Laboratoire de l'Informatique du Parallélisme

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

Comments on "Design and performance evaluation of load distribution strategies for multiple loads on heterogeneous linear daisy chain networks"

Matthieu Gallet ,
Yves Robert ,
Frédéric Vivien

February 2007

Research Report N° 2007-07

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`

Comments on "Design and performance evaluation of load distribution strategies for multiple loads on heterogeneous linear daisy chain networks"

Matthieu Gallet , Yves Robert , Frédéric Vivien

February 2007

Abstract

Min, Veeravalli, and Barlas proposed [8, 9] strategies to minimize the overall execution time of one or several divisible loads on a heterogeneous linear network, using one or more installments. We show on a very simple example that the approach proposed in [9] does not always produce a solution and that, when it does, the solution is often suboptimal. We also show how to find an optimal scheduling for any instance, once the number of installments per load is given. Finally, we formally prove that under a linear cost model, as in [8, 9], an optimal schedule has an infinite number of installments. Such a cost model can therefore not be used to design practical multi-installment strategies.

Keywords: scheduling, heterogeneous processors, divisible loads, single-installment, multiple-installments.

Résumé

Min, Veeravalli, and Barlas ont proposé [8, 9] des stratégies pour minimiser le temps d'exécution d'une ou de plusieurs tâches divisibles sur un réseau linéaire de processeurs hétérogènes, en distribuant le travail en une ou plusieurs tournées. Sur un exemple très simple nous montrons que l'approche proposée dans [9] ne produit pas toujours une solution et que, quand elle le fait, la solution est souvent sous-optimale. Nous montrons également comment trouver un ordonnancement optimal pour toute instance, quand le nombre de tournées par tâches est spécifié. Finalement, nous montrons formellement que lorsque les fonctions de coûts sont linéaires, comme c'est le cas dans [8, 9], un ordonnancement optimal a un nombre infini de tournées. Un tel modèle de coût ne peut donc pas être utilisé pour définir des stratégies en multi-tournées utilisables en pratique.

Mots-clés: ordonnancement, ressources hétérogènes, tâches divisibles, tournées.

1 Introduction

Min, Veeravalli and Barlas proposed [8, 9] strategies to minimize the overall execution time of one or several divisible loads on a heterogeneous linear network. Initially, the authors targeted single-installment strategies, that is strategies under which a processor receives in a single communication all its share of a given load. When they were not able to design single-installment strategies, they proposed multi-installment ones.

In this research note, we first show on a very simple example that the approach proposed in [9] does not always produce a solution and that, when it does, the solution is often suboptimal. The fundamental flaw of the approach of [9] is that the authors are optimizing the scheduling load by load, instead of attempting a global optimization. The load by load approach is suboptimal and overconstrains the problem.

On the contrary, we show how to find an optimal scheduling for any instance, once the number of installments per load is given. In particular, our approach always find the optimal solution in the single-installment case. Finally, we formally prove that under a linear cost model for communication and communication, as in [8, 9], an optimal schedule has an infinite number of installments. Such a cost model can therefore not be used to design practical multi-installment strategies.

Please refer to the papers [8, 9] for a detailed introduction to the optimization problem under study. We briefly recall the framework in Section 2, and we deal with an illustrative example in Section 3. Then we directly proceed to the design of our solution (Section 4), we discuss its possible extensions and the linear cost model (Section 5), before concluding (Section 6).

2 Problem and Notations

We summarize here the framework of [8, 9]. The target architecture is a linear chain of m processors $(P_1, P_2, \ldots, P_m)$. Processor P_i is connected to processor P_{i+1} by the communication link l_i (see Figure 1). The target application is composed of N loads, which are *divisible*, which means that each load can be split into an arbitrary number of chunks of any size, and these chunks can be processed independently. All the loads are initially available on processor P_1, which processes a fraction of them and delegates (sends) the remaining fraction to P_2. In turn, P_2 executes part of the load that it receives from P_1 and sends the rest to P_3, and so on along the processor chain. Communications can be overlapped with (independent) computations, but a given processor can be active in at most a single communication at any time-step: sends and receives are serialized (this is the full *one-port* model).

Since the last processor P_m cannot start computing before having received its first message, it is useful for P_1 to distribute the loads in several installments: the idle time of remote processors in the chain will be reduced due to the fact that communications are smaller in the first steps of the overall execution.

We deal with the general case in which the nth load is distributed in Q_n installments of different sizes. For the jth installment of load n, processor P_i takes a fraction $\gamma_j^n(i)$, and sends the remaining part to the next processor while processing its own fraction.

In the framework of [8, 9], loads have different characteristics. Every load n (with $1 \leq n \leq N$) is defined by a volume of data $V_{comm}(n)$ and a quantity of computation $V_{comp}(n)$. Moreover, processors and links are not identical either. We let w_i be the time taken by P_i

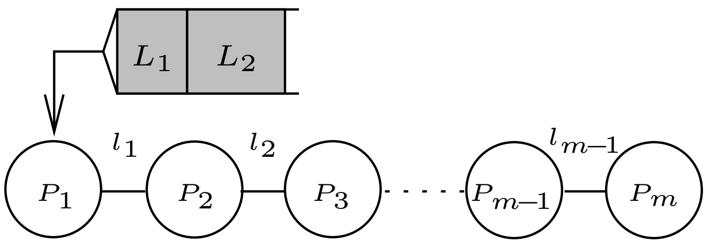

Figure 1: Linear network, with m processors and $m-1$ links.

m	Number of processors in the system.
P_i	Processor i, where $i = 1, \dots, m$.
w_i	Time taken by processor P_i to compute a unit load.
z_i	Time taken by P_i to transmit a unit load to P_{i+1}.
τ_i	Availability date of P_i (time at which it becomes available for processing the loads).
N	Total number of loads to process in the system.
Q_n	Total number of installments for nth load.
$V_{comm}(n)$	Volume of data for nth load.
$V_{comp}(n)$	Volume of computation for nth load.
$\gamma_i^j(n)$	Fraction of nth load computed on processor P_i during the jth installment.
$Comm_{i,n,j}^{start}$	Start time of communication from processor P_i to processor P_{i+1} for jth installment of nth load.
$Comm_{i,n,j}^{end}$	End time of communication from processor P_i to processor P_{i+1} for jth installment of nth load.
$Comp_{i,n,j}^{start}$	Start time of computation on processor P_i for jth installment of nth load.
$Comp_{i,n,j}^{end}$	End time of computation on processor P_i for jth installment of nth load.

Table 1: Summary of notations.

to compute a unit load ($1 \leq i \leq m$), and z_i be the time taken by P_i to send a unit load to P_{i+1} (over link l_i, $1 \leq i \leq m-1$). Note that we assume a linear model for computations and communications, as in the original articles, and as is often the case in divisible load literature [7, 4].

For the jth installment of the nth load, let $Comm_{i,n,j}^{start}$ denote the starting time of the communication between P_i and P_{i+1}, and let $Comm_{i,n,j}^{end}$ denote its completion time; similarly, $Comp_{i,n,j}^{start}$ denotes the start time of the computation on P_i for this installment, and $Comp_{i,n,j}^{end}$ denotes its completion time. The objective function is to minimize the *makespan*, i.e., the time at which all loads are computed. For the sake of convenience, all notations are summarized in Table 1.

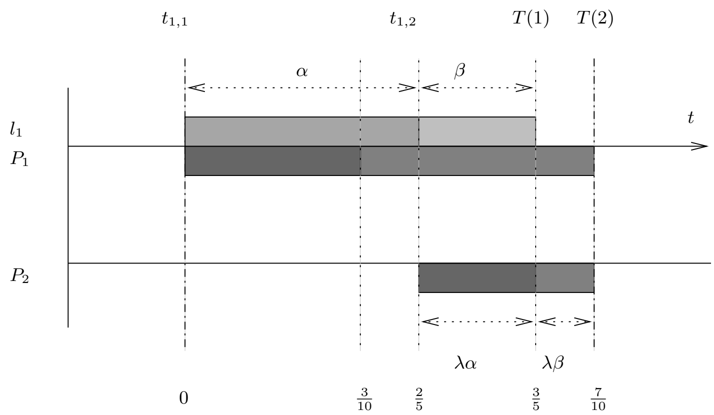

Figure 2: The example schedule, with $\lambda = \frac{1}{2}$, α is $\gamma_2^1(1)$ and β is $\gamma_2^1(2)$.

3 An illustrative example

3.1 Presentation

To show the limitations of [8, 9], we deal with a simple illustrative example. We use 2 identical processors P_1 and P_2 with $w_1 = w_2 = \lambda$, and $z(1) = 1$. We consider $N = 2$ identical divisible loads to process, with $V_{comm}(1) = V_{comm}(2) = 1$ and $V_{comp}(1) = V_{comp}(2) = 1$. Note that when λ is large, communications become negligible and each processor is expected to process around half of both loads. But when λ is close to 0, communications are very important, and the solution is not obvious. To ease the reading, we only give a short (intuitive) description of the schedules, and provide their different makespans without justification (we refer the reader to Appendix A for all proofs).

We first consider a simple schedule which uses a single installment for each load, as illustrated in Figure 2. Processor P_1 computes a fraction $\gamma_1^1(1) = \frac{2\lambda^2+1}{2\lambda^2+2\lambda+1}$ of the first load, and a fraction $\gamma_1^1(2) = \frac{2\lambda+1}{2\lambda^2+2\lambda+1}$ of the second load. Then the second processor computes a fraction $\gamma_2^1(1) = \frac{2\lambda}{2\lambda^2+2\lambda+1}$ of the first load, and a fraction $\gamma_2^1(2) = \frac{2\lambda^2}{2\lambda^2+2\lambda+1}$ of the second load. The makespan achieved by this schedule is equal to makespan$_1 = \frac{2\lambda\left(\lambda^2+\lambda+1\right)}{2\lambda^2+2\lambda+1}$.

3.2 Solution of [9], one-installment

In the solution of [9], P_1 and P_2 have to simultaneously complete the processing of their share of the first load. The same holds true for the second load. We are in the one-installment case when P_1 is fast enough to send the second load to P_2 while it is computing the first load. This condition writes $\lambda \geq \frac{\sqrt{3}+1}{2} \approx 1.366$.

In the solution of [9], P_1 processes a fraction $\gamma_1^1(1) = \frac{\lambda+1}{2\lambda+1}$ of the first load, and a fraction $\gamma_1^1(2) = \frac{1}{2}$ of the second one. P_2 processes a fraction $\gamma_2^1(1) = \frac{\lambda}{2\lambda+1}$ of the first load L_1, and a fraction $\gamma_2^1(2) = \frac{1}{2}$ of the second one. The makespan achieved by this schedule is makespan$_2 = \frac{\lambda(4\lambda+3)}{2(2\lambda+1)}$.

Comparing both makespans, we have $0 \leq \text{makespan}_2 - \text{makespan}_1 \leq \frac{1}{4}$, the solution of [9] having a strictly larger makespan, except when $\lambda = \frac{\sqrt{3}+1}{2}$. Intuitively, the solution of [9] is

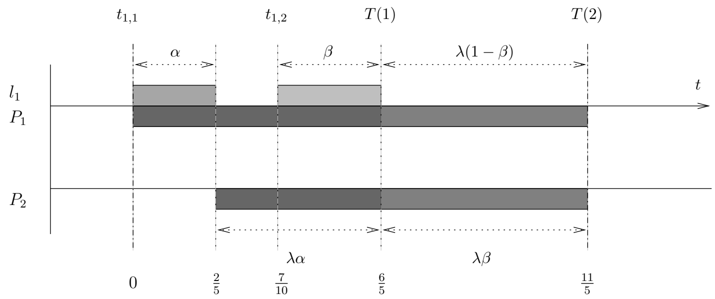

Figure 3: The schedule of [9] for $\lambda = 2$, with $\alpha = \gamma_2^1(1)$ and $\beta = \gamma_2^1(2)$.

worse than the schedule of Section 3.1 because it aims at locally optimizing the makespan for the first load, and then optimizing the makespan for the second one, instead of directly searching for a global optimum. A visual representation of this case is given in Figure 3 for $\lambda = 2$.

3.3 Solution of [9], multi-installment

The solution of [9] is a multi-installment strategy when $\lambda < \frac{\sqrt{3}+1}{2}$, i.e., when communications tend to be important compared to computations. More precisely, this case happens when P_1 does not have enough time to completely send the second load to P_2 before the end of the computation of the first load on both processors.

The way to proceed in [9] is to send the second load using a multi-installment strategy. Let Q denote the number of installments for this second load. We can easily compute the size of each fraction distributed to P_1 and P_2. Processor P_1 has to process a fraction $\gamma_1^1(1) = \frac{\lambda+1}{2\lambda+1}$ of the first load, and fractions $\gamma_1^1(2), \gamma_1^2(2), \dots, \gamma_1^Q(2)$ of the second one. Processor P_2 has a fraction $\gamma_2^1(1) = \frac{\lambda}{2\lambda+1}$ of the first load, and fractions $\gamma_2^1(2), \gamma_2^2(2), \dots, \gamma_2^Q(2)$ of the second one. Moreover, we have the following equality for $1 \leq k < Q$:

$$\gamma_1^k(2) = \gamma_2^k(2) = \lambda^k \gamma_2^1(1).$$

And for $k = Q$ (the last installment), we have $\gamma_1^Q(2) = \gamma_2^Q(2) \leq \lambda^Q \gamma_2^1(1)$. Let $\beta_k = \gamma_1^k(2) = \gamma_2^k(2)$. We can then establish an upper bound on the portion of the second load distributed in Q installments:

$$\sum_{k=1}^{Q} (2\beta_k) \leq 2 \sum_{k=1}^{Q} \left(\gamma_2^1(1) \lambda^k \right) = \frac{2\left(\lambda^Q - 1\right)\lambda^2}{2\lambda^2 - \lambda - 1}$$

if $\lambda \neq 1$, and $Q = 2$ otherwise.

We have three cases to discuss:

1. $0 < \lambda < \frac{\sqrt{17}+1}{8} \approx 0.64$: Since $\lambda < 1$, we can write for any nonnegative integer Q:

$$\sum_{k=1}^{Q} (2\beta_k) < \sum_{k=1}^{\infty} (2\beta_i) = \frac{2\lambda^2}{(1-\lambda)(2\lambda+1)}$$

   We have $\frac{2\lambda^2}{(1-\lambda)(2\lambda+1)} < 1$ for all $\lambda < \frac{\sqrt{17}+1}{8}$. So, even in the case of an infinite number of installments, the second load will not be completely processed. In other words, no

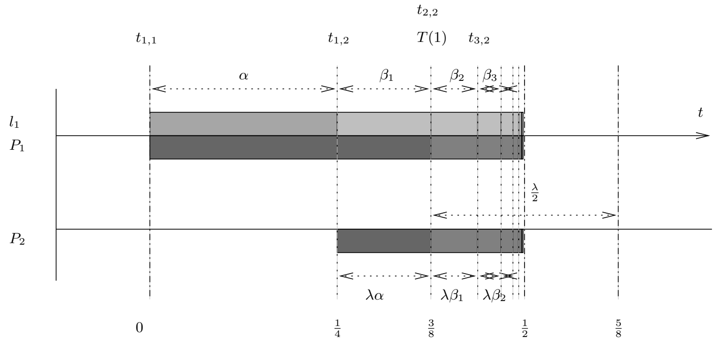

Figure 4: The example with $\lambda = \frac{1}{2}$, $\alpha = \gamma_2^1(1)$ and $\beta = \gamma_2^1(2)$.

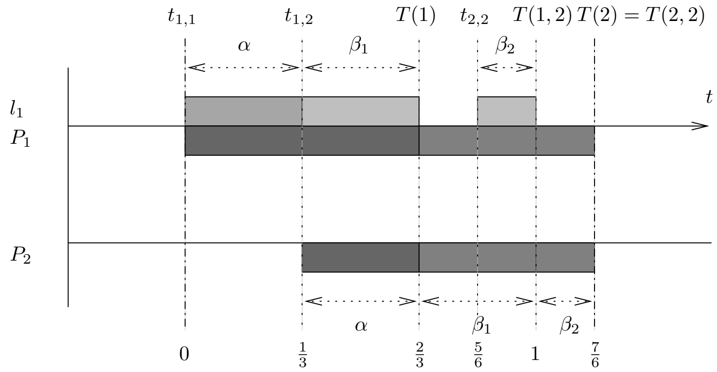

Figure 5: The example with $\lambda = 1$, $\alpha = \gamma_2^1(1)$ and $\beta = \gamma_2^1(2)$.

solution is found in [9] for this case. A visual representation of this case is given in Figure 4 with $\lambda = 0.5$.

2. $\lambda = \frac{\sqrt{17}+1}{8}$: We have $\frac{2\lambda^2}{(1-\lambda)(2\lambda+1)} = 1$, so an infinite number of installments is required to completely process the second load. Again, this solution is obviously not feasible.

3. $\frac{\sqrt{17}+1}{8} < \lambda < \frac{\sqrt{3}+1}{2}$: In this case, the solution of [9] is better than any solution using a single installment per load, but it may require a very large number of installments. A visual representation of this case is given in Figure 5 with $\lambda = 1$.

   In this case, the number of installments is set in [9] as $Q = \left\lceil \frac{\ln\left(\frac{4\lambda^2-\lambda-1}{2\lambda^2}\right)}{\ln(\lambda)} \right\rceil$. To see that this choice is not optimal, consider the case $\lambda = \frac{3}{4}$. The algorithm of [9] achieves a makespan equal to $\left(1 - \gamma_2^1(1)\right)\lambda + \frac{\lambda}{2} = \frac{9}{10}$. The first load is sent in one installment and the second one is sent in 3 installments (according to the previous equation).

   However, we can come up with a better schedule by splitting both loads into two installments, and distributing them as follows:

   - during the first round, P_1 processes 0 unit of the first load,
   - during the second round, P_1 processes $\frac{317}{653}$ unit of the first load,

- during the first round, P_2 processes $\frac{192}{653}$ unit of the first load,
- during the second round, P_2 processes $\frac{144}{653}$ unit of the first load,
- during the first round, P_1 processes 0 unit of the second load,
- during the second round, P_1 processes $\frac{464}{653}$ unit of the second load,
- during the first round, P_2 processes $\frac{108}{653}$ unit of the second load,
- during the second round, P_2 processes $\frac{81}{653}$ unit of the second load,

This scheme gives us a total makespan equal to $\frac{781}{653}\frac{3}{4} \approx 0.897$, which is (slightly) better than 0.9. This shows that among the schedules having a total number of four installments, the solution of [9] is suboptimal.

3.4 Conclusion

Despite its simplicity (two identical processors and two identical loads), the analysis of this illustrative example clearly outlines the limitations of the approach of [9]: this approach does not always return a feasible solution and, when it does, this solution is not always optimal. In the next section, we show how to compute an optimal schedule when dividing each load into any prescribed number of installments.

4 Optimal solution

We now show how to compute an optimal schedule, when dividing each load into any prescribed number of installments. Therefore, when this number of installment is set to 1 for each load (i.e., $Q_n = 1$, for any n in $[1, N]$), the following approach solves the problem originally target by Min, Veeravalli, and Barlas.

To build our solution we use a linear programming approach. In fact, we only have to list all the (linear) constraints that must be fulfilled by a schedule, and write that we want to minimize the makespan. All these constraints are captured by the linear program in Figure 6. The optimality of the solution comes from the fact that the constraints are exactly all the constraints a schedule must fulfill, and a solution to the linear program is obviously always feasible. This linear program simply encodes the following constraints (where a number in brackets is the number of the corresponding constraint on Figure 6):

- P_i cannot start a new communication to P_i before the end of the corresponding communication from P_{i-1} to P_i (1),
- P_i cannot start to receive the next installment of the nth load before having finished to send the current one to P_{i+1} (2),
- P_i cannot start to receive the first installment of the next load before having finished to send the last installment of the current load to P_{i+1} (3),
- any transfer has to begin at a nonnegative time (4),
- the duration of any transfer is equal to the product of the time taken to transmit a unit load (5) by the volume of data to transfer,

$$
\begin{array}{rrcll}
\forall i < m-1, n \leq N, j \leq Q_n & Comm^{start}_{i+1,n,j} & \geq & Comm^{end}_{i,n,j} & (1)\\
\forall i < m-1, n \leq N, j < Q_n & Comm^{start}_{i,n,j+1} & \geq & Comm^{end}_{i+1,n,j} & (2)\\
\forall i < m-1, n < N & Comm^{start}_{i,n+1,1} & \geq & Comm^{end}_{i+1,n,Q_n} & (3)\\
\forall i \leq m-1, n \leq N, j \leq Q_n & Comm^{start}_{i,n,j} & \geq & 0 & (4)\\
\forall i \leq m-1, n \leq N, j \leq Q_n & Comm^{end}_{i,n,j} & = & Comm^{start}_{i,n,j} + z_i V_{comm}(n) \sum_{k=i+1}^{m} \gamma_k^j(n) & (5)\\
\forall i \geq 2, n \leq N, j \leq Q_n & Comp^{start}_{i,n,j} & \geq & Comm^{end}_{i,n,j} & (6)\\
\forall i \leq m, n \leq N, j \leq Q_n & Comp^{end}_{i,n,j} & = & Comp^{start}_{i,n,j} + w_i \gamma_i^j(n) V_{calc}(n) & (7)\\
\forall i \leq m, n < N & Comp^{start}_{i,n+1,1} & \geq & Comp^{end}_{i,n,Q_n} & (8)\\
\forall i \leq m, n \leq N, j < Q_n & Comp^{start}_{i,n,j+1} & \geq & Comp^{end}_{i,n,j} & (9)\\
\forall i \leq m & Comp^{start}_{i,1,1} & \geq & \tau_i & (10)\\
\forall i \leq m, n \leq N, j \leq Q_n & \gamma_i^j(n) & \geq & 0 & (11)\\
\forall n \leq N & \sum_{i=1}^{m} \sum_{j=1}^{Q} \gamma_i^j(n) & = & 1 & (12)\\
\forall i \leq m & \text{makespan} & \geq & Comp^{end}_{i,N,Q} & (13)
\end{array}
$$

Figure 6: The complete linear program.

- processor P_i cannot start to compute the jth installment of the nth load before having finished to receive the corresponding data (6),
- the duration of any computation is equal to the product of the time taken to compute a unit load (7) by the volume of computations,
- processor P_i cannot start to compute the first installment of the next load before it has completed the computation of the last installment of the current load (8),
- processor P_i cannot start to compute the next installment of a load before it has completed the computation of the current installment of that load (9),
- processor P_i cannot start to compute the first installment of the first load before its availability date (10),
- every portion of a load dedicated to a processor is necessarily nonnegative (11),
- any load has to be completely processed (12),
- the makespan is no smaller than the completion time of the last installment of the last load on any processor (13).

Altogether, we have a linear program to be solved over the rationals, hence a solution in polynomial time [6]. In practice, standard packages like Maple [3] or GLPK [5] will return the optimal solution for all reasonable problem sizes.

Note that the linear program gives the optimal solution for a prescribed number of installments for each load. We will discuss the problem of the number of installments in the next section.

5 Possible extensions

There are several restrictions in the model of [9] that can be alleviated. First the model uses *uniform machines*, meaning that the speed of a processor does not depend on the task that it executes. It is easy to extend the linear program for unrelated parallel machines, introducing w_i^n to denote the time taken by P_i to process a unit load of type n. Also, all processors and loads are assumed to be available from the beginning. In our linear program, we have introduced availability dates for processors. The same way, we could have introduced release dates for loads. Furthermore, instead of minimizing the makespan, we could have targeted any other objective function which is an affine combination of the loads completion time and of the problem characteristics, like the average completion time, the maximum or average (weighted) flow, etc.

The formulation of the problem does not allow any piece of the n'th load to be processed before the nth load is completely processed, if $n' > n$. We can easily extend our solution to allow for N rounds of the N loads, each load being still divided into several installments. This would allow to interleave the processing of the different loads.

The divisible load model is linear, which causes major problems for multi-installment approaches. Indeed, once we have a way to find an optimal solution when the number of installments per load is given, the question is: what is the optimal number of installments? Under a linear model for communications and computations, the optimal number of installments is infinite, as the following theorem states:

Theorem 1. *Let us consider, under a linear cost model for communications and computations, an instance of our problem with one or more load and at least two processors. Then, any schedule using a finite number of installments is suboptimal for makespan minimization.*

This theorem is proved by building, from any schedule, another schedule with a strictly smaller makespan. The proof is available in Appendix B.

An infinite number of installments obviously does not define a feasible solution. Moreover, in practice, when the number of installments becomes too large, the model is inaccurate, as acknowledged in [2, p. 224 and 276]. Any communication incurs a startup cost K, which we express in bytes. Consider the nth load, whose communication volume is $V_{comm}(n)$: it is split into Q_n installments, and each installment requires $m-1$ communications. The ratio between the actual and estimated communication costs is roughly equal to $\rho = \frac{(m-1)Q_n K + V_{comm}(n)}{V_{comm}(n)} > 1$. Since K, m, and V_{comm} are known values, we can choose Q_n such that ρ is kept relatively small, and so such that the model remains valid for the target application. Another, and more accurate solution, would be to introduce latencies in the model, as in [1]. This latter article shows how to design asymptotically optimal multi-installment strategies for star networks. A similar approach should be used for linear networks.

6 Conclusion

We have shown that a linear programming approach allows to solve all instances of the scheduling problem addressed in [8, 9]. In contrast, the original approach was providing a solution only for particular problem instances. Moreover, the linear programming approach returns an optimal solution for any number of installments, while the original approach was empirically limited to very special strategies, and was often sub-optimal.

Intuitively, the solution of [9] is worse than the schedule of Section 3.1 because it aims at locally optimizing the makespan for the first load, and then optimizing the makespan for the second one, and so on, instead of directly searching for a global optimum. We did not find beautiful closed-form expressions defining optimal solutions but, through the power of linear programming, we were able to find an optimal schedule for any instance.

A Analytical computations for the illustrative example

In this appendix, we prove the results stated in Sections 3.2 and 3.3. In order to simplify equations, we write α instead of $\gamma_2^1(1)$ (i.e., α is the fraction of the first load sent from the first processor to the second one), and β instead of $\gamma_2^2(1)$ (similarly, β is the fraction of the second load sent to the second processor).

In this research note we used simpler notations than the ones used in [9]. However, as we want to explicit the solutions proposed by [9] for our example, we need to use the original notations to enable the reader to double-check our statements. The necessary notations from [9] are recalled in Table 2.

T_{cp}^n	Time taken by the standard processor ($w = 1$) to compute the load L_n.
T_{cm}^n	Time taken by the standard link ($z = 1$) to communicate the load L_n.
L_n	Size of the nth load, where $1 \leq n \leq N$.
$L_{k,n}$	Portion of the load L_n assigned to the kth installment for processing.
$\alpha_{n,i}^{(k)}$	The fraction of the total load $L_{k,n}$ to P_i, where $0 \leq \alpha_{n,i}^{(k)} \leq 1, \quad \forall i = 1, \ldots, m$ and $\sum_{i=1}^{m} \alpha_{n,i}^{(k)} = 1$.
$t_{k,n}$	The time instant at which is initiated the first communication for the kth installment of load L_n ($L_{k,n}$).
$C_{k,n}$	The total communication time of the kth installment of load L_n when $L_{k,n} = 1$; $C_{k,n} = \frac{T_{cm}^n}{L_n} \sum_{p=1}^{m-1} z_p \left(1 - \sum_{j=1}^{p} \alpha_{n,j}^{(k)}\right)$.
$E_{k,n}$	The total processing time of P_m for the kth installment of load L_n when $L_{k,n} = 1$; $E_{k,n} = \alpha_{n,m}^{(k)} w_m T_{cp}^n \frac{1}{L_n}$.
$T(k,n)$	The *finish time* of the kth installment of load L_n; it is defined as the time instant at which the processing of the kth installment of load L_n ends.
$T(n)$	The *finish time* of the load L_n; it is defined as the time instant at which the processing of the nth load ends, i.e., $T(n) = T(Q_n)$ where Q_n is the total number of installments required to finish processing load L_n. $T(N)$ is the finish time of the entire set of loads resident in P_1.

Table 2: Summary of the notations of [9] used in this paper.

In the solution of [9], both P_1 and P_2 have to finish the first load at the same time, and the same holds true for the second load. The transmission for the first load will take α time units, and the one for the second load β time units. Since P_1 (respectively P_2) will process the first load during $\lambda(1-\alpha)$ (respectively $\lambda\alpha$) time units and the second load during $\lambda(1-\beta)$ (respectively $\lambda\beta$) time units, we can write the following equations:

$$\begin{aligned} \lambda(1-\alpha) &= \alpha + \lambda\alpha \\ \lambda(1-\alpha) + \lambda(1-\beta) &= (\alpha + \max(\beta, \lambda\alpha)) + \lambda\beta \end{aligned} \tag{14}$$

There are two cases to discuss:

1. $\max(\beta, \lambda\alpha) = \lambda\alpha$. We are in the one-installment case when $L_2 C_{1,2} \leq T(1) - t_{1,2}$, i.e., $\beta \leq \lambda(1-\alpha) - \alpha$ (equation (5) in [9], where $L_2 = 1$, $C_{1,2} = \beta$, $T(1) = \lambda(1-\alpha)$ and $t_{1,2} = \alpha$). The values of α and β are given by:

$$\alpha = \frac{\lambda}{2\lambda+1} \qquad \text{and} \qquad \beta = \frac{1}{2}$$

   This case is true for $\lambda\alpha \geq \beta$, i.e., $\frac{\lambda^2}{2\lambda+1} \geq \frac{1}{2} \Leftrightarrow \lambda \geq \frac{1+\sqrt{3}}{2} \approx 1.366$.

   In this case, the makespan is equal to:

$$\text{makespan}_2 = \lambda(1-\alpha) + \lambda(1-\beta) = \frac{\lambda(4\lambda+3)}{2(2\lambda+1)}.$$

   Comparing both makespans, we have:

$$\text{makespan}_2 - \text{makespan}_1 = \frac{\lambda\left(2\lambda^2 - 2\lambda - 1\right)}{8\lambda^3 + 12\lambda^2 + 8\lambda + 2}.$$

   For all $\lambda \geq \frac{\sqrt{3}+1}{2} \approx 1.366$, our solution is better than their one, since:

$$\frac{1}{4} \geq \text{makespan}_2 - \text{makespan}_1 \geq 0$$

   Furthermore, the solution of [9] is strictly suboptimal for any $\lambda > \frac{\sqrt{3}+1}{2}$.

2. $\max(\beta, \lambda\alpha) = \beta$. In this case, P_1 does not have enough time to completely send the second load to P_2 before the end of the computation of the first load on both processors. The way to proceed in [9] is to send the second load using a multi-installment strategy.

   By using 14, we can compute the value of α:

$$\alpha = \frac{\lambda}{2\lambda+1}.$$

   Then we have $T(1) = (1-\alpha)\lambda = \frac{\lambda+1}{2\lambda+1}\lambda$ and $t_{1,2} = \alpha = \frac{\lambda}{2\lambda+1}$, i.e., the communication for the second request begins as soon as possible.

   We know from equation (1) of [9] that $\alpha_{2,1}^k = \alpha_{2,2}^k$, and by definition of the α's, $\alpha_{2,1}^k + \alpha_{2,2}^k = 1$, so we have $\alpha_{2,i}^k = \frac{1}{2}$. We also have $C_{1,2} = 1 - \alpha_{2,1}^k = \frac{1}{2}$, $E_{1,2} = \frac{\lambda}{2}$, $Y_{1,2}^{(1)} = 0$, $X_{1,2}^{(1)} = \frac{1}{2}$, $H = H(1) = \frac{X_{1,2}^{(1)} C_{1,2}}{C_{1,2}} = \frac{1}{2}$, $B = C_{1,2} + E_{1,2} - H = \frac{\lambda}{2}$.

   We will denote by $\beta_1, \ldots, \beta_n$ the sizes of the different installments processed on each processor (then we have $L_{k,2} = 2\beta_k$).

   Since the second processor is not left idle, and since the size of the first installment is such that the communication ends when P_2 completes the computation of the first load, we have $\beta_1 = T(1) - t_{1,2} = \lambda\alpha$ (see equation (27) in [9], in which we have $C_{1,2} = \frac{1}{2}$).

   By the same way, we have $\beta_2 = \lambda\beta_1$, $\beta_3 = \lambda\beta_2$, and so on (see equation (38) in [9], we recall that $B = \frac{\lambda}{2}$, and $C_{1,2} = \frac{1}{2}$):

$$\beta_k = \lambda^k \alpha$$

Each processor computes the same fraction of the second load. If we have Q installments, the total processed portion of the second load is upper bounded as follows:

$$\sum_{k=1}^{Q}(2\beta_k) \leq 2\sum_{k=1}^{Q}\left(\alpha\lambda^k\right) = 2\frac{\lambda}{2\lambda+1}\lambda\frac{\lambda^Q-1}{\lambda-1} = \frac{2\left(\lambda^Q-1\right)\lambda^2}{2\lambda^2-\lambda-1}$$

if $\lambda \neq 1$, and $Q = 2$ otherwise.

$$\sum_{k=1}^{Q}(2\beta_k) \leq \frac{2\lambda^2 Q}{2\lambda+1}.$$

We have four sub-cases to discuss:

(a) $0 < \lambda < \frac{\sqrt{17}+1}{8} \approx 0.64$: Since $\lambda < 1$, we can write for any nonnegative integer Q:

$$\sum_{k=1}^{Q}(2\beta_k) < \sum_{k=1}^{\infty}(2\beta_k) = \frac{2\lambda^2}{(1-\lambda)(2\lambda+1)}$$

We have $\frac{2\lambda^2}{(1-\lambda)(2\lambda+1)} < 1$ for all $\lambda < \frac{\sqrt{17}+1}{8}$. So, even in the case of an infinite number of installments, the second load will not be completely processed. In other words, no solution is found in [9] for this case.

(b) $\lambda = \frac{\sqrt{17}+1}{8}$: We have $\frac{2\lambda^2}{(1-\lambda)(2\lambda+1)} = 1$, so an infinite number of installments is required to completely process the second load. Again, this solution is obviously not feasible.

(c) $\frac{\sqrt{17}+1}{8} < \lambda < \frac{\sqrt{3}+1}{2}$ and $\lambda \neq 1$: In this case, the solution of [9] is better than any solution using a single installment per load, but it may require a very large number of installments.

Now, let us compute the number of installments. We know that the ith installment is equal to $\beta_i = \lambda^i\gamma_2^1(1)$, excepting the last one, which can be smaller than $\lambda^Q\gamma_2^1(1)$. So, instead of writing $\sum_{i=1}^{Q} 2\beta_i = \left(\sum_{i}^{Q-1} 2\lambda^i\gamma_2^1(1)\right) + 2\beta_Q = 1$, we write:

$$\sum_{i=1}^{Q} 2\lambda^i\gamma_2^1(1) \geq 1 \Leftrightarrow \frac{2\lambda^2\left(\lambda^Q-1\right)}{(\lambda-1)(2\lambda+1)} \geq 1 \Leftrightarrow \frac{2\lambda^{Q+2}}{(\lambda-1)(2\lambda+1)} \geq \frac{2\lambda^2}{(\lambda-1)(2\lambda+1)}+1.$$

If λ is strictly smaller than 1, we obtain:

$$\begin{aligned} & \tfrac{2\lambda^{Q+2}}{(\lambda-1)(2\lambda+1)} \geq \tfrac{2\lambda^2}{(\lambda-1)(2\lambda+1)}+1 \quad \Leftrightarrow \quad 2\lambda^{Q+2} \leq 4\lambda^2-\lambda-1 \\ \Leftrightarrow \quad & \ln(\lambda^Q) \leq \ln\left(\tfrac{4\lambda^2-\lambda-1}{2\lambda^2}\right) \quad \Leftrightarrow \quad Q\ln(\lambda) \leq \ln\left(\tfrac{4\lambda^2-\lambda-1}{2\lambda^2}\right) \\ \Leftrightarrow \quad & Q \geq \frac{\ln\left(\frac{4\lambda^2-\lambda-1}{2\lambda^2}\right)}{\ln(\lambda)} \end{aligned}$$

We thus obtain:

$$Q = \left\lceil \frac{\ln\left(\frac{4\lambda^2-\lambda-1}{2\lambda^2}\right)}{\ln(\lambda)} \right\rceil.$$

When λ is strictly greater than 1 we obtain the exact same result (then $\lambda-1$ and $\ln(\lambda)$ are both positive).

(d) $\lambda = 1$. In this case,

$$\sum_{i=1}^{Q} 2\lambda^i \gamma_2^1(1) \geq 1$$

simply leads to $Q = 2$.

B Proof of Theorem 1

Proof. We first remark that in any optimal solution to our problem all processors work and complete their share simultaneously. To prove this statement, we consider a schedule where one processor completes its share strictly before the makespan (this processor may not be doing any work at all). Then, under this schedule there exists two neighbor processors, P_i and P_{i+1}, such that one finishes at the makespan, denoted $\mathcal{M}$, and one strictly earlier. We have two cases to consider:

1. There exists a processor P_i which finishes strictly before the makespan $\mathcal{M}$ and such that the processor P_{i+1} completes its share exactly at time $\mathcal{M}$. P_{i+1} receives all the data it processes from P_i. We consider any installment j of any load L_n that is effectively processed by P_{i+1} (that is, P_{i+1} processes a non null portion of the jth installment of load L_n). We modify the schedule as follows: P_i enlarges by an amount ϵ, and P_{i+1} decreases by an amount ϵ, the portion of the jth installment of the load L_n it processes. Then, the completion time of P_i is increased, and that of P_{i+1} is decreased, by an amount proportional to ϵ as our cost model is linear. If ϵ is small enough, both processors complete their work strictly before $\mathcal{M}$. With our modification of the schedule, the size of a single communication was modified, and this size was decreased. Therefore, this modification did not enlarge the completion time of any processor except P_i. Therefore, the number of processors whose completion time is equal to $\mathcal{M}$ is decreased by at least one by our schedule modification.

2. No processor which completes it share strictly before time $\mathcal{M}$ is followed by a processor finishing at time $\mathcal{M}$. Therefore, there exists an index i such that the processors P_1 through P_i all complete their share exactly at $\mathcal{M}$, and the processors P_{i+1} through P_m complete their share strictly earlier. Then, let the last data to be effectively processed by P_i be a portion of the jth installment of the load L_n. Then P_i decreases by a size ϵ, and P_{i+1} increases by a size ϵ, the portion of the jth installment of load L_n that it processes. Then the completion time of P_i is decreased by an amount proportional to ϵ and the completion time of the processors P_{i+1} through P_m is increased by an amount proportional to ϵ. Therefore, if ϵ is small enough, the processors P_i through P_m complete their work strictly before $\mathcal{M}$.

In both cases, after we modified the schedule, there is at least one more processor which completes its work strictly before time $\mathcal{M}$, and no processor is completing its share after that time. If no processor is any longer completing its share at time $\mathcal{M}$, we have obtained a schedule with a better makespan. Otherwise, we just iterate our process. As the number of processors is finite, we will eventually end up with a schedule whose makespan is strictly smaller than $\mathcal{M}$. Hence, in an optimal schedule all processors complete their work simultaneously (and thus all processors work).

We now prove the theorem itself by contradiction. Let $\mathcal{S}$ be any optimal schedule using a finite number of installments. As processors P_2 through P_m initially hold no data, they stay temporarily idle during the schedule execution, waiting to receive some data to be able to process them. Let us consider processor P_2. As the idleness of P_2 is only temporary (all processors are working in an optimal solution), this processor is only idle because it is lacking data to process and it is waiting for some. Therefore, the last moment at which P_2 stays temporarily idle under $\mathcal{S}$ is the moment it finished to receive some data, namely the jth installment of load L_n sent to him by processor P_1.

As previously, Q_k is the number of installments of the load L_k under $\mathcal{S}$. Then from the schedule $\mathcal{S}$ we build a schedule $\mathcal{S}'$ by dividing in two identical halves the jth installment of load L_n. Formally:

- All loads except L_n have the exact same installments under $\mathcal{S}'$ than under $\mathcal{S}$.
- The load L_n has $(1+Q_n)$ installments under $\mathcal{S}'$, defined as follows.
- The first $(j-1)$ installments of L_n under $\mathcal{S}'$ are identical to the first $(j-1)$ installments of this load under $\mathcal{S}$.
- The jth and $(j+1)$th installment of L_n under $\mathcal{S}'$ are identical to the jth installment of L_n under $\mathcal{S}$, except that all sizes are halved.
- The last (Q_n-j) installments of L_n under $\mathcal{S}'$ are identical to the last (Q_n-j) installments of this load under $\mathcal{S}$.

We must first remark that no completion time is increased by the transformation from $\mathcal{S}$ to $\mathcal{S}'$. Therefore the makespan of $\mathcal{S}'$ is no greater than the makespan of $\mathcal{S}$. We denote by $Comm^{start}_{1,n,j}$ (respectively $Comm^{end}_{1,n,j}$) the time at which processor P_1 starts (resp. finishes) sending to processor P_2 the jth installment of load L_n under $\mathcal{S}$. We denote by $Comp^{start}_{2,n,j}$ (respectively $Comp^{end}_{2,n,j}$) the time at which processor P_2 starts (resp. finishes) computing the jth installment of load L_n under $\mathcal{S}$. We use similar notations, with an added prime, for schedule $\mathcal{S}'$. One can then easily derive the following properties:

$$Comm'^{\,start}_{1,n,j} = Comm^{start}_{1,n,j}. \tag{15}$$

$$Comm'^{\,start}_{1,n,j+1} = Comm'^{\,end}_{1,n,j} = \frac{Comm^{start}_{1,n,j} + Comm^{end}_{1,n,j}}{2}. \tag{16}$$

$$Comm'^{\,end}_{1,n,j+1} = Comm^{end}_{1,n,j}. \tag{17}$$

$$Comp'^{\,start}_{2,n,j} = Comm'^{\,end}_{1,n,j}. \tag{18}$$

$$Comp'^{\,end}_{2,n,j} = Comm'^{\,end}_{1,n,j} + \frac{Comp^{end}_{2,n,j} - Comp^{start}_{2,n,j}}{2}. \tag{19}$$

$$Comp'^{\,start}_{2,n,j+1} = \max\{Comp'^{\,end}_{2,n,j}, Comm'^{\,end}_{1,n,j+1}\}. \tag{20}$$

$$Comp'^{\,end}_{2,n,j} = Comp'^{\,start}_{2,n,j+1} + \frac{Comp^{end}_{2,n,j} - Comp^{start}_{2,n,j}}{2}. \tag{21}$$

Using equations 16, 17, 19, 20, and 21 we then establish that:

$$Comp'^{\,end}_{2,n,j} = \max\left\{ \frac{Comm^{start}_{1,n,j} + Comm^{end}_{1,n,j}}{2} + Comp^{end}_{2,n,j} - Comp^{start}_{2,n,j}, \right.$$
$$\left. Comm^{end}_{1,n,j} + \frac{Comp^{end}_{2,n,j} - Comp^{start}_{2,n,j}}{2} \right\}.$$

Therefore, under schedule $\mathcal{S}'$ processor P_2 completes strictly earlier than under $\mathcal{S}$ the computation of what was the j installment of load L_n under $\mathcal{S}$. If P_2 is no more idle after the time $Comp'^{\,end}_{2,n,j}$, then it completes its overall work strictly earlier under $\mathcal{S}'$ than under $\mathcal{S}$. On the other hand, P_1 completes its work at the same time. Then, using the fact that in an optimal solution all processors finish simultaneously, we conclude that $\mathcal{S}'$ is not optimal. As we have already remarked that its makespan is no greater than the makespan of $\mathcal{S}$, we end up with the contradiction that $\mathcal{S}$ is not optimal. Therefore, P_2 must be idled at some time after the time $Comp'^{\,end}_{2,n,j}$. Then we apply to $\mathcal{S}'$ the transformation we applied to $\mathcal{S}$ as many times as needed to obtain a contradiction. This process is bounded as the number of communications that processor P_2 receives after the time it is idled for the last time is strictly decreasing when we transform the schedule $\mathcal{S}$ into the schedule $\mathcal{S}'$. □

References

[1] Olivier Beaumont, Henri Casanova, Arnaud Legrand, Yves Robert, and Yang Yang. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Trans. Parallel Distributed Systems*, 16(3):207–218, 2005.

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

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

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

[5] GLPK: GNU Linear Programming Kit. http://www.gnu.org/software/glpk/.

[6] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th ACM symposium on Theory of Computing (STOC'84)*, pages 302–311. ACM Press, 1984.

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

[8] Han Min Wong and Bharadwaj Veeravalli. Scheduling divisible loads on heterogeneous linear daisy chain networks with arbitrary processor release times. *IEEE Trans. Parallel Distributed Systems*, 15(3):273–288, 2004.

[9] Han Min Wong, Bharadwaj Veeravalli, and Gerassimos Barlas. Design and performance evaluation of load distribution strategies for multiple divisible loads on heterogeneous linear daisy chain networks. *J. Parallel Distributed Computing*, 65(12):1558–1577, 2005.