New Heuristics and Integer Programming Formulations for Scheduling Divisible Load Tasks

Elbio R. T. Abib and Celso C. Ribeiro

Abstract— **Divisible load applications occur in many fields of science and engineering. They can be parallelized in a master-worker fashion, but they pose several scheduling challenges. We propose single-round and multi-round integer programming formulations for scheduling divisible load applications with minimum makespan. An innovative linear-time exact algorithm improving the complexity of the best known algorithm to date is described for the special case in which the processor activation order is known beforehand. This algorithm is embedded within a greedy-with-feedback heuristic for finding good solutions for single-round problems. Numerical results illustrate the speed and the effectiveness of the proposed heuristic.**

I. Motivation

The divisible load model was introduced by Cheng and Robertazzi [9] and has been studied and spread, see e.g. Bharadwaj et al. [4]. Divisible loads can be split into arbitrarily many tasks or fractions that can be easily processed in parallel without precedence restrictions, following the master-worker paradigm. We assume that the load may be split continuously, since a high data granularity is assumed. Many applications in science and engineering fit this model. As an example, we cite the search for the best matching of a given image with those in a database, when the size of each single image is much smaller than that of the entire database itself. Each image in the database can be sent to a processor that performs its matching with the given image. All processors can work in parallel, without precedence constraints. Other problems in video and image processing [13], linear algebra [5], simulation [1], and other science and engineering problems [8] can also be handled as divisible load applications. The robustness of the divisible load model was addressed in [11].

We propose new algorithms and integer programming formulation for scheduling divisible load applications, for which numerical results are reported. We assume that the divisible load applications run in dedicated grids with a star structure, in which one specific processor is always in charge of distributing the load to all others taking part in the computations. The system model and the problem formulation are described in detail in Section II. Related work is reviewed in Section III. An integer programming model for the single-round problem is presented in Section IV. A low complexity exact algorithm for a special case is presented in Section V. A fast feedback heuristic is proposed in Section VI. Numerical results are reported in Section VII. An integer programming model for the multi-round problem is presented in Section VIII. Concluding remarks are drawn in the last section.

Elbio R. T. Abib is now with Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, USA. This work was supported by a CNPq scholarship and developed while he was with União Pioneira de Integração Social, Department of Informatics, SEPS 712/912, Conjunto A, Asa Sul, Brasília, DF 70390-125, Brazil (e-mail: elbio.abib@microsoft.com).

Celso C. Ribeiro is now with the Department of Computer Science, Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil (e-mail: celso@ic.uff.br). Work of this author was supported by CNPq research grants 301.694/2007-9 and 485.328/2007-0, and by FAPERJ research grant E-152.522/2006.

II. System model and problem formulation

We denote by a processor a single processing unit, with its own memory, disk, and network device. Terms like data and load are used interchangeably in this work. We assume a dedicated cluster is running a parallel application following the master/worker model [3], [15]. There are n worker processors P_1, P_2, $\ldots$, P_n and one master processor P_0. At the beginning, the master owns the total load W to be processed. It splits the load into chunks and send them to the workers, but does not take part in the computations. The interconnection topology is a star, composed of n links connecting the master P_0 to the workers. There is no communication overlap: the master is able to send data to at most one processor at a time, without concurrency.

We denote by $G_i > 0$ the time needed to send one unit of data from the master to processor P_i, for every $i = 1, \ldots, n$. The latency (or startup time) g_i is the time needed to initiate the communication with processor P_i (involving e.g. the establishment of a TCP connection and the authentication of the master). Furthermore, we denote by $w_i > 0$ the time needed to process one unit of data at P_i.

The model assumes concurrency between communication and computation: any given processor may start receiving a new chunk of data while it is still processing the previous chunk. However, one processor may only execute one chunk at a time and it can only start processing a new chunk after receiving it entirely.

In a *single-installment* scheduling, only one chunk of data is sent to each processor. Let α_i be the load sent to processor P_i, for $i = 1, \ldots, n$, with $\sum_{i=1}^{i=n} \alpha_i = W$. Then, $g_i + G_i\alpha_i$ is the total time needed to send the load to be processed at P_i. Execution at processor P_i starts once all data has been received and lasts for $w_i\alpha_i$ time units. We define the *makespan* as the total elapsed time since the master began to send data to the first processor, until the last processor stops its computations.

The single-installment scheduling problem consists in determining the processors to be used, the order in which data should be sent to them (i.e., their activation order), and the

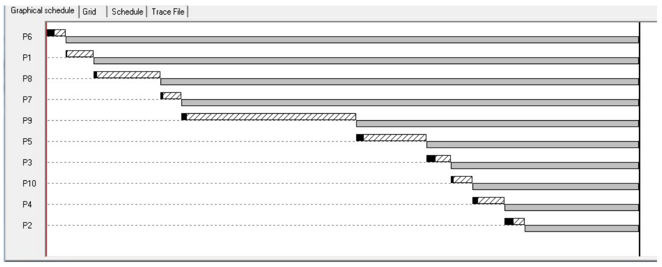

Fig. 1. Optimal single-installment schedule.

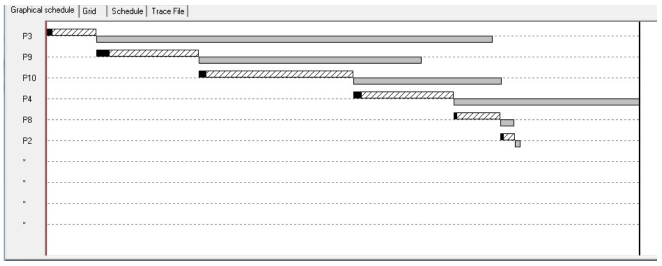

Fig. 2. Non-optimal single-installment schedule.

amount of load to be sent to each of them so as to minimize the makespan.

Figure 1 depicts an optimal schedule in which the worker processors P_6, P_1, P_8, P_7, P_9, P_5, P_3, P_{10}, P_4, and P_2 are activated in this order. Black bars indicate latencies, hatched bars represent data transfer periods, and gray bars correspond to processing periods at each worker. Each processor taking part of the computation receives exactly one chunk of data. All processors stop their computations at the same time and the solution is optimal. Figure 2 displays a non-optimal schedule for the worker processors P_3, P_9, P_{10}, P_4, P_8, and P_2 activated in this order.

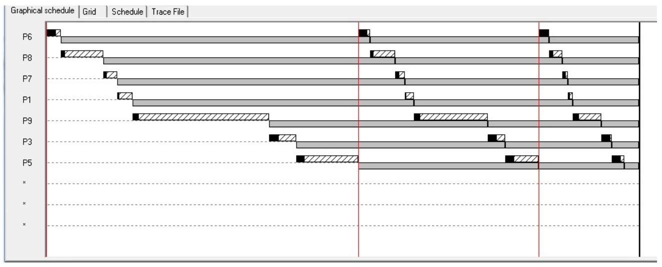

Fig. 3. Three-installment schedule.

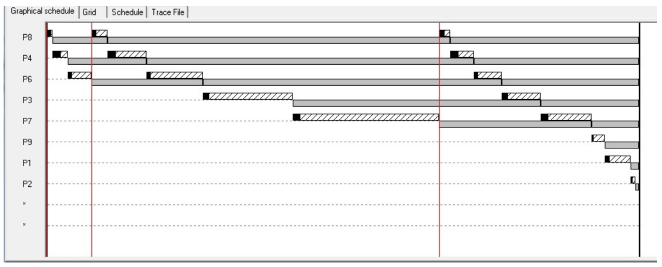

Fig. 4. More flexible three-installment schedule.

In a *multi-installment scheduling*, several chunks of data may be sent to each processor. The schedule is organized into multiple rounds (or installments) of transmission, following the same activation order [15], [16], [17]. Multi-installment schedules may reduce the makespan by decreasing the idle times, at the additional cost of repeated latency periods. The plot in Figure 3 illustrates a three-round schedule where all processors receive data in all rounds. In this work, we propose a more flexible and generic formulation allowing the use of a different number of processors in each round, what may lead to better solutions as shown in Figure 4.

III. Related work

The divisible load model was first introduced by Cheng and Robertazzi [9] to address computation and communication costs in networks of intelligent sensors. Afterwards, it was generalized and applied to different interconnection topologies, such as buses [4], star networks [4], and hypercubes [6]. The star topology is the most widely considered, since it seems to be the more appropriate to represent current interconnection technologies in clusters and local networks.

Latency times were not considered in [4], but have been addressed in more recent models such as [2], [3], [7], [12], [14], [16].

The use of communication costs was generalized in [7] by fixed startup times, also considered in [2], [3], [14]. Other models of communication and computation latencies were considered in [16]. In this work, we adopted the same model used in [2], [3], [7], [14], since the additional complexity introduced in the solution of the model proposed in [16] does not contribute too much to its robustness.

Two different types of scenarios have been considered in [3], [4], depending on whether or not the workers perform computations while they are receiving data from the master. The model that allows the superposition of communications and computations is more used in the literature and was adopted in this work, since it is more close to the actual characteristics of real-life computing systems.

Bharadwaj et al. [4] described simple algorithms to determine optimal schedules for both homogeneous and heterogeneous systems when there are no communication latencies (i.e., $g_i = 0$ for $i = 1, \dots, n$). Beaumont et al. [3] and Blazewicz and Drosdowski [7] reported exact algorithms for special cases considering communication latencies. A nonlinear integer programming formulation was reported in [10] and improved in [14] for the case where there are memory constraints on the processors.

In addition to the variables considered in the single-installment case, multi-installment schedules should also determine the total number of communication rounds, their durations, and which processors will be activated in each round. For the sake of simplicity, the same activation order is used in all installments, but the number of processors used in each round can vary from one round to another. For the special case where the number of processors is fixed and equal in all rounds, heuristics were given in [3], [4], [17].

IV. Single-installment model

Approximate non-linear integer programming models for the single-installment divisible load scheduling problem can be found in [2], [10], [14]. The first model presented in [10] may not even give a solution if there is not enough time to send data to all processors and make them finish processing at the same time. The models in [2], [14] addressed this drawback, but remain difficult to be solved.

We propose an innovative linear integer programming formulation for the single-installment problem. Let the binary variable x_{ij} be equal to 1 if processor P_i is the j^{th} to be activated and receive data, 0 otherwise. Moreover, let $\alpha_{ij} > 0$ be the amount of data sent to processor P_i if it is the j^{th} to be activated ($\alpha_{ij} = 0$ otherwise), t_j be the time instant in which the j^{th} processor to be activated starts receiving data, and T be the makespan. Model 1 below is an appropriate formulation for the single-installment divisible load scheduling problem.

Model 1 Single-installment mixed integer program

$$T^* = \text{ minimum } T \tag{1}$$

subject to:

$$\sum_{i=1}^{n} x_{ij} \leq 1 \qquad j = 1, \dots, n \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} \leq 1 \qquad i = 1, \dots, n \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} \geq \sum_{i=1}^{n} x_{i,j+1} \qquad j = 1, \dots, n-1 \tag{4}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{ij} = W \tag{5}$$

$$\alpha_{ij} \leq W x_{ij} \qquad i, j = 1, \dots, n \tag{6}$$

$$t_1 = 0 \tag{7}$$

$$t_j \geq t_{j-1} + \sum_{i=1}^{n} (g_i x_{i,j-1} + G_i \alpha_{i,j-1}) \qquad j = 2, \dots, n \tag{8}$$

$$t_j + \sum_{i=1}^{n} (g_i x_{ij} + (G_i + w_i)\alpha_{ij}) = T \qquad j = 1, \dots, n \tag{9}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \dots, n \tag{10}$$

$$\alpha_{ij} \geq 0 \qquad i, j = 1, \dots, n. \tag{11}$$

Constraints (2) imply that at most one processor can be the j^{th} to receive data. Constraints (3) ensure that each processor can be activated at most once. Constraints (4) guarantee that there must be j previously activated processors when the $(j+1)^{th}$ is activated. Constraint (5) ensures that the total load W is divided over the processors.

Constraints (6) ensure that processor i can only be the j^{th} to receive data if it was chosen to be the j^{th} in the activation order. Constraint (7) establishes that the first processor to be activated will start to receive data at instant zero. Constraints (8) are used to enforce that the j^{th} processor to be activated will start receiving data after the previous processor in the activation order finishes receiving its data. Constraints (9) imply that the makespan T is equal to the time t_j in which any processor P_j starts receiving its data plus the time $\sum_{i=1}^{n} g_i x_{ij} + (G_i + w_i)\alpha_{ij}$ it needs to receive and process its data, since in any optimal solution all processors finish at the same time [7].

This formulation improves and extends that in [2]. Constraints (9) remove the nonlinearities in the previous formulation, while constraints (4) improve solution consistency.

V. Linear-time algorithm for a given activation order

In this section, we describe a linear-time algorithm for the special case in which the processor activation order is known beforehand. In this case, the scheduling problem consists exclusively in computing the loads to be sent to each processor. The new algorithm improves upon the $O(n \log n)$ algorithm in [7].

Without loss of generality and for easiness of notation, we assume that processor P_i is the i-th to be activated. We denote by α_i^* the optimal amount of data to be sent to processor P_i and we define $f_i = (w_i + G_i)/w_{i-1}$, for $i = 1, \dots, n$.

A. Feasible solutions with a given number of processors

We first suppose that the number ℓ of processors to be used is also known. In this case, Blazewicz and Drozdowski [7] established that the solution to the system

$$\alpha_k w_k = g_{k+1} + \alpha_{k+1}(w_{k+1} + G_{k+1}), \quad k = 1, \dots, \ell - 1 \tag{12}$$

$$\sum_{k=1}^{\ell} \alpha_k = W \tag{13}$$

gives the optimal loads:

$$\alpha_k = \alpha_\ell \prod_{j=k+1}^{\ell} f_j + \sum_{j=k+1}^{\ell} \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i\right), \quad k = 1, \dots, \ell - 1 \tag{14}$$

$$\alpha_\ell = \frac{W - \sum_{k=1}^{\ell-1} \sum_{j=k+1}^{\ell} \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i\right)}{1 + \sum_{k=1}^{\ell-1} \prod_{j=k+1}^{\ell} f_j} \tag{15}$$

These values yield a feasible solution if $\alpha_k \geq 0$, for every $k = 1, \dots, \ell$. Equations (12) imply that the products $\alpha_i w_i$ are non-increasing for $i = 1, \dots, \ell$, since all constants G_i, w_i, and g_i are non-negative. Therefore, $\alpha_i \geq 0$ if and only if $\alpha_\ell \geq 0$, for $i = 1, \dots, \ell$. Finally, we conclude from equation (15) that the above solution is feasible if and only if $V(\ell) = \sum_{k=1}^{\ell-1} \sum_{j=k+1}^{\ell} \left(\frac{g_j}{w_{j-1}} \prod_{i=k+1}^{j-1} f_i\right) \leq W$.

B. Optimal number of processors

We now investigate the exact number ℓ^* of processors that should be activated in an optimal solution.

The term $V(\ell)$ appearing in the numerator of equation (15) may be recursively defined by

$$V(\ell) \quad = \quad \frac{g_\ell}{w_{\ell-1}} \sum_{k=1}^{\ell-1} \prod_{i=k+1}^{\ell-1} f_i + V(\ell-1). \qquad (16)$$

For any activation order, $V(\ell)$ is a non-decreasing function of the number ℓ of activated processors. A solution is infeasible if $V(\ell) > W$. Let the function $F(\ell) = \sum_{k=1}^{\ell-1} \prod_{i=k+1}^{\ell-1} f_i$ be recursively defined by

$$F(1) = 0, F(2) = 1, \text{ and } F(\ell) = 1 + F(\ell-1) f_{\ell-1}.$$

Therefore, $V(1) = 0$ and for any $\ell > 1$

$$V(\ell) \quad = \quad \frac{g_\ell}{w_{\ell-1}} F(\ell) + V(\ell-1) \qquad (17)$$

may be computed in time O(1) from $V(\ell-1)$ and $F(\ell)$.

Let us assume that a feasible solution exists for r processors and suppose that one additional processor is made available. If a feasible solution satisfying constraints (12-13) still exists, then some load will be transferred from one of the original r processors to the new processor taking part in the computations. In consequence, the loads assigned to the other processors will be decreased and the makespan will be reduced. Therefore, the optimal solution (i.e., that with minimum makespan) will have as many processors as possible to achieve feasibility.

C. Linear-time algorithm

Given a fixed activation order, the algorithm starts by sending all load exclusively to the first processor. Next, the number ℓ of processors is iteratively increased from 1 to n, until $V(\ell)$ turns out to be greater than W. Then, the optimal number of processors is set as $\ell^* = \ell - 1$.

Once the optimal number ℓ^* of processors has been computed, the load α_{ℓ^*} sent to the last processor is computed from equation (15). The other loads α_i, for $i = 1, \ldots, \ell^* - 1$, are recursively computed from ℓ^* using equation (14). Algorithm 1 implements the above computations in time $O(n)$.

Besides the number of processors and all their related data, this algorithm takes as input a vector π describing the activation order, such that $\pi(i) = j$ indicates that processor P_j is the i-th to be activated, for $i, j = 1, \ldots, n$. For instance, if $n = 3$ and $\pi =< 2, 3, 1 >$, then P_2 is the first processor to be activated, P_3 is the second, and P_1 is the third.

Algorithm 1 Linear-time algorithm for a fixed order

```
Require: Vector $\pi$ establishing the activation order
Ensure: Minimum makespan $T^*$, number $\ell^*$ of processors,
    loads $\alpha_i$ for $i = 1, \ldots, n$
 1: $F[1] \leftarrow 0$
 2: $F[2] \leftarrow 1$
 3: for $i \leftarrow 3$ to $n$ do
 4:     $F[i] \leftarrow 1 + F[i-1] \cdot (w_{\pi[i-1]} + G_{\pi[i-1]})/w_{\pi[i-2]}$
 5: end for
 6: $V[1] \leftarrow 0$
 7: for $i \leftarrow 2$ to $n$ do
 8:     $V[i] \leftarrow (g_{\pi[i]}/w_{\pi[i-1]}) \cdot F[i] + V[i-1]$
 9: end for
10: for $\ell \leftarrow 1$ to $n$ do
11:     if $V[\ell] \leq W$ then
12:         $\ell^* \leftarrow \ell$
13:     end if
14: end for
15: $numerator \leftarrow W - V[\ell^*]$
16: $product \leftarrow (w_{\pi[\ell^*]} + G_{\pi[\ell^*]})/w_{\pi[\ell^*-1]}$
17: $denominator \leftarrow 1$
18: for $k \leftarrow \ell^* - 1$ down to 1 do
19:     $denominator \leftarrow denominator + product$
20:     if $k \neq 1$ then
21:         $product \leftarrow product \cdot (w_{\pi[k]} + G_{\pi[k]})/w_{\pi[k-1]}$
22:     end if
23: end for
24: $\alpha_{\pi[\ell^*]} \leftarrow numerator/denominator$
25: for $k \leftarrow \ell^* - 1$ to 1 do
26:     $\alpha_{\pi[k]} \leftarrow (g_{\pi[k+1]} + \alpha_{\pi[k+1]}(w_{\pi[k+1]} + G_{\pi[k+1]}))/w_{\pi[k]}$
27: end for
28: for $k \leftarrow \ell^* + 1$ to $n$ do
29:     $\alpha_{\pi[k]} \leftarrow 0$
30: end for
31: $T^* \leftarrow \alpha_{\pi[1]} \cdot (w_{\pi[1]} + G_{\pi[1]}) + g_{\pi[1]}$
```

VI. A FAST CONSTRUCTIVE HEURISTIC WITH FEEDBACK

Algorithm 1 described in the previous section computes the optimal number of processors and the optimal loads for a given activation order. In this section, we propose a quick constructive heuristic for the general case, in which the activation order is unknown. Since a solution S is uniquely associated with an order given by a vector π, it will be represented by $S.\pi$, the associated makespan by $S.T^*$, and the optimal number of processors by $S.\ell^*$.

A heuristic for the problem of scheduling divisible loads may then be seen as any algorithm that generates a good activation order and then computes the associated optimal loads. The heuristic proposed in this section is based on the idea of equivalent processors. Given a time duration T, if a load $\alpha_i = (T - g_i)/(w_i + G_i)$ is sent to processor P_i, for any $i = 1, \ldots, n$, then it remains busy with communication and processing for the whole duration T. The behavior of this processor in this scenario is then equivalent to that of a processor P_i^{eq} with the same processing power, but with no latency and an equivalent communication throughput $1/G_i^{eq} = 1/(G_i + (g_i/\alpha_i))$. For instance, consider a processor P_1 with $w_1 = 10$, $G_1 = 5$, and $g_1 = 7$, receiving a load $\alpha_1 = 10$. This processor is equivalent to another processor P_1^{eq} with $w_1^{eq} = 10$, $g_1^{eq} = 0$, and $G_1^{eq} = 5 + 7/10$, that would take exactly the same communication time (57 time units) and the same processing time (50 time units) to process the same load ($\alpha_1 = 10$).

The optimal activation order for systems with no latencies can be determined by the algorithm in [7], which schedules first the processors with faster transmission links. This strategy is applied to find an optimal activation order for the equivalent processors P_i^{eq}, which is then used for the original processors P_i, with $i = 1, \ldots, n$,

Algorithm 2 describes a heuristic using a feedback mechanism for the problem of scheduling divisible loads. It computes a solution using an upper bound to the optimal makespan. Using some initial activation order, the algorithm computes equivalent processors and schedules them considering their transmission links. A new activation order results. The makespan associated with this solution is used as a refinement of the original upper bound and a new solution is computed. The procedure continues until the upper bound cannot be further refined.

A tentative activation order defined by vector π is initialized in line 1, with $G_{\pi[1]} \leq \ldots \leq G_{\pi[n]}$. Algorithm 1 is used in line 2 to determine the makespan UB for this order, which is used as the initial upper bound to the optimal makespan.

Initializations associated with the new solution to be computed are performed in lines 4 to 6, with no processors activated. The upper bound UB is used in the computation of a possibly better activation order in lines 7 to 32. The yet non-activated processors are scanned in lines 9 to 26. The next processor to be activated is that with the faster equivalent communication link under the assumption of UB as an upper bound to the optimal makespan. Processors that cannot be activated in the remaining time due to their latencies are eliminated from the computations in lines 11 to 13. For the others, their loads and their equivalent G^{eq} are computed in lines 15 and 16. The next processor j^* to be activated is updated in lines 17 to 23. Processor j^* is inserted in the current solution as the next to be activated in lines 27 to 29 and the remaining time is updated in line 30. In line 33 the makespan UB associated with the new activation order is computed by Algorithm 1. A new solution is computed if the new makespan improves the current best, otherwise the algorithm stops (line 34). We notice that each solution is computed in time $O(n^2)$.

Algorithm 2 Feedback heuristic

```
Ensure: Optimal activation order and minimum makespan
 1: Let $\pi$ define the initial activation order, with $G_{\pi[1]} \leq \ldots \leq G_{\pi[n]}$.
 2: Compute the makespan $UB$ using Algorithm 1.
 3: repeat
 4:   $BestOrder \leftarrow \pi$
 5:   $MinimumMakespan \leftarrow UB$
 6:   $activated(j) \leftarrow$ .FALSE., $\forall j = 1, \ldots, n$
 7:   for $i \leftarrow 1$ to $n$ do
 8:     $j^* \leftarrow 0$
 9:     for $j \leftarrow 1$ to $n$ do
10:       if .NOT.$activated(j)$ then
11:         if $g_j \geq UB$ then
12:           $\alpha_j \leftarrow 0$
13:           $G_j^{eq} \leftarrow \infty$
14:         else
15:           $\alpha_j \leftarrow (UB - g_j)/(w_j + G_j)$
16:           $G_j^{eq} \leftarrow G_j + g_j/\alpha_j$
17:           if $j^* = 0$ then
18:             $j^* \leftarrow j$
19:           else
20:             if $G_j^{eq} \leq G_{j^*}^{eq}$ then
21:               $j^* \leftarrow j$
22:             end if
23:           end if
24:         end if
25:       end if
26:     end for
27:     if $j^* \neq 0$ then
28:       $\pi[i] \leftarrow j^*$
29:       $activated(j^*) \leftarrow$ .TRUE.
30:       $UB \leftarrow UB - G_{j^*}^{eq}\alpha_{j^*}$
31:     end if
32:   end for
33:   Compute the new makespan $UB$ using Algorithm 1.
34: until $UB \geq MinimumMakespan$
```

VII. Computational experiments

All computational experiments were performed on a Pentium 4 computer with a 3 GHz processor and 512 Mbytes of RAM memory running under the Microsoft Windows XP operating system. The heuristics were executed in a single user environment with the highest execution priority. All algorithms were coded in Microsoft Visual C++(R) 6.0 configured to maximize speed. CPLEX version 8.0 was used to solve the exact single-installment mixed integer programming model, configured to search for the solution for up to one hour, timing out otherwise.

We have generated 120 configurations for the grids involved in the experiments, varying the number n (10, 20, 40, 80, and 160) of processors and considering eight combinations of their parameter values w_i, G_i, and g_i, with low assignments ranging in the interval $[1, 100]$ and high ones in the interval $[1000, 100000]$. Three instances were generated for each combination of n, w_i, G_i, and g_i. Six different values of the load W (100, 200, 400, 800, 1600, and 3200) were considered for each grid configuration, corresponding to 18 instances for each combination of parameters w_i, G_i, and g_i and to a total of 720 test instances.

We first report numerical results obtained with the use of CPLEX to solve the exact single-installment mixed integer programming model. A time limit of 3600 seconds is imposed to each CPLEX run. Table I presents the number of optimal solutions found by CPLEX over the 18 instances associated with each grid configuration. It also gives the average running time over the 18 instances, whenever CPLEX was able to found optimal solutions to all of them.

These results show that the computation times depend not only on the number of processors, but also on the grid

TABLE I

OPTIMAL SOLUTIONS AND RUNNING TIMES IN SECONDS

			$n = 10$		$n = 20$		$n = 40$		$n = 80$		$n = 160$	
w_i	g_i	G_i	opt.	time	opt.	time	opt.	time	opt.	time	opt.	time
low	low	low	18	0.25	18	0.81	18	43.81	5	–	1	–
low	low	high	18	0.05	18	0.08	18	0.36	18	1.37	18	4.94
low	high	low	18	0.08	18	0.20	18	0.46	18	2.78	18	9.27
low	high	high	18	0.14	18	0.25	18	0.50	18	1.20	18	3.57
high	low	low	18	14.64	0	–	0	–	0	–	0	–
high	low	high	18	0.06	18	9.37	16	–	0	–	0	–
high	high	low	18	2.85	5	–	1	–	2	–	0	–
high	high	high	18	0.13	18	0.68	18	52.52	8	–	2	–

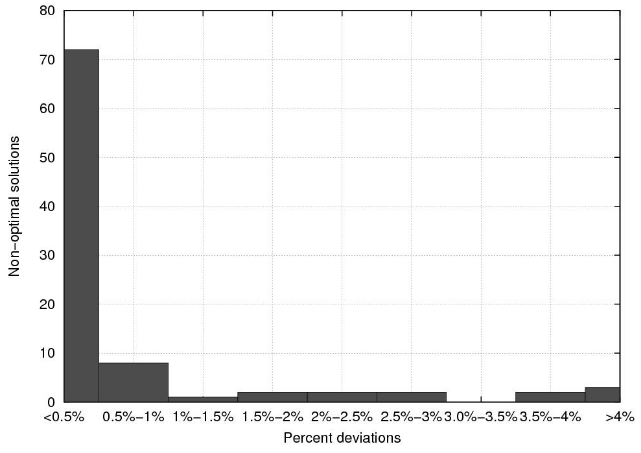

Fig. 5. Percent deviation of non-optimal makespans

characteristics. The exact solver found 490 optimal solutions over the 720 test instances. Although all instances with ten processors have been solved within the one-hour time limit, already 31 over the 144 instances with 20 processors could not be solved within this time limit. Furthermore, instances with high w_i and low G_i seem to be particularly hard, showing that the use of exact solution does not seem to be feasible for solving real life problems.

The heuristic described in Algorithm 2 was applied to all test instances in the second experiment. It found the optimal solutions for 398 out of the 490 instances for which CPLEX was able to prove optimality, as shown in Table II. The average and maximum percent deviations from the optimal value were never greater than 0.60% and 8.62%, respectively, for the remaining 92 instances. Figure 5 displays the distribution of the percent deviation of the non-optimal makespans obtained by the heuristic for these instances, with respect to the optimal schedules found by CPLEX. The feedback heuristic is very quick and never required more than four iterations to find the best solution. It took 3 ms of computation time on average, and no more than 32 ms for the hardest instance.

The average relative gap (over all 720 test instances)

TABLE II

PROVABLE OPTIMAL SOLUTIONS FOUND BY THE FEEDBACK HEURISTIC

			Processors (n)				
w_i	g_i	G_i	10	20	40	80	160
low	low	low	11	10	4	1	0
low	low	high	18	18	18	18	17
low	high	low	13	17	17	11	13
low	high	high	18	18	18	18	14
high	low	low	18	–	–	–	–
high	low	high	18	18	16	–	–
high	high	low	16	1	0	1	–
high	high	high	14	14	5	3	2

between the best solution value found by the feedback heuristic and the optimal value of the linear relaxation is only 17%, which is an additional indication of the high quality of the solutions provided by Algorithm 2.

VIII. MULTIPLE-INSTALLMENT MODEL

In this section, we address the formulation of the multi-installment problem as an integer linear program. Given an upper bound p, the proposed model determines the optimal number of installments, instead of heuristically fixing the latter as e.g. in [2], [16]. Furthermore, the new model also determines the best processor activation order, instead of choosing some activation sequence using a heuristics like that in [16].

Let the new binary variable x_{kij} be equal to 1 if processor P_i is the j^{th} to be activated and receive data in round k, 0 otherwise. As noticed above, the model allows different activation sequences within each installment. Accordingly, let $\alpha_{kij} > 0$ be the amount of data sent to processor P_i if it is the j^{th} to be activated in installment k ($\alpha_{kij} = 0$ otherwise), t_{kj} be the time instant in which the j^{th} processor to be activated starts receiving data in installment k, and T be the makespan, as before. Model 2 is an appropriate formulation for the multi-installment divisible load scheduling problem.

Constraints (19) imply that at most one processor can be the j^{th} to receive data at each round k. Constraints (20) imply that each processor can be chosen at most once to receive data at each round k. Constraints (21) guarantee that there must be j previously activated processors when the $(j+1)^{th}$

is activated at each round k. Constraint (22) ensures that the total load W is divided over the processors.

For each round k, constraints (23) ensure that processor i can only be the j^{th} to receive data if it was chosen to be the j^{th} in the activation order for that round. Constraint (24) establishes that the first processor to be activated will start to receive data at instant zero. Constraints (25) are used to enforce that, for each round k, the j^{th} processor to be activated will start to receive data after the previous processor finishes receiving its data. For each round k, constraints (26) imply that the j^{th} processor in the activation order can only start processing data after it has finished processing all the data it has received in the previous round. Constraints (27) imply that, for each round k, the first processor in the activation order can only start to receive data after the last processor of the previous round has finished receiving data. Constraints (28) imply that for the last round all active processors must finish processing data at the same time.

Small instances with up to ten processors could be solved to optimality with the above model by CPLEX 8.0. Preliminary results indicate that significant improvements in the makespans are possible, with respect to those obtained by single-installment schedulings and by several heuristics for multi-installment scheduling. This fact illustrates the need for more powerful heuristics for scheduling divisible loads, based on intelligent search procedures. Furthermore, the above model provides a framework for comparing the quality of different heuristics with respect to the exact optimal solutions.

IX. Concluding remarks

Divisible load applications occur in many fields of science and engineering and can be parallelized following the master-worker paradigm. However, they pose several scheduling challenges and require fast, effective algorithms.

New mixed integer linear programming formulations for finding single-round and multi-round schedules minimizing the makespan were presented in this work. A linear-time exact algorithm was proposed for the special case in which the processor activation order is known beforehand, improving the $O(n \log n)$ complexity of the best algorithm to date. This algorithm is embedded within a greedy-with-feedback heuristic for finding good solutions for the single-round problem.

Numerical results on randomly generated instances illustrate the effectiveness of the heuristic. The algorithm is very fast and ran in no more than 32 ms for the hardest and largest test instances, with 160 processors. The speed of the proposed algorithm makes it a good candidate to be used as an on-line scheduler.

The heuristic is currently being improved by the hybridization of a randomized multistart version of Algorithm 2 with local search and path-relinking. It is also being extended to handle the general multi-installment case.

Model 2 Multiple-installment mixed integer program

$$T^* = \text{ minimum } T \qquad (18)$$

subject to:

$$\sum_{i=1}^{n} x_{kij} \leq 1 \qquad j = 1, \ldots, n, \; k = 1, \ldots, p \qquad (19)$$

$$\sum_{j=1}^{n} x_{kij} \leq 1 \qquad i = 1, \ldots, n, \; k = 1, \ldots, p \qquad (20)$$

$$\sum_{i=1}^{n} x_{kij} \geq \sum_{i=1}^{n} x_{ki,j+1} \qquad j = 1, \ldots, n-1, \; k = 1, \ldots, p \qquad (21)$$

$$\sum_{k=1}^{p} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{kij} = W \qquad (22)$$

$$\alpha_{kij} \leq W x_{kij} \qquad i, j = 1, \ldots, n, \; k = 1, \ldots, p \qquad (23)$$

$$t_{11} = 0 \qquad (24)$$

$$t_{kj} \geq t_{k,j-1} + \sum_{i=1}^{n} (g_i x_{ki,j-1} + G_i \alpha_{ki,j-1}) \qquad j = 2, \ldots, n, \; k = 1, \ldots, p \qquad (25)$$

$$t_{k,j} + \sum_{i=1}^{n} (g_i x_{kij} + G_i \alpha_{kij}) \geq t_{k-1,j} + \sum_{i=1}^{n} (g_i x_{k-1,ij} + (G_i + w_i)\alpha_{k-1,ij}) \qquad j = 1, \ldots, n, \; k =, \ldots, p \qquad (26)$$

$$t_{k1} \geq t_{k-1,n} + \sum_{i=1}^{n} (g_i x_{k-1,in} + G_i \alpha_{k-1,in}) \qquad k = 2, \ldots, p \qquad (27)$$

$$t_{pj} + \sum_{i=1}^{n} (g_i x_{pij} + (G_i + w_i)\alpha_{pij}) = T \qquad j = 1, \ldots, n \qquad (28)$$

$$x_{kij} \in \{0, 1\} \qquad i, j = 1, \ldots, n, \; k = 1, \ldots, p \qquad (29)$$

$$\alpha_{kij} \geq 0 \qquad i, j = 1, \ldots, n, \; k = 1, \ldots, p. \qquad (30)$$

References

[1] W. Alda, W. Dzwinel, J. Kitowski, J. Moscinski, and D. A. Yuen. Penetration mechanics via molecular dynamics. Research Report UMSI 93/58, University of Minnesota Supercomputing Institute, 1993.

[2] O. Beaumont, H. Casanova, A. Legr, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: Results and open problems. *IEEE Transactions on Parallel and Distributed Systems*, 16:207–218, 2005.

[3] O. Beaumont, A. Legrand, and Y. Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 98.2, Washington, 2003. IEEE Computer Society Press.

[4] V. Bharadwaj, D. Ghose, V. Mani, and T. G Robertazzi. *Scheduling divisible loads in parallel and distributed Systems*. IEEE Computer Society Press, 1996.

[5] J.-Y. Blanc and D. Trystram. Implementation of parallel numerical routines using broadcast communication schemes. In *Joint International Conference on Vector and Parallel Processing*, volume 457 of *Lecture Notes in Computer Science*, pages 467–478. Springer-Verlag, 1990.

[6] J. Blazewicz and M. Drozdowski. Scheduling divisible jobs on hypercubes. *Parallel Computing*, 21:1945–1956, 1995.

[7] J. Blazewicz and M. Drozdowski. Distributed processing of divisible jobs with communication startup costs. *Discrete Applied Mathematics*, 76:21–41, 1997.

[8] W. J. Camp, S. J. Plimpton, B. A. Hendrickson, and R. W. Leland. Massively parallel methods for engineering and science problems. *Communications of the ACM*, 37:30–41, 1994.

[9] Y. C. Cheng and T. G Robertazzi. Distributed computation with communication delay. *IEEE Transactions on Aerospace and Electronic Systems*, 24:700–712, 1988.

[10] M. Drozdowski. *Selected problems of scheduling tasks in multiprocessor computing systems*. PhD thesis, Poznan University of Technology, Poznan, 1997.

[11] M. Drozdowski and P. Wolniewicz. Experiments with scheduling divisible tasks in clusters of workstations. In A. Bode, T. Ludwig II, W. Karl, and R. Wismüller, editors, *6th International Euro-Par Conference*, volume 1900 of *Lecture Notes in Computer Science*, pages 311–319. Springer-Verlag, 2000.

[12] J. Hu and R. Klefstad. Scheduling divisible loads on bus networks with start-up costs by utilizing multiple data transfer streams: PORI. In *International Conference on Parallel Processing*, page 23, 2007.

[13] D. Turgay Altilar and P. Yakup. Optimal scheduling algorithms for communication constrained parallel processing. In B. Monien and R. Feldmann, editors, *8th International Euro-Par Conference*, volume 2400 of *Lecture Notes in Computer Science*, pages 197–206. Springer-Verlag, 2002.

[14] P. Wolniewicz. *Divisible job scheduling in systems with limited memory*. PhD thesis, Poznan University of Technology, Poznan, 2003.

[15] Y. Yang and H. Casanova. RUMR: Robust scheduling for divisible workloads. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 114–125, Seattle, 2003.

[16] Y. Yang and H. Casanova. UMR: A multi-round algorithm for scheduling divisible workloads. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 24.2, Washington, 2003. IEEE Computer Society.

[17] Y. Yang, K. van der Raadt, and H. Casanova. Multiround algorithms for scheduling divisible loads. *IEEE Transactions on Parallel and Distributed Systems*, 16:1092–1102, 2005.