\mathcal{S}_2$, which is in contradiction with our previous hypothesis.

Therefore, at $\mathcal{S}_2$, none of constraints UB(i), $\forall 1 \leq i \leq j$ is tight, which means that in an optimal solution, all participating workers terminate the execution at the same time. $\square$