

## **Bounding the makespan of best pre-schedulings of task graphs with fixed communication delays and random execution times on a virtual distributed system**

M. Nakechbandi \*   J.-Y. Colin \*   C. Delaruelle \*

---

**Abstract :** *In this paper, we consider the problem of scheduling tasks with fixed small communications delays on a virtual distributed memory multiprocessor when the tasks execution times are random. This problem extends the classical PERT scheduling problem with random execution times and no communications. We first present how to efficiently build pre-schedulings for this problem. We then compute a lower bound of the average makespan of a given pre-scheduling. We finally propose a lower and an upper bound of the execution time of the best pre-scheduling.*

**Key words :** makespan, parallel processors, distributed memory, communication delays, non deterministic scheduling

---

### **1. Introduction**

The efficient use of distributed memory multiprocessors is a very hard problem that involves efficient scheduling of the different parts of the parallel application. The different tasks of the application must be scheduled on the available processors and the communications must be mapped on the communication network.

Scheduling algorithms can be either static or dynamic. Static scheduling algorithms work at compile time, giving a solution ready for execution, while dynamic scheduling algorithms work at execution time and build the scheduling on-the-fly. While theoretically more efficient than dynamic algorithms, static scheduling algorithms suppose that the problem is deterministic, the execution and communication times being perfectly known at compile time. In [CoCh91], the authors present an optimal algorithm for the static scheduling of tasks of known duration with small communication delays and tasks duplication on a Virtual Distributed System. But tasks execution times are not always perfectly known, however. They may depend on the data, thus appearing random. Several papers deal with static scheduling of tasks graphs with random execution times, and no communication times. In [FrGr85] for example, the authors give in polynomial time lower bounds to the mean makespan.

In this paper, we study the static scheduling problem where the tasks execution times are positive independent random variables, and the communication delays between the tasks are perfectly known, positive values. We first present the formal definition of this new scheduling problem. We then propose a lower bound to the mean makespan of any scheduling. We next present a lower and an upper bound of the mean makespan of all the possible most efficient schedulings.

### **2. Problem definition**

#### **2.1 The VDSOPT problem**

We first briefly present the VDSOPT algorithm (described in [CoCh91]).

We first define the VDS scheduling problem.

A virtual distributed system (VDS) is a distributed memory multiprocessor architecture with a non

---

\* Laboratoire informatique du Havre (LIH), IUT du Havre, Place Robert Schuman, 76610 Le Havre, France  
email : [nakech@iut.univ-lehavre.fr](mailto:nakech@iut.univ-lehavre.fr), [colin@iut.univ-lehavre.fr](mailto:colin@iut.univ-lehavre.fr), [delaruelle@iut.univ-lehavre.fr](mailto:delaruelle@iut.univ-lehavre.fr)

limited number of identical processors and a complete communication network between these processors, so that each processor is directly connected to each other.

A VDS scheduling problem is specified by the four parameters  $(I, U, p, c)$ .

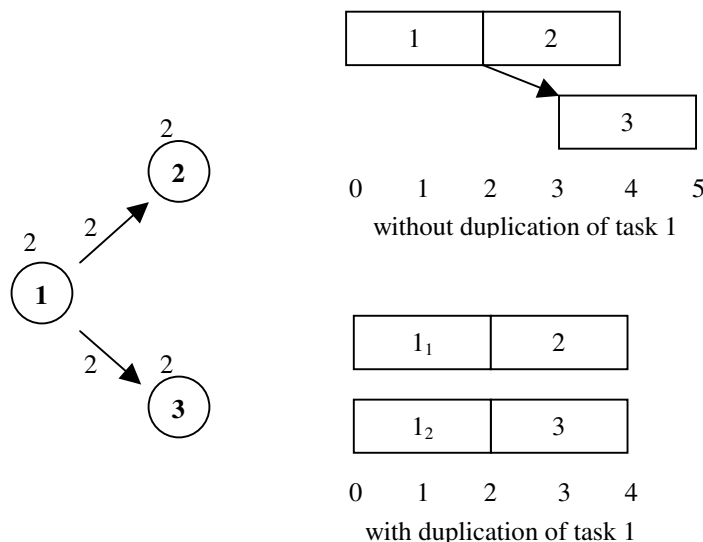
- $I = \{1, 2, \dots, n\}$  is a set of  $n$  tasks. The processing time of task  $i$  is  $p_i$ .
- $G = (I, U)$  is a directed acyclic graph (DAG) that models the precedence constraints. To each arc  $(i, j)$  is associated a positive communication time  $c_{i,j}$ , representing the communication delay if  $i$  and  $j$  are executed on different processors (if  $i$  and  $j$  are executed on the same processor, there is no need for a communication, so there is no communication delay).

A task is indivisible, starts when all the pieces of informations it needs from its predecessors are available to it, and gives all the informations needed by its successors at the end of its execution.

In the following, we will denote  $PRED(i)$  (respectively  $SUCC(i)$ ) the set of immediate predecessors (resp. successors) of task  $i$  in  $G$ .

Task duplication is allowed, that is several instances of the same task may be executed on different processors. We will denote  $i_k$  the  $k^{th}$  instance of task  $i$ . Figure 1 presents an example where duplication allows for a better schedule. On this figure, the values above the nodes are the processing times of the tasks, and the values above or below the edges are the communication delays.

**Example 1 :**



**Figure 1.** Example of a graph with its optimal schedules:  
the value above each node  $i$  of the graph is the processing time  $p_i$ ,  
the value above each arc  $(i, j)$  of the graph is the communication delay  $c_{i,j}$ .

A schedule  $S$  of a VDS scheduling problem is then a triple  $(F, t, \pi)$ , where

- $F(i), i \in I$  is the positive number of copies of task  $i$ .
- $t(i_k)$  is the starting time of copy  $i_k$  of task  $i$ .
- $\pi(i_k)$  is the processor assigned to copy  $i_k$  of task  $i$ .

A schedule  $S$  must also satisfy the following conditions:

- At least one copy of each task is processed.
- At any time, a processor executes at most one copy.
- If  $(i,j)$  is an arc of  $G$ , then for any copy  $j_l$  of task  $j$ , there must exist at least one copy  $i_k$  of task  $i$  such that

$$\begin{aligned} t(j_l) &\geq t(i_k) + p_i && \text{if } \pi(j_l) = \pi(i_k) \\ t(j_l) &\geq t(i_k) + p_i + c_{i,j} && \text{if } \pi(j_l) \neq \pi(i_k) \end{aligned}$$

If, in a schedule  $S$ ,  $i_k$  and  $j_l$  satisfy the two above inequalities, we will say that the **Generalized Precedence Constraint** is true for the two copies (in short, that  $\text{GPC}(i_k, j_l)$  is true). This Generalized Precedence Constraint means that a task needs its informations from one copy only of each one of its predecessors.

Let  $C(i_k)$  denote the completion time of copy  $i_k$  of task  $i$ . We then wish to minimize the makespan of the schedule, that is, the maximum task completion time  $C_{\max} = \max_{i \in G, k \leq F(i)} C(i_k)$ .

## 2.2 The VDSOPT algorithm

The VDSOPT algorithm is a polynomial algorithm that builds an optimal solution to this VDS scheduling problem, if the following condition H holds:

$$\forall i \in G, \min_{g \in \text{PRED}(i)} p_g \geq \max_{h \in \text{PRED}(i)-\{g\}} c_{h,i}$$

Basically, the condition H states that the processing times are superior or equal to the communication delays. Note that if condition H is not verified, then the VDS scheduling problem is NP-hard in most cases.

This algorithm may be considered as an extension of the Critical Path Method (C.P.M). It works in two steps.

First is a procedure VDSLWB that computes lower bounds of the starting times of any copy of each task. The second step is the procedure VDSSOL which uses a critical rooted tree to build up a schedule in which any copy of a task starts at its lower bound.

**Procedure VDSLWB can be stated as follows :**

For any task  $i$  such that  $\text{PRED}(i)=\emptyset$ , let its lower bound  $b_i$  be zero;

While there is a task  $i$  which has not been assigned a lower bound  $b_i$  and whose predecessors in  $\text{PRED}(i)$  all have lower bounds assigned do

Define

$$C = \max_{k \in \text{PRED}(i)} (b_k + p_k + c_{k,i})$$

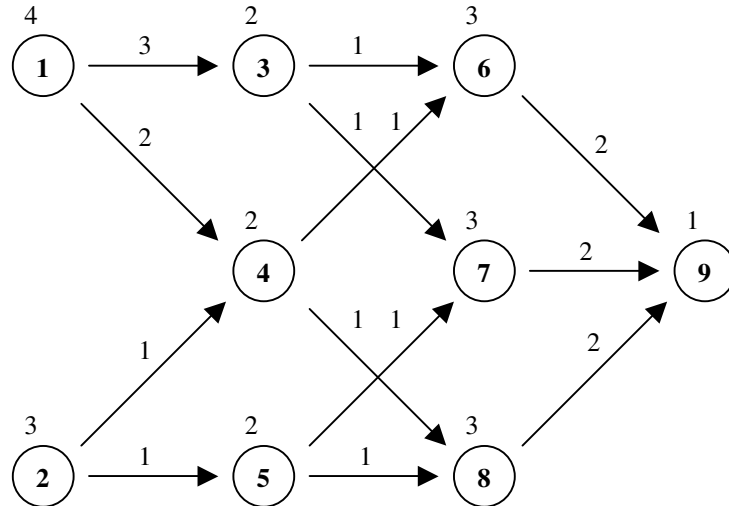
Let  $s$  be such that  $b_s + p_s + c_{s,i} = C$

Define the lower bound

$$b_i = \max ( b_s + p_s, \max_{k \in \text{PRED}(i)-\{s\}} (b_k + p_k + c_{k,i}) )$$

end while

**Example 2 :**



**Figure 2.** Example of a graph with communication delays:  
the value above each node  $i$  of the graph is the processing time  $p_i$ ,  
the value above each arc  $(i,j)$  of the graph is the communication delay  $c_{i,j}$ .

Figure 2 presents an example of task graph. The earliest execution dates computed by VDSLWB are given in Figure 3.

task $i$ :	1	2	3	4	5	6	7	8	9
lower bound $b_i$ :	0	0	4	4	3	7	6	6	11

**Figure 3.** The earliest execution dates of tasks from Figure 2.

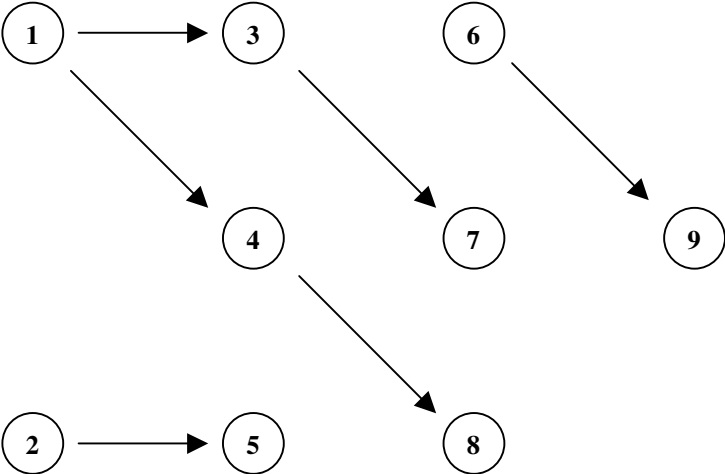
Following C.P.M. terminology, we define an arc  $(i,j)$  of  $U$  to be critical if  $b_i + p_i + c_{i,j} > b_j$ . From this definition, it is clear that if  $(i,j)$  is a critical arc, then in any earliest schedule every copy of task  $j$  must be preceded by a copy of task  $i$  executed on the same processor.

A critical sequence of the precedence graph is a path such that all of its arcs are critical and it is not a proper subpath of a critical sequence. The critical graph of the VDS scheduling problem is the subgraph of  $G$  induced by the critical arcs. The critical graph is always a forest, thus making the search of all critical sequences easy.

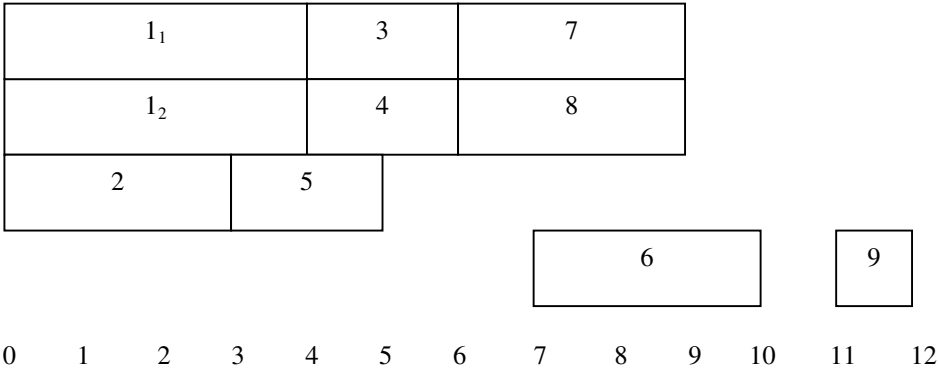
**Procedure VDSSOL, the second step of the algorithm, can then be stated as follows :**

- Assign each critical sequence of the critical graph to a distinct processor : one copy of each task belonging to a critical sequence must be executed on the processor assigned to this critical sequence;
- Execute each copy  $i_k$  of each task  $i$  at its lower bound  $b_i$ .

Figure 4 presents the critical subgraph extracted from figure 2 using the results of figure 3. The optimal schedule of this graph is shown in figure 5.



**Figure 4.** Critical graph extracted from figure 2. The critical sequences are (1,3,7), (1,4,8), (2,5) and (6,9).



**Figure 5.** The earliest schedule of copies of the tasks of figure 2. One processor is assigned to each critical sequence. }

The overall complexity of the whole algorithm is  $O(n^2)$ . Its proof can be found in [CoCh91].

The number of executed copies of a task is equal to the number of critical sequences the task belongs to. Note also that the algorithm does not necessarily minimize the number of processors used.

Finally, a more complex version of this algorithm is presented in [CoNa99b]. It solves the problem of scheduling tasks with communication delays on a multi-levels virtual distributed system.

### 3. The non deterministic problem

We now extend the above problem and suppose that the processing times of all tasks of graph  $G$  are now random, independent variables. We also suppose that the value  $p_i$  of task  $i$  takes a finite number of different values  $\{p_i^a, p_i^b, p_i^c, \dots\}$ , with a known probability for each possible value  $p_i^u$ .

Note that the the new condition (H) becomes then

$$\forall i \in I, \min\{p_g^u \mid g \in \text{PRED}(i), p_g^u \in \{p_g^a, p_g^b, p_g^c, \dots\}\} \geq \max\{c_{g,j} \mid g \in \text{PRED}(i)\}$$

that is, the communication times are smaller than or equal to the smallest processing times.

We begin with a few definitions.

We call *pre-scheduling* a partial solution that, to any task graph  $G$ ,

- for each task, states the number of copies to be executed on VDS ;
- for each copy, states which processor will execute it;
- for each processor, states in which order the copies will be processed.

Let  $X$  be a vector  $(p_1^u, p_2^v, \dots, p_n^z)$  of actual processing times (remember that  $n$  is the number of tasks in  $G$ ). Thus a given  $X$  defines an instance of a VDS scheduling problem.

Let now  $E^*$  be the set of all possible cases of  $X$ , i.e.  $E^* = \{X_1, X_2, \dots, X_N\}$ .  $N$  is finite because there is only a limited number of cases.

Let  $\alpha_i$  be the probability of  $X$  being  $X_i$  i.e.  $\alpha_i = P(X = X_i)$ .

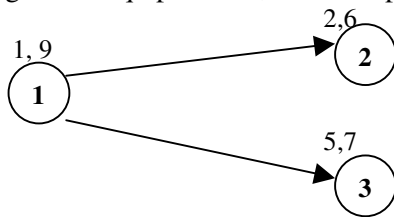
Let  $\text{avg}(X)$  be the average vector, i.e.  $\text{avg}(X) = \sum_{i=1,n} (\alpha_i X_i)$ .

**Lemma 1** : we have  $\text{avg}(X) = (\text{avg}(p_1), \text{avg}(p_2), \dots, \text{avg}(p_n))$   
 where  $\{p_i, 1 \leq i \leq n\}$  is the set of tasks processing times.

One can note that this average vector  $\text{avg}(X)$  does not always belong to  $E^*$ .

So let  $E$  be the set of all possible  $X_i$  with the average vector added, i.e.  $E = E^* \cup \{\text{avg}(X)\}$ .

**Example 3** : The next figure presents an example of a random task graph with 3 tasks, we assume that the task lengths are equiprobable, with independence between task length random variables.



We have :

$$E^* = \{(1,2,5), (1,2,7), (1,6,5), (1,6,7), \{(9,2,5), (9,2,7), (9,6,5), (9,6,7)\}\}$$

$$\text{avg}(X) = ((1+9)/2, (2+6)/2, (5+7)/2) = (5, 4, 6)$$

$$E = E^* \cup \{(5, 4, 6)\} = \{(1,2,5), (1,2,7), (1,6,5), (1,6,7), \{(9,2,5), (9,2,7), (9,6,5), (9,6,7), (5, 4, 6)\}\}$$

#### 3.1 Building a pre-scheduling

We denote  $A_X$  the pre-scheduling that results from the application of the VDSOPT algorithm to an instance  $X$  of the new problem. This pre-scheduling uses the solution of the VDSOPT algorithm applied to  $X$  in the following way :

- for each task, the number of copies to be executed on VDS is the number computed by VDSOPT,

- for each copy, the processor that will execute it is the processor allocated by VDSOPT,
- for each processor, the order of the copies strictly follows the order of these copies in the critical sequence allocated to this processor.

Note that because all the copies to be executed on a given processor belong to the same critical sequence in the solution computed by VDSOPT, the resulting pre-scheduling may always be executed. The reason is that no pre-scheduling built this way may have one copy of a task to be executed before any of its ancestors, direct or not, in the task graph

Now, let  $T_Y(A_X)$  be the optimal makespan of vector  $Y$  using the pre-scheduling  $A_X$ . That is,  $T_Y(A_X)$  is the makespan of the optimal solution build using the  $A_X$  pre-scheduling when the real vector of processing times is in fact  $Y$ . Note that the optimal solution build using the  $A_X$  pre-scheduling may easily be determined for each  $Y$  by simply executing each copy as soon as possible once the durations of its predecessors are known. Finally, let  $\text{avg}(T(A_X))$  be the average makespan of all possible vectors of  $E^*$  using the pre-scheduling  $A_X$ , ie.  $\text{avg}(T(A_X)) = \sum_{i=1,N} (\alpha_i \cdot T_{X_i}(A_X))$

### 3.2 Average makespan of any pre-scheduling

We first try to see if it is possible to compute the average performance of any pre-scheduling.

#### Theorem 1 :

If condition (H) is verified, then for each vector  $Y$  of  $E$ , the optimal makespan of the average vector  $\text{avg}(X)$  using the pre-scheduling  $A_Y$  is not greater than the average of all possible optimal executions using  $A_Y$ , i.e.

$$\forall Y \in E, \text{ we have } T_{\text{avg}(X)}(A_Y) \leq \text{avg}(T(A_Y))$$

Thus, for each pre-scheduling we got using the VDSOPT algorithm, it is possible to compute a lower bound of its average makespan. This lower bound is computed using the pre-scheduling with the average vector  $\text{avg}(X)$  as real vector.

Basically, this result states, that, for DAG with communication delays, the classical method of using the average task graph to estimate the average performance of any pre-scheduling yields a result that is never pessimistic, and tends to be too optimistic.

### 3.3 Bounding the average makespan of optimal pre-schedulings

Let  $Z$  be a vector of  $E$ , that gives the best pre-scheduling, i.e.  $Z$  is such that  $\text{avg}(T(A_Z))$  is minimal.

First note that finding this  $Z$ , or any best pre-scheduling for that matter, is a very difficult problem. This is because if any algorithm could find it efficiently for any DAG with communication delays, it also could be used to find the best pre-scheduling for any DAG without communication delays, a problem that is itself known to be very difficult.

Now let us state our second result.

#### Theorem 2 :

If condition (H) is verified, then we have

$$T_{\text{avg}(X)}(A_{\text{avg}(X)}) \leq \text{avg}(T(A_Z)) \leq \text{avg}(T(A_{\text{avg}(X)}))$$

In other words :

- the average makespan of the best pre-scheduling is not inferior to the makespan of the pre-scheduling build by VDSOPT using the average vector  $\text{avg}(X)$ , when the real vector is the average vector  $\text{avg}(X)$ ,
- this average makespan of the best pre-scheduling is not greater than the average makespan of the pre-scheduling build using  $\text{avg}(X)$ .

Note that :

- $\text{Avg}(X)$  can be easily computed (see lemma 1)
- $T_{\text{avg}(X)}(A_{\text{avg}(X)})$  can efficiently be computed by applying VDSOPT algorithm on  $\text{avg}(X)$ , thus giving a lower bound of the best pre-scheduling with  $O(n^2)$  complexity (c.f. 2.2 ).

Again , this results means that, for DAG with communication delays, the classical method of using the average DAG as a basis for pre-scheduling, will give an estimate of the average performance of the best pre-scheduling that is never pessimistic, and tends to be too optimistic.

## 4. Conclusions

In this paper, we considered the problem of scheduling tasks with fixed small communications delays on a virtual distributed memory multiprocessor when the tasks execution times are random. This problem extends the classical PERT scheduling problem with random execution times and no communications. We first presented how to efficiently build pre-schedulings for this problem, using the optimal VDSOPT scheduling algorithm for deterministic VDS scheduling problems. We then computed a lower bound of the average makespan of a given pre-scheduling. We finally proposed a lower and an upper bound of the execution time of the best pre-scheduling. The lower bound is efficiently computed by using the VDSOPT algorithm on the average vector  $\text{avg}(X)$ . The upper bound we propose is the average execution time of the pre-scheduling computed by VDSOPT using  $\text{avg}(X)$ . It is hard to compute however, and finding a good upper bound is still an open problem.

## References

- [Ful62] Fulkerson, Expected Critical Path Lengths in PERT Networks, *Opns Res.* 10, 1962, 808-817.
- [Bod85] B. Bodin, Bounding the project completion time distribution in pert networks, *Opns Res*, v. 33, 1985.
- [ColCh91] J-Y Colin and P. Chrétienne. C.P.M. scheduling with small communication delays and task duplication, *Operations Research*, 39:681-684, 1991
- [CoNa99a] J-Y Colin, M. Nakechbandi Scheduling Tasks with communication delays on a tow level virtuel disruted system, *7th Euromicro Workshop on Parallel and distributed Processing*, University of Madeira, Funchal, Portugal, February 3rd-5th 1999
- [CoNa99b] J.-Y. Colin, P. Colin, M. Nakechbandi, and F. Guinand, Scheduling Tasks with communication Delays on Multi-Levels Clusters, *PDPTA'99 Parallel and Distributed Techniques and Application*, June 1999, Las Vegas, U.S.A.
- [Elm77] Elmaghraby S.E. Activity networks :project planning andcontrol by networks models, *Wiley N.Y.* 1977.
- [FiLi95] L. Finta, Z. LIU Makespan minimization of task graph with random task running times, *Discrete Mathematics and Computer Science*,V.21, 1995.
- [FrGr85] A. M. Frieze and G. R. Grimmet, The shortest-path problem for graphs with random arc-lengths', *Discrete Applied Mathematics*, 10, 1985.