Proceedings Paper:
Shakhlevich, NV (2013) Scheduling divisible loads to optimize the computation time and cost. In: Altmann, J, Vanmechelen, K and Rana, OF, (eds.) Economics of Grids, Clouds, Systems, and Services, 10th International Conference (GECON 2013), Proceedings. 10th International Conference (GECON 2013), 18-20 Sep 2013, Zaragoza, Spain. Springer, 138 - 148. ISBN: 978-3-319-02413-4.

https://doi.org/10.1007/978-3-319-02414-1_10

Scheduling Divisible Loads to Optimize the Computation Time and Cost

Natalia V. Shakhlevich
School of Computing, University of Leeds, Leeds LS2 9JT, U.K.

Abstract

Efficient load distribution plays an important role in grid and cloud applications. In a typical problem, a divisible load should be split into parts and allocated to several processors, with one processor responsible for the data transfer. Since processors have different speed and cost characteristics, selecting the processors order for the transmission and defining the chunk sizes affect the computation time and cost. We perform a systematic analysis of the model analysing the properties of Pareto optimal solutions. We demonstrate that the earlier research has a number of limitations. In particular, it is generally assumed that the load should be distributed so that all processors have equal completion times, while in fact there often exists a dominating schedule with non-simultaneous finishing times of the processors. Moreover, fixing the processor sequence in the non-decreasing order of the cost-characteristic may be appropriate only for Pareto-optimal solutions with relatively large deadlines; optimal schedules for tight deadlines may have a different order of processors. We conclude with an efficient algorithm for finding the time-cost tradeoff.
Keywords: scheduling, divisible load, time/cost optimization

1 Introduction

Parallel computer systems have given rise to new scheduling models that go beyond the classical scheduling theory. While in a traditional scheduling model a task can be processed by one machine at a time, a new feature of multiprocessor computations is the ability to split tasks into several parts and to process them simultaneously by different processors, see, e.g., [5, 8]. An additional feature of modern Grid computing and cloud computing systems is the introduction of the cost factor, see, e.g. [2, 6, 10]. This study is motivated by the lack of theoretical research in the area and some inaccuracies which can be found in the earlier research.

We consider the network model described in [7]. There is a set $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ of m processors connected via a bus type communication medium. One processor of the set $\mathcal{P}$ is selected as a master processor to receive a divisible load of size τ and to divide it into portions of size $\alpha_1\tau$, $\alpha_2\tau$,..., $\alpha_m\tau$, $\sum_{k=1}^{m} \alpha_k = 1$, which are then transmitted to slave processors from $\mathcal{P}$ to perform required computations.

The processors have different computation speeds and for each processor $P_k \in \mathcal{P}$ the inverse of the speed w_k is given. This implies that the load of size $\alpha_k\tau$ allocated to processor P_k requires computation time $\alpha_k w_k \tau$.

If P_1 is selected as a master processor and the transmission sequence is P_2, P_3, ..., P_m, then P_1 can start processing its own load of size $\alpha_1\tau$ at time 0 and at the same time it can start transmitting the relevant portions of the load first to P_2, then to P_3, etc., until the last

portion is transmitted to P_m, see Fig. 1. If z is the time needed to transmit the whole load of size τ, then the communication time for transmitting the portion $\alpha_k\tau$ to processor P_k is $\alpha_k z$.

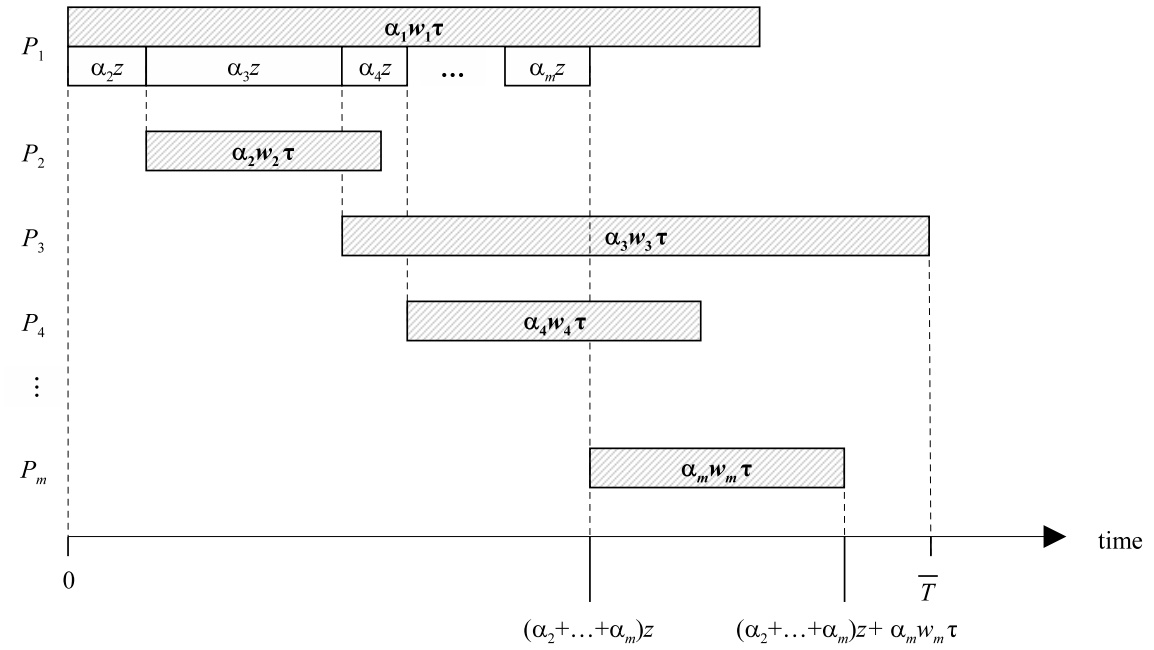

Figure 1: An example of a schedule with master processor P_1 and transmission sequence $P_2, \ldots, P_m$

With the selected transmission order, processor P_1 completes its portion of computation at time

$$T_1 = \alpha_1 w_1 \tau. \tag{1}$$

Processor P_k, $2 \leq k \leq m$, receives its portion of the load at time $\sum_{i=2}^{k} \alpha_i z$ and immediately after that it can start computation, which takes $\alpha_k w_k \tau$ time. Thus processor P_k completes its portion of the load at time

$$T_k = \sum_{i=2}^{k} \alpha_i z + \alpha_k w_k \tau.$$

The finish time T of the load is defined as the *makespan* of the schedule; it is equal to the maximum completion time among all processors,

$$T = \max_{1 \leq k \leq m} \{T_k\}. \tag{2}$$

It is assumed in the described scenario that the master processor can perform data transmission and computation simultaneously. This usually happens if the processor is equipped with an additional front-end co-processor which takes care of all data transfer so that the master processor can perform computation as any other processor of the network. In the absence of a front-end co-processor, the master processor performs data transmission first and only after that it can start computing its portion of the load. In the latter scenario, Fig. 1 should be modified so that for processor P_1 the box "$\alpha_1 w_1 \tau$" is moved immediately

after "$\alpha_m z$", and formula (1) should be replaced by

$$T_1 = \sum_{i=2}^{m} \alpha_i z + \alpha_1 w_1 \tau. \tag{3}$$

Processing the load in accordance with the load distribution α_1, α_2,..., α_m incurs computation cost which depends on processors' costs. Following the notation from [7], we denote the cost of using processor $P_k \in \mathcal{P}$ during one time unit by c_k so that the cost of performing the portion of the load $\alpha_k w_k \tau$ by processor P_k is $c_k \alpha_k w_k \tau$. The overall cost of using all processors P is therefore

$$K = \sum_{k=1}^{m} c_k \alpha_k w_k \tau.$$

Thus a schedule S is given by

- the transmission sequence with the first processor of the sequence selected as a master processor

  and

- the load distribution α_1, α_2,..., α_m with $\sum_{k=1}^{m} \alpha_k = 1$.

In this paper we assume that the processors are numbered so that

$$c_1 w_1 \leq c_2 w_2 \leq \cdots \leq c_m w_m. \tag{4}$$

The quality of a scheduled is measured in terms of the two characteristics: maximum completion time T and computation cost K. As a solution of a bicriteria problem we accept the set of Pareto optimal points defined by the break-points of the so-called *efficiency frontier*. In a pair of the associated single criterion problems,

$$\begin{array}{ll} \min & K \\ \text{s.t.} & T \leq \overline{T} \end{array} \tag{5}$$

and

$$\begin{array}{ll} \min & T \\ \text{s.t.} & K \leq \overline{K} \end{array}$$

one of the objectives is bounded while the other one is to be minimized. Here $\overline{T}$ and $\overline{K}$ are threshold values of the load finish time and computation cost, respectively.

2 Finding the Efficiency Frontier

In the (T, K)-space, the set of Pareto-optimal points represents a time-cost efficiency frontier. We start with an overview of the main outcomes of [7] and then proceed with the description of additional steps needed to find a correct efficiency frontier.

It is claimed in [7] that all break-points correspond to the schedules of a special type: the processor sequence is the same for all break-points and it is $(P_1, P_2, \ldots, P_m)$; only a subset of the several first processors have a non-zero load, while the remaining processors are idle. Recall that processors are numbered in accordance with (4).

To represent the described schedules formally, introduce notation ($P_1^*, P_2^*, \ldots, P_k^*, -, \ldots, -$) to indicate that processors $P_1, P_2, \ldots, P_k$ are fully loaded completing computation at time T, while the remaining processors $P_{k+1}, P_{k+2}, \ldots, P_m$ are idle. Then the set of the break-points established in [7] is of the form:

$$
\begin{array}{lllllllll}
(& P_1^*, & -, & -, & \cdots, & -, & -, & \cdots, & -, & - \quad) \\
(& P_1^*, & P_2^*, & -, & \cdots, & -, & -, & \cdots, & -, & - \quad) \\
(& P_1^*, & P_2^*, & P_3^*, & \cdots, & -, & -, & \cdots, & -, & - \quad) \\
 & & & \ddots & & & & & & \\
(& P_1^*, & P_2^*, & P_3^*, & \cdots, & P_k^*, & -, & \cdots, & -, & - \quad) \\
(& P_1^*, & P_2^*, & P_3^*, & \cdots, & P_k^*, & P_{k+1}^*, & & -, & - \quad) \\
 & & & & & & \ddots & & & \\
(& P_1^*, & P_2^*, & P_3^*, & \cdots, & P_k^*, & P_{k+1}^*, & \cdots, & P_{m-1}^*, & - \quad) \\
(& P_1^*, & P_2^*, & P_3^*, & \cdots, & P_k^*, & P_{k+1}^*, & \cdots, & P_{m-1}^*, & P_m^* \quad)
\end{array}
$$

The graphical representation of the efficiency frontier from [7] for the case of $m = 3$ processors is shown in Fig. 2. The three break-points, considered right to left, are $(P_1^*, -, -)$, $(P_1^*, P_2^*, -)$ and (P_1^*, P_2^*, P_3^*). When transition from $(P_1^*, -, -)$ to $(P_1^*, P_2^*, -)$ is performed, the load from P_1 is re-distributed to P_2 until both processors have equal completion time; the intermediate points belonging to that segment of the efficiency frontier are denoted by $(P_1^*, P_2, -)$, where notation P_2 in the schedule description indicates that processor P_2 is partly loaded. Similarly, when transition from $(P_1^*, P_2^*, -)$ to (P_1^*, P_2^*, P_3^*) is performed, the load from P_1 and P_2 is re-distributed to P_3 until all three processors have equal completion time; the intermediate points belonging to that segment are denoted by (P_1^*, P_2^*, P_3), where notation P_3 indicates that processor P_3 is partly loaded, while notation P_1^*, P_2^* implies that the corresponding processors are fully loaded completing their portions of the load simultaneously.

It appears that the efficiency frontier is more complicated than the one presented in [7]. In particular, it includes also the points with the processor order different from $(P_1, P_2, \ldots, P_m)$. In fact, the efficiency frontier can be found as the set of non-dominating segments of m curves $\mathcal{C}_\ell$, $\ell = 1, \ldots, m$. Each curve $\mathcal{C}_\ell$ consists of linear segments and corresponds to a processor sequence with a fixed master processor P_ℓ. As we prove in the appendix, in the class of schedules with a fixed master processor P_ℓ, an optimal processor sequence is $(P_\ell, P_1, P_2, \ldots, P_{\ell-1}, P_{\ell+1}, \ldots, P_m)$. If $\ell > 1$, then the first $\ell - 1$ breakpoints (considered in the (T, K)-space from right to left) correspond to schedules in which the master processor P_ℓ performs only data transmission and does not perform ant computation; the next breakpoint involves all ℓ processors fully loaded, so that the master processor P_ℓ performs both, data transmission and computation; in the remaining $m - \ell$ schedules, ℓ first processors are fully loaded together with an increasing number of additional slave processors with indices larger than ℓ.

Formally, the break-points of the curve $\mathcal{C}_\ell$ with a fixed master processor P_ℓ are of the form:

processor P_ℓ does not perform any computation, only data transmission

$$
\begin{array}{lllllllllll}
(& \underline{P_\ell}, & P_1^*, & -, & -, & \cdots, & -, & -, & \cdots, & -, & - \quad) \\
(& \underline{P_\ell}, & P_1^*, & P_2^*, & -, & \cdots, & -, & -, & \cdots, & -, & - \quad) \\
(& \underline{P_\ell}, & P_1^*, & P_2^*, & P_3^*, & \cdots, & -, & -, & \cdots, & -, & - \quad) \\
 & & & & & \ddots & & & & -, & \\
(& \underline{P_\ell}, & P_1^*, & P_2^*, & P_3^*, & \cdots, & P_{\ell-1}^*, & -, & \cdots, & -, & - \quad)
\end{array}
$$

processor P_ℓ performs computation (until T) and data transmission

$$
\begin{array}{lllllllllll}
(& P_\ell^*, & P_1^*, & P_2^*, & P_3^*, & \cdots, & P_{\ell-1}^*, & -, & \cdots, & -, & - \quad) \\
(& P_\ell^*, & P_1^*, & P_2^*, & P_3^*, & \cdots, & P_{\ell-1}^*, & P_{\ell+1}^*, & \cdots, & -, & - \quad) \\
 & & & & & & & & \ddots & & \\
(& P_\ell^*, & P_1^*, & P_2^*, & P_3^*, & \cdots, & P_{\ell-1}^*, & P_{\ell+1}^*, & \cdots, & P_{m-1}^*, & - \quad) \\
(& P_\ell^*, & P_1^*, & P_2^*, & P_3^*, & \cdots, & P_{\ell-1}^*, & P_{\ell+1}^*, & \cdots, & P_{m-1}^*, & P_m^* \quad)
\end{array}
$$

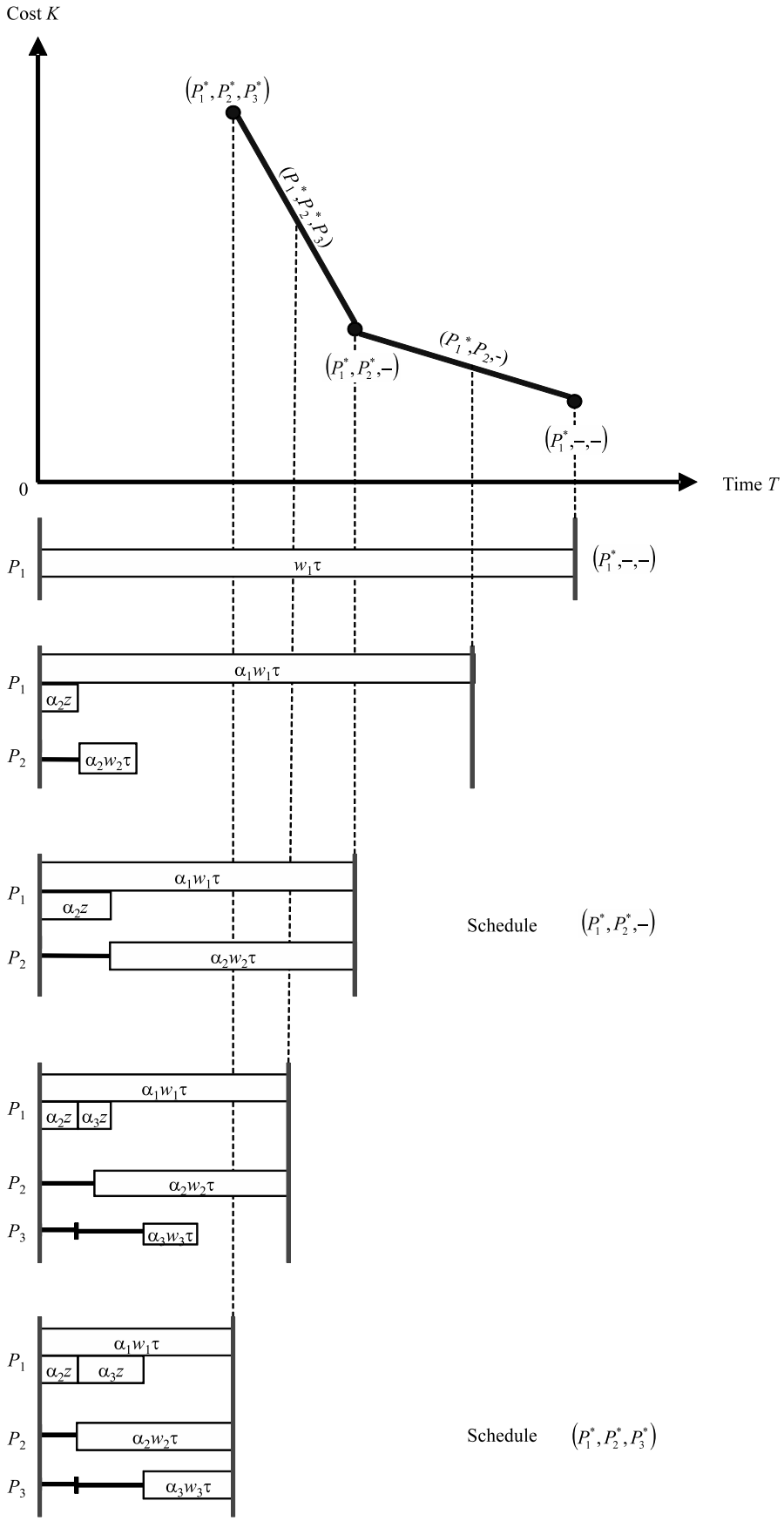

Figure 2: Efficiency frontier defined in [7] for the case of $m = 3$ processors and the associated schedules (idle processors are omitted)

Here notation $\underline{P}_\ell$, which appears in the first $\ell - 1$ schedules, indicates that processor P_ℓ performs only data transmission and no computation.

The intermediate points of the segments connecting the first $\ell-1$ break-points correspond to the re-distribution of the load to one additional slave processor, without involving the master processor P_ℓ in the computation; the previously loaded slave processors complete their load simultaneously. The transition from the break-point $\left(\underline{P}_\ell, P_1^*, P_2^*, \ldots, P_{\ell-1}^*, -, \ldots, -\right)$ to the ℓ-th break-point $\left(P_\ell^*, P_1^*, P_2^*, \ldots, P_{\ell-1}^*, -, \ldots, -\right)$ corresponds to the reallocation of the load from the slave processors $P_1^*, P_2^*, \ldots, P_{\ell-1}^*$ to the master processor P_ℓ, keeping the slave processors completing their computation simultaneously. Finally, the intermediate points of the segments the last $m - \ell$ break-points correspond to the re-distribution of the load to one additional slave processor that follows the current busy processors in the processor sequence $(P_\ell, P_1, P_2, \ldots, P_{\ell-1}, P_{\ell+1}, \ldots, P_m)$; all previously loaded processors complete their load simultaneously.

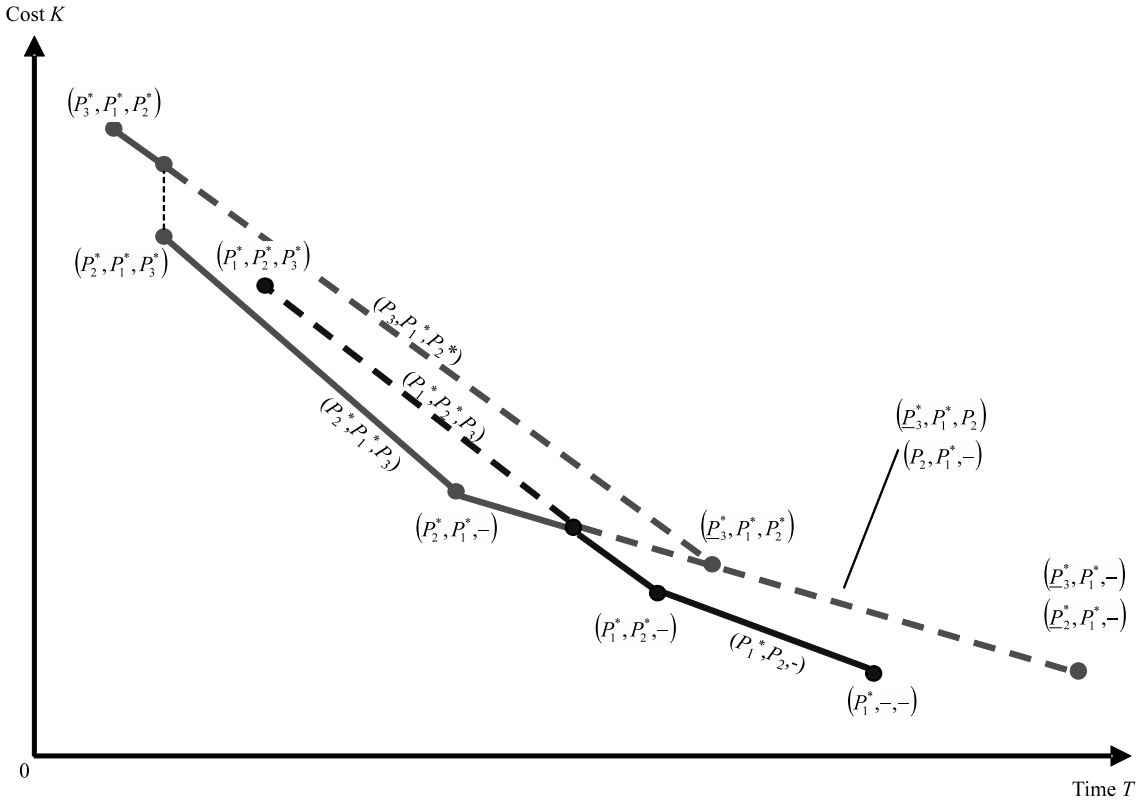

Figure 3: Three curves
$\mathcal{C}_1$ coonecting $(P_1^*, -, -)$, $(P_1^*, P_2^*, -)$, (P_1^*, P_2^*, P_3^*)
$\mathcal{C}_2$ coonecting $(\underline{P}_2^*, P_1^*, -)$, $(P_2^*, P_1^*, -)$, (P_2^*, P_1^*, P_3^*)
$\mathcal{C}_2$ coonecting $(\underline{P}_3^*, P_1^*, -)$, (P_3^*, P_1^*, P_2^*), (P_3^*, P_1^*, P_2^*)
and the trade-off curve (in solid lines) consisting of non-dominating segments and their parts

An example of the three curves $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ for the three-processor case is shown in Fig. 3. The resulting efficiency frontier consisting of non-dominated solutions is represented as solid lines. The efficiency frontier consists of the following components, listed from right

to left:

(i) the right-most segment of the curve $\mathcal{C}_1$ that connects $(P_1^*, -, -)$ and $(P_1^*, P_2^*, -)$;

(ii) a part of the second segment of $\mathcal{C}_1$ that connects $(P_1^*, P_2^*, -)$ and (P_1^*, P_2^*, P_3^*) until its intersection point with the first segment of $\mathcal{C}_2$;

(iii) a part of the segment of the curve $\mathcal{C}_2$ connecting $(\underline{P}_2, P_1^*, -)$ and $(P_2^*, P_1^*, -)$ starting at the right end with the previously defined intersection with $\mathcal{C}_1$;

(iv) the full segment of the curve $\mathcal{C}_2$ connecting $(P_2^*, P_1^*, -)$ and (P_2^*, P_1^*, P_3^*);

(v) a part of the last segment of the curve $\mathcal{C}_3$ connecting $(\underline{P}_3, P_1^*, P_2^*)$ and (P_3^*, P_1^*, P_2^*); its right-most T-value corresponds to the T-value of the left end (P_2^*, P_1^*, P_3^*) of the previous segment.

Notice that the resulting efficiency frontier is not convex and even not continuous.

While it is possible to prove that some points of the curves $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_m$ always dominate each other (for example, $(P_1^*, -, -, \ldots, -)$ always dominate $(\underline{P}_k, P_1^*, -, \ldots, -)$ for any $1 < k \leq m$), the dominance relation between other points can vary depending on the specific c_i- and w_i-values. For example in the three processor case, the may be no intersection point between curves $\mathcal{C}_1$ and $\mathcal{C}_2$, so that the whole curve $\mathcal{C}_1$ dominates all points of the curve $\mathcal{C}_2$.

We demonstrate in the full version of the paper that for each curve $\mathcal{C}_\ell$, all its break-points can be found in $O(m^2)$ time since each subsequent break-point can be defined from the previous one in $O(m)$ time by re-calculating the associated α_i-values, $1 \leq i \leq m$. Thus all break-points of the curves $\mathcal{C}_1$, $\mathcal{C}_2$, $\ldots$, $\mathcal{C}_m$ can be found in $O(m^3)$ time.

Having constructed $m(m-1)$ segments of the curves $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_m$, the required efficiency frontier is found as the lower boundary among the curves.

3 Conclusions

In this paper, we have performed a systematic analysis of the problem of scheduling a divisible load on m processes in order to minimize the computation time and cost. An efficient algorithm for solving the bicriteria version of the problem defines optimal processor sequences for different segments of the efficiency frontier and the corresponding optimal load distribution among the processors.

Our study demonstrates that the earlier research [7] has a number of limitations. Some assumptions result in incorrect major conclusions. In particular, it is generally assumed in [7] that the load should be distributed so that all processors complete their portions simultaneously, while as we show, there often exists a dominating schedule with non-simultaneous finishing times of the processors. Moreover, fixing the processor sequence in the non-decreasing order of the cost/speed characteristic given by (4) may be appropriate only for Pareto-optimal solutions with relatively large deadlines; optimal schedules for tight deadlines may have a different order of processors with master processor P_ℓ, $1 < \ell \leq m$, moved in front of slave processors $P_1, P_2, \ldots, P_{\ell-1}, P_{\ell+1}, \ldots, P_m$.

The described model with a single divisible load provides a foundation for more advanced models which better describe various real-world scenarios. Further generalizations include multiple divisible loads, bandwith dependent formulae for calculation transmission times, multi-installment load distribution, multi-round schedules and more complex network

topologies. An attempt to generalize the results for the case of a more complex cost function that includes data transmission costs in addition to computation costs is presented in [3]. Clearly, a study of more complex models should rely on accurate analysis of the simplified model.

Appendix

The validity of the described algorithm follows from a number of properties of optimal schedules. The properties are proved for a single-criterion version of the problem (5) for a fixed makespan parameter T. Since T may take different values, the properties are correct for all schedules of the efficiency frontier.

The first two propositions provide a justification for fixing a processor sequence in an optimal solution; the third proposition establishes how the load should be distributed in an optimal solution.

We assume that processors are numbered in accordance with (4). Initially we consider an arbitrary processor sequence which can be different from the sequences listed in Section 2.

Proposition 1 'Swapping Two Neighbour Slave Processors'
Consider schedule S in which two neighbour slave processors P_i and P_k in the processor sequence compute portions of load α_i and α_k and have finishing times T_i and T_k, respectively. It is always possible to change the order of P_i and P_k in the processor sequence so that in a new schedule S' the loads are α'_i and α'_k, processor finish times are T'_i and T'_k and
(a) the loads are re-distributed so that $\alpha'_i = \alpha_i - \delta$ and $\alpha'_k = \alpha_k + \delta$ for $0 \leq \delta \leq \alpha_i$;
(b) the load on other processors remains the same;
(c) the maximum finish time of processors P_i and P_k does not increase:
$\max\{T'_i, T'_k\} \leq \max\{T_i, T_k\}$.

Proof. Introduce notation Θ for $\max\{T_i, T_k\}$. We consider the two cases depending on whether processor P_i finishes its portion of the load earlier than P_k or not.

Case 1: in the initial schedule S, $T_i \leq T_k$. This implies that

$$\alpha_i w_i \tau \leq \alpha_k (z + w_k \tau). \tag{6}$$

In the initial schedule S, we denote by H the length of the time interval from the start of $\alpha_i z$ until $\Theta = T_k$,

$$H = \alpha_i z + \alpha_k (z + w_k \tau).$$

Consider schedule S' obtained from S by swapping P_k and P_i. If in S' condition (c) is satisfied, then Proposition 1 holds. Otherwise we have a schedule shown in the right-hand-side of the figure, with $T'_i > \Theta$ (notice, that T'_k cannot exceed Θ since $\alpha_k (z + w_k \tau) < H$). In order to achieve condition (c), we need to move part $\delta \leq \alpha_i$ of the P_i-load to P_k, so that in the resulting schedule S' the load on P_i is $\alpha'_i = \alpha_i - \delta$ and the load on P_k is $\alpha'_k = \alpha_k + \delta$.

The following inequalities should be satisfied:

$$\begin{array}{ll} \alpha'_i \geq 0, & \text{(load allocated to } P_i \text{ does not become negative)}, \\ \alpha'_k (z + w_k \tau) \leq H, & (T'_k \text{ does not exceed } \Theta), \\ \alpha'_k z + \alpha'_i (z + w_i \tau) \leq H, & (T'_i \text{ does not exceed } \Theta). \end{array} \tag{7}$$

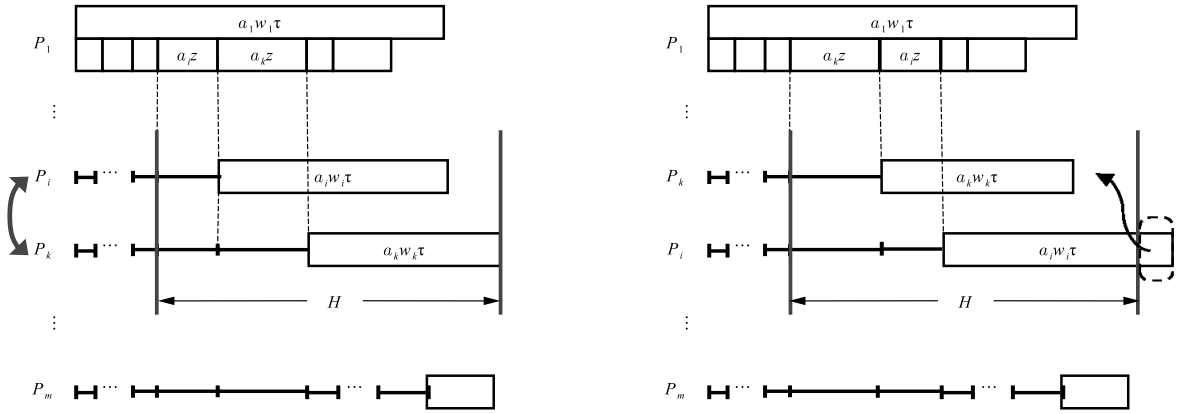

Figure 4: Changing the sequence of P_i and P_k in Case 1

It follows that

$$\begin{array}{lll} \alpha_i - \delta \geq 0, & \Rightarrow & \delta \leq a_i, \\ (\alpha_k + \delta)(z + w_k\tau) \leq \alpha_i z + \alpha_k (z + w_k\tau) & \Rightarrow & \delta \leq \frac{\alpha_i z}{z + w_k\tau}, \\ (\alpha_k + \delta) z + (\alpha_i - \delta)(z + w_i\tau) \leq \alpha_i z + \alpha_k (z + w_k\tau) & \Rightarrow & \delta \geq \alpha_i - \frac{\alpha_k w_k}{w_i}. \end{array}$$

The second and the third inequalities imply that

$$\alpha_i - \frac{\alpha_k w_k}{w_i} \leq \delta \leq \alpha_i - \frac{\alpha_i w_k \tau}{z + w_k\tau}, \tag{8}$$

while the first condition $\delta \leq a_i$ is redundant since $\frac{\alpha_i z}{z + w_k\tau} \leq a_i$. Notice, that (8) is feasible since

$$\frac{\alpha_k w_k}{w_i} \geq \frac{\alpha_i w_k \tau}{z + w_k\tau}$$

by (6).

Case 2: in the initial schedule S, $T_i > T_k$:

$$\alpha_i w_i \tau > \alpha_k (z + w_k\tau). \tag{9}$$

In the initial schedule S, we denote by G the length of the time interval from the start of $\alpha_i z$ until T_i,

$$G = \alpha_i (z + w_i\tau).$$

After swapping P_k and P_i, if the load is kept unchanged, we obtain schedule S' with $T_i' > \Theta$, while $T_k' \leq \Theta$. Hence we need to move part $\delta \leq \alpha_i$ of the P_i-load to P_k, so that in the resulting schedule S' the load on P_i is $\alpha_i' = \alpha_i - \delta$ and the load on P_k is $\alpha_k' = \alpha_k + \delta$. We need to guarantee that inequalities (7) with H replaced by G should be satisfied. It follows that

$$\begin{array}{lll} \alpha_i - \delta \geq 0, & \Rightarrow & \delta \leq a_i, \\ (\alpha_k + \delta)(z + w_k\tau) \leq \alpha_i (z + w_i\tau) & \Rightarrow & \delta \leq \frac{\alpha_i (z + w_i\tau)}{z + w_k\tau} - \alpha_k, \\ (\alpha_k + \delta) z + (\alpha_i - \delta)(z + w_i\tau) \leq \alpha_i (z + w_i\tau) & \Rightarrow & \delta \geq \frac{\alpha_k z}{w_i\tau}. \end{array}$$

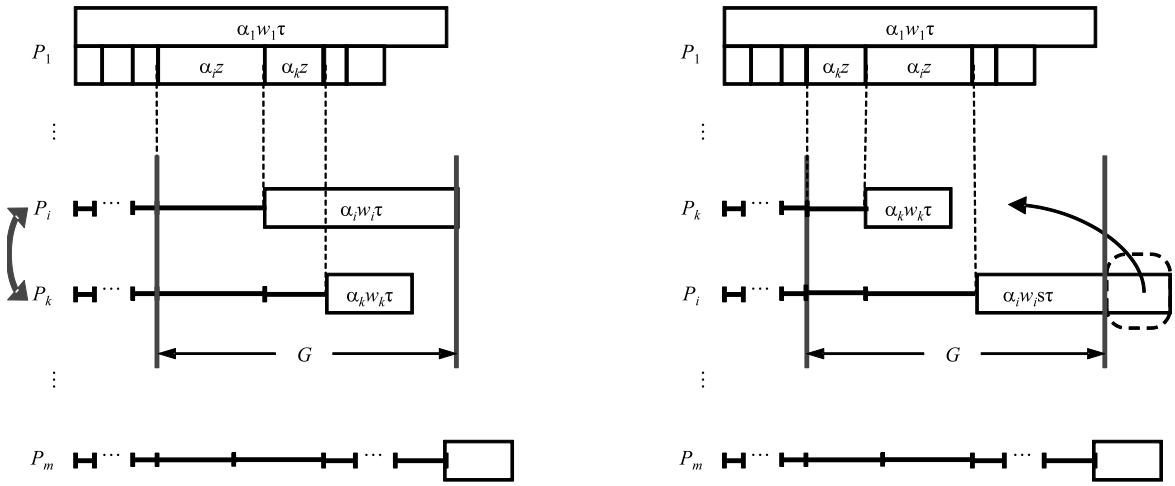

Figure 5: Changing the sequence of P_i and P_k in Case 2

It remains to show that condition

$$\frac{\alpha_k z}{w_i \tau} \leq \delta \leq \min\left\{\alpha_i, \frac{\alpha_i\left(z + w_i\tau\right)}{z + w_k\tau} - \alpha_k\right\}$$

is feasible. Indeed, if the smallest value in the right hand side is α_i, then

$$\frac{\alpha_k z}{w_i \tau} < a_i$$

due to (9). Alternatively, if α_i is the largest value in the r.h.s., then

$$\text{r.h.s.} - \text{l.h.s.} = \frac{\alpha_i\left(z + w_i\tau\right)}{z + w_k\tau} - \alpha_k - \frac{\alpha_k z}{w_i\tau} = \frac{\alpha_i w_i \tau - \alpha_k\left(z + w_k\tau\right)}{w_i\tau\left(z + w_k\tau\right)}\left(z + w_i\tau\right).$$

The numerator in the last expression is positive due to (9), so that a feasible δ that lies in-between the l.h.s and the r.h.s. does exist. ∎

Proposition 2 'Non-decreasing Sequence of $c_i w_i$ for Slave Processors'
If the master processor P_ℓ is fixed, then an optimal processor sequence is $(P_\ell, P_1, P_2, \ldots, P_{\ell-1}, P_{\ell+1}, \ldots, P_m)$.

Proof. Suppose in an optimal schedule there are two neighbour processors P_i and P_k, P_i precedes P_k and $c_i w_i > c_k w_k$. Then such a schedule is not optimal. Indeed, changing the order P_i and P_k in the processor sequence, as described in Property 1, leads to a schedule S' with the same finish time of the load and $\alpha_i' = \alpha_i - \delta$, $\alpha_k' = \alpha_k + \delta$. Since the load of other processors does not change, the computation cost changes from K to K' and $K' - K = \left(c_k w_k - c_i w_i\right)\tau\delta < 0$. ∎

Given a schedule, let T be its makespan, see (2). Depending on processors' finish times, we classify them as fully loaded, partly loaded or idle. Processor P_i is *busy* if $T_i \geq 0$, and it is *idle* otherwise. To be precise, we call processor P_i *fully loaded* if $T_i = T$ and it is *partly loaded* if $0 < T_i < T$. Notice that the master processor can be idle if its performs only data transmission and no computation.

Proposition 3 'Unique Partly Loaded Processor'
Consider a class of schedules with master processor P_ℓ and an optimal schedule with processor sequence $(P_\ell, P_1, P_2, \ldots, P_{\ell-1}, P_{\ell+1}, \ldots, P_m)$. Let k be the largest index among busy processors, $1 \le k \le m$. Then all processors with smaller indices $P_1, P_2, \ldots, P_{k-1}$ are fully loaded and all processors with larger indices $P_{k+1}, P_{k+2}, \ldots, P_m$ are idle.

Proof. We first show that there cannot be an idle or partly loaded slave processor, after which there is another (partly or fully) loaded processor. Suppose in schedule S processor P_i is idle or it is partly loaded, while P_{i+1} has a non-zero load. Then the load of P_{i+1} can be re-distributed by moving part δ of that load from P_{i+1} to P_i, $0 < \delta \le \alpha_{i+1}$, so that in the resulting schedule S',

$$\begin{aligned} \alpha'_{i+1} &= \alpha_{i+1} - \delta, \\ \alpha'_i &= \alpha_i + \delta, \end{aligned}$$

see Fig. 6. As a result of this transformation, the finish time of P_i increases (due to the increase in the transition time $\alpha'_i z > \alpha_i z$ and the increase in the computation time $\alpha'_i w_i \tau > \alpha_i w_i \tau$) while the finish time of P_{i+1} decreases (due to the decrease in the computation time $\alpha'_{i+1} w_{i+1} \tau < \alpha_{i+1} w_{i+1} \tau$; the total transition time does not change since $(\alpha'_i + \alpha'_{i+1})\, z = ((\alpha_i + \delta) + (\alpha_{i+1} - \delta))\, z = (\alpha_i + \alpha_{i+1})\, z$). The finish times of the remaining processors P_{i+1}, $\ldots$, P_m does not change decrease. The largest feasible value of δ either makes P_i fully loaded or makes P_{i+1} idle. Since $c_i w_i \le c_{i+1} w_{i+1}$ by (4), the cost of the resulting schedule does not increase.

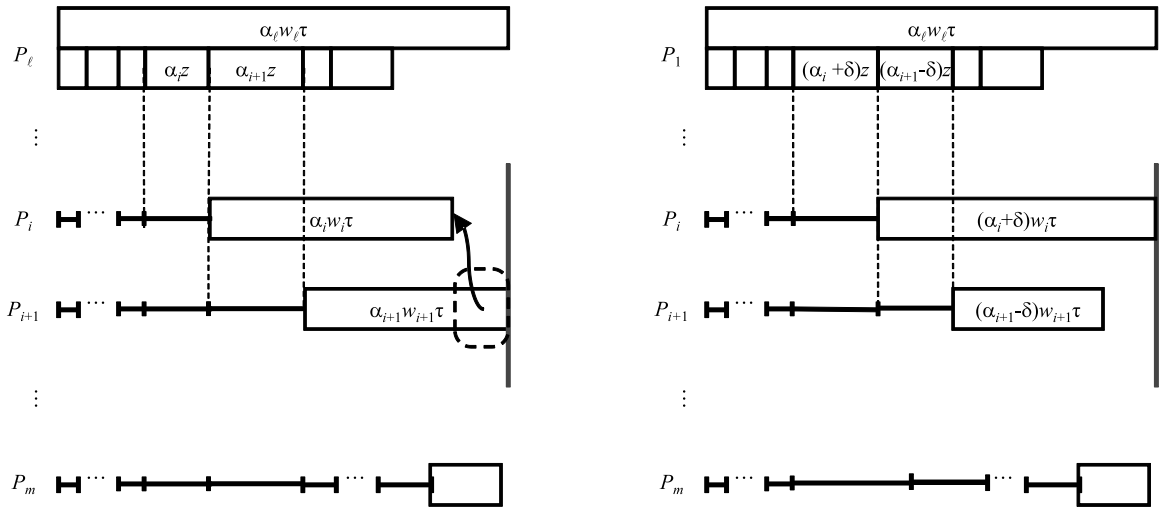

Figure 6: Re-distributing the load between two slave processors

In what follows we consider an optimal schedule in which the last slave processor with non-zero load is P_k, all slave processors with smaller indices are fully loaded and all slave processors with larger indices are idle. If index ℓ of the master processor P_ℓ satisfies $\ell > k$, so that $c_\ell w_\ell \ge c_k w_k$ and that processor has a non-zero load, then we re-distribute the load from P_ℓ to P_k. In the resulting schedule S',

$$\begin{aligned} \alpha'_\ell &= \alpha_\ell - \delta, \\ \alpha'_k &= \alpha_k + \delta, \end{aligned}$$

where $0 < \delta \leq \alpha_\ell$. This transformation does not affect other processors with non-zero load, but decreases the cost, see Fig. 7.

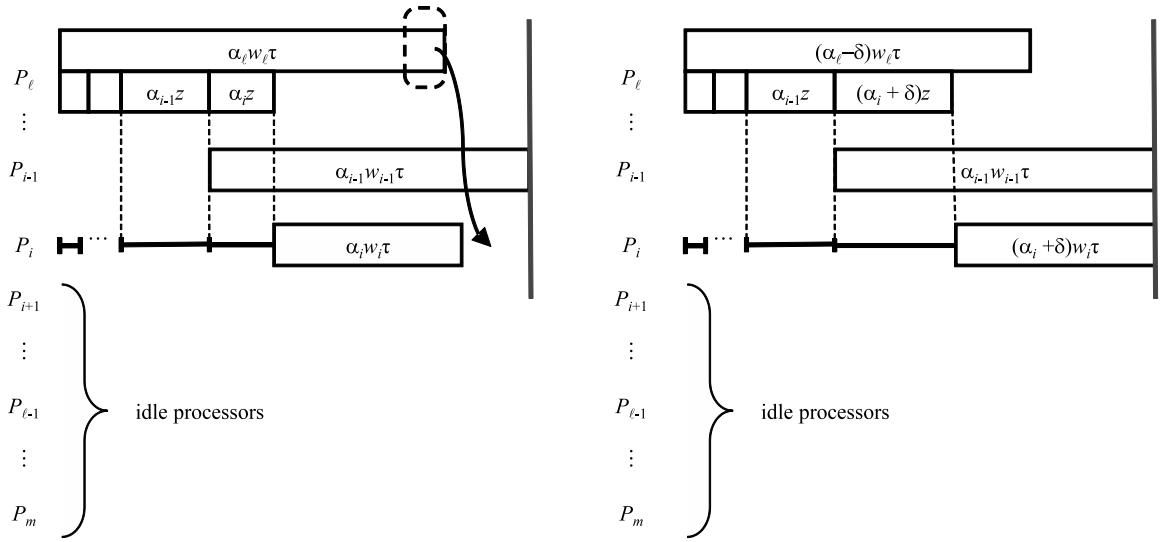

Figure 7: Re-distributing the load from P_ℓ to P_i, $i < \ell$

If in S', P_ℓ becomes idle, Proposition 3 is proved. Otherwise P_k becomes fully loaded and we perform a similar transformation by moving the load from P_ℓ to $P_{k+1}, \ldots, P_{\ell-1}$ until P_ℓ becomes idle or all slave processors with indices smaller than ℓ become fully loaded.

Finally, consider that case that index ℓ of the master processor P_ℓ satisfies $\ell < k$, so that $c_\ell w_\ell \leq c_k w_k$. We re-distribute the load from P_k to P_ℓ so that either P_k becomes idle or P_ℓ becomes fully loaded. As a result of this transformation, the finish time of P_k decreases (due to the decrease in the transition time $\alpha'_k z < \alpha_k z$ and the decrease in the computation time $\alpha'_k w_k \tau < \alpha_k w_k \tau$), and the finish time of P_ℓ increases (due to the increase in the computation time $\alpha'_\ell w_\ell \tau < \alpha_\ell w_\ell \tau$). The largest feasible value of δ either makes P_ℓ fully loaded or makes P_k idle; the cost of the resulting schedule does not increase. ∎

It follows from Propositions 1-3 that in a class of schedules with a fixed master processor P_ℓ all optimal schedules have processor order $(P_\ell, P_1, P_2, \ldots, P_{\ell-1}, P_{\ell+1}, \ldots, P_m)$ and for a given makespan threshold value T, an optimal schedule can be constructed by loading in full processors in the order $P_1, P_2, \ldots, P_{k-1}$ until the remaining load can be processed by P_k. Varying the T-values we conclude that all optimal schedules in that class belong to the curve $\mathcal{C}_\ell$ defined in Section 2.

Acknowledgements

This research was supported by the EPSRC funded project EP/G054304/1 "Quality of Service Provision for Grid applications via Intelligent Scheduling".

References

[1] Beaumont, O., Legrand, A., Robert, Y.: Scheduling divisible workloads on heterogeneous platforms, Parallel Comp. **29** (2003) 1121–1152

[2] Buyya, R., Abramson, D., Venugopal, S.: The grid economy, Proceedings of the IEEE **93** (2005) 698–714

[3] Charcranoon, S., Robertazzi, G.R., Luryu, S.: Parallel processor configuration design with processing/transmission costs, IEEE Trans. on Computers **49** (2000) 987–991.

[4] Chuprat, S., Baruah, S.: Real-time divisible load theory: incorporating computation costs, Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications

[5] Drozdowski, M.: Scheduling for Parallel Processing, Springer, London, 2009

[6] Kumar, S. Dutta, K., Mookerjee, V.: Maximizing business value by optimal assignment of jobs to resources in grid computing, European J. of Oper. Res. **194** (2009) 856–872

[7] Sohn, J., Robertazzi, T.G. and Luryi, S.: Optimizing computing costs using divisible load analysis, IEEE Trans. Parallel and Distributed Systems **9** (1998) 225–234

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

[9] van Hoesel, S., Wagelmans, A., Moerman, B.: Using geometric techniques to improve dynamic programming algorithms for the economic lot-sizing problem and extensions, European J. Oper. Res. **75** (1994) 312–331

[10] Yu, J., Buyya, R., Ramamohanarao, K.: Workflow Scheduling Algorithms for Grid Computing. In: F. Xhafa, A. Abraham. eds, Metaheuristics for Scheduling in Distributed Computing Environments, Springer, Berlin, Germany, 2008