A multi-valued DAG model and an optimal PERT-like Algorithm for the Distribution of Applications on Heterogeneous Computing Systems

Jean-Yves Colin, Moustafa Nakechbandi, Patrice Colin

Communication PDPTA'05 : The 2005 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, USA, pp 876-882 (2005)..

A multi-valued DAG model and an optimal PERT-like Algorithm for the Distribution of Applications on Heterogeneous Computing Systems

J.-Y. Colin , M. Nakechbandi, P. Colin

LIH - Laboratoire d'Informatique du Havre, UFR Sciences et Techniques,
25 rue Ph. Lebon - BP 540, 76058 Le Havre Cedex - FRANCE
email : jean-yves.colin@univ-lehavre.fr, moustafa.nakechbandi@univ-lehavre.fr

Abstract

In this paper, we study the following theoretical scheduling problem: the tasks of a given application have to be statically distributed and executed on the heterogeneous servers of a multi-user system on a wide area, heterogeneous network, such as Internet. The application is divided into a set of communicating tasks that may be executed on at least one, and possibly several, servers. The processing time of each task depends on the server processing it, and the communication delay between two tasks depends on the communicating servers. Task duplication is allowed. We propose an efficient algorithm that first uses a PERT-like algorithm to compute the earliest execution dates of each task and then build an optimal static solution for this scheduling problem, with a low number of tasks duplications.

Keywords: Wide-Area Distributed System, Communication Delays, Scheduling, Critical-Path Method, Minimal Makespan.

1 Introduction

The development of geographically distributed systems, also known as meta-computing systems, or wide-area systems, or computational grids, presents new opportunities. A growing number of applications try to use the offered computational power. Some of the current ones include Automated Document Factories (ADF) in banking environments where several hundred thousands documents are produced each day on networks of several multiprocessors servers, or high performance Data Mining (DM) systems [10] or Grid Computing [9, 11]. However, using efficiently these heterogeneous systems is a hard problem.

When the application tasks can be represented by Directed Acyclic Graphs (DAGs), many dynamic scheduling algorithms have been devised. For some examples, see [2, 3, 7]. Also, several static algorithms for scheduling DAGs in metacomputing systems are described in [4, 6, 13]. Most of them suppose that tasks competes for limited processor resources, and thus these algorithms are mostly heuristics, in [5] is presented an optimal polynomial algorithm that schedules the tasks and communications of an application on a Virtual Distributed System with several clusters levels, although, in [8] we studied the static scheduling problem where the tasks execution times are positive independent random variables, and the communication delays between the tasks are perfectly known.

In the first part of this paper, we present the following theoretical scheduling problem: the tasks of a given application have to be statically distributed and executed on the heterogeneous servers of a multi-users system on a wide area, heterogeneous network, such as Internet. In the following, we call this theoretical architecture a Distributed Servers System (DSS). The application is divided into a set of communicating tasks that may be executed on at least one, and possibly several servers of a Distributed Servers System. The processing time of each task depends on the server processing it, and the communication delay between two tasks depends on the tasks and on the communicating servers. Task duplication, that is the execution of the same task on several servers, is allowed. This application is represented by an extended Directed Acyclic Graph.

In the second part of this paper, we present DSS_OPT. This algorithm is a polynomial algorithm that computes the earliest execution date of each task on each server of the DSS, and then uses these dates to build an optimal static solution that schedules the tasks on the servers of the DSS. Although task duplication is allowed, the solution found uses a low number of executions of the same task on different servers.

Finally, we discuss the validity of our hypothesis and conclude with some remarks on the DSS scheduling problem and on the DSS_OPT algorithm in the last part of our paper.

2 The data of the central problem

In this part, we first define what we call a Distributed Servers System. We then present the formal definition of our scheduling problem. Next, we define what constitutes a feasible solution. Finally, we state our optimality conditions.

2.1 The Distributed Servers System

We call Distributed Servers System (DSS) a virtual set of geographically distributed, multi-users, heterogeneous or not, servers. Therefore, a DSS has the following properties:
First, the processing time of a task on a DSS may vary from a server to another. This may be due to the processing power available on each server of the DSS for example. The processing time of each task on each server is supposedly known.

Second, although it may be possible that some servers of a DSS are potentially able to execute all the tasks of an application, it may also be possible in some applications that some tasks may not be executed by all servers. This could be due to the fact that specific hardware is needed to process these tasks and that this hardware is not available on some servers. Or it could be that some specific data needed to compute these tasks are not available on these servers for some reason. Or it could be that some user input is needed and the user is only located in a geographically specific place. Obviously, in our problem we suppose that the needs of each task of an application are known, and that at least one server of the DSS may process it, else there is no possible solution to the scheduling problem.

Furthermore, an important hypothesis is that the concurrent executions of some tasks of the application on a server have a negligible effect on the processing time of any other task of the application on the same server. Although apparently far-fetched, this hypothesis may hold if the server is a multiprocessors architecture with enough processors to simultaneously execute all the tasks of the application that are to be processed concurrently. Or it may be that the server is a time-shared, multi-user system with a permanent heavy load coming from other applications, and the tasks of an application on this server represent a negligible additional load compared to the rest.

In addition, in the network interconnecting the servers of a DSS, the transmission delay of a result between two tasks varies depending on the tasks and on their respective sites.

Again, we suppose that concurrent communications between tasks of the same application on two servers have a negligible effect on the communication delays between two others tasks located on the same two servers. This hypothesis may hold if the network already has a permanent heavy load due to other applications, and the communications of the application represent a negligible additional load compared to the one already present. Figure 1 presents un example of a DSS.

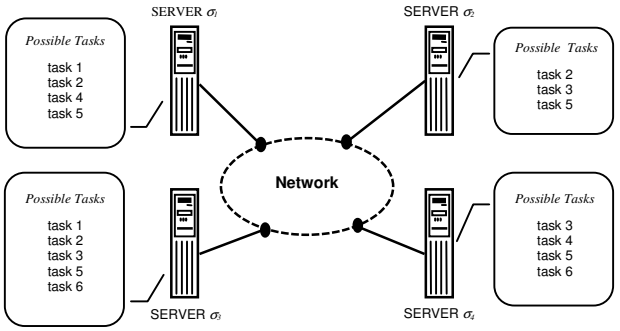

Figure 1: Example of Distributed Servers System

2.2 Directed Acyclic Graph

We now describe the application itself in our problem. An application is decomposed into a set of indivisible tasks that have to be processed. A task may need data or results from other tasks to fulfil its function and then send its results to other tasks. The transfers of data between the tasks introduce dependencies between them. The resulting dependencies form a Directed Acyclic Graph.

Because the servers are not necessarily identical, the processing time of a given task can vary from one server to the next.

Furthermore, the duration of the transfer of a result on the network cannot be ignored. This communication delay is function of the size of the data to be transferred and of the transmission speed that the network can provide between the involved servers. Note that if two dependent tasks are processed themselves on the same server, this communication delay is considered to be 0.

The central scheduling problem *P* on a Distributed Server System, is represented therefore by the following parameters:

- a set of servers, noted $\Sigma = \{\sigma_1, ..., \sigma_s\}$, interconnected by a network,
- a set of the tasks of the application, noted I = {1,..., n}, to be executed on Σ. The execution of task i, i ∈ I, on server σ_r, $\sigma_r \in \Sigma$, is noted i/σ_r. The subset of the servers able to process task i is noted Σ_i, and may be different from Σ,
- the processing times of each task *i* on a server σ_r is a positive value noted π_{i/σ_r}. The set of processing times of a given task *i* on all servers of Σ is noted $\Pi_i(\Sigma)$. $\pi_{i/\sigma_r} = \infty$ means that the task *i* cannot be executed by the server σ_r.
- a set of the transmissions between the tasks of the application, noted U. The transmission of a result of an task *i*, *i* ∈ I, toward a task *j*, *j* ∈ I, is noted (i, j). It is supposed in the following that the tasks are numbered so that if (*i, j*) ∈ U, then i < j,
- the communication delays of the transmission of the result (i, j) for a task i processed by server σ_r toward a task j processed by server σ_p is a positive value noted $c_{i/\sigma_r, j/\sigma_p}$. The set of all possible communication delays of the transmission of the result of task i, toward task j is noted $\Delta_{i,j}(\Sigma)$. Note that a zero in $\Delta_{i,j}(\Sigma)$ mean that i and j are on the same server, i.e. $c_{i/\sigma_r, j/\sigma_p} = 0 \Rightarrow \sigma_r = \sigma_p$. And $c_{i/\sigma_r, j/\sigma_p} = \infty$ means that either task *i* cannot be executed by server σ_r, or task *j* cannot be executed by server σ_p, or both.

Let $\Pi(\Sigma) = \bigcup_{i \in I} \Pi_i(\Sigma)$ be the set of all processing times of the tasks of *P* on Σ.

Let $\Delta(\Sigma) = \bigcup_{(i,j) \in U} \Delta_{i,j}(\Sigma)$ be the set of all communication delays of transmissions (i, j) on Σ.

The central scheduling problem *P* on a distributed servers system DSS can be modelled by a multi-valued DAG $G = \{I, U, \Pi(\Sigma), \Delta(\Sigma)\}$. In this case we note $P=\{G, \Sigma\}$. Figure 2 is an example with six tasks:

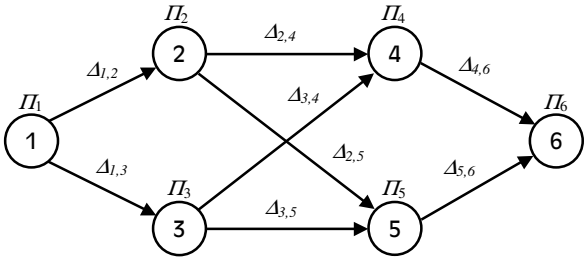

Figure 2: Example of a multi-valued DAG

On four servers we have $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$. If $\Pi_1 = (3, \infty, 2, \infty)$, then the execution time of task 1 on server 1 is $\pi_{1/\sigma_1} = 3$, and the execution time of the same task on server 2 is $\pi_{1/\sigma_2} = \infty$ that is, the server 2 is not able to execute task 1. If the communications between task 1 and task 2 are represented by the following matrix $\Delta_{1,2}$ (Table 1):

	σ_1	σ_2	σ_3	σ_4
σ_1	0	3	2	∞
σ_2	∞	∞	∞	∞
σ_3	2	**3**	0	∞
σ_4	∞	∞	∞	∞

Table 1: Example of a communication matrix between the tasks 1 and 2

Then, if task 1 is executed on server σ_3 and must send its result to task 2 on server σ_2 the communication delay $c_{1/\sigma_3, 2/\sigma_2}$ will be 3.

2.3 Definition of a feasible solution

We note PRED(*i*), the set of the predecessors of task *i* in *G*:

$$\text{PRED}(i) = \{ k / k \in I \text{ et } (k,i) \in U \}$$

And we note SUCC(*i*), the set of the successors of task *i* in *G* :

$$\text{SUCC}(i) = \{ j / j \in I \text{ et } (i,j) \in U \}$$

A feasible solution S for the problem P is a subset of executions $\{ i/\sigma_r , i \in I \}$ with the following properties:

- each task *i* of the application is executed at least once on at least one server σ_r of Σ_i,
- to each task *i* of the application executed by a server σ_r of Σ_i, is associated one positive execution date t_{i/σ_r} ,
- for each execution of a task *i* on a server σ_r, such that PRED(*i*) ≠ ∅, there is at least an execution of a task *k*, *k* ∈ PRED(*i*), on a server

σ_p, $\sigma_p \in \Sigma_\kappa$, that can transmit its result to server σ_r before the execution date t_{i/σ_r}.

The last condition, also known as the Generalized Precedence Constraint (GPC) [5], can be expressed more formally as:

$$\forall i/\sigma_r \in S \begin{cases} t_{i/\sigma_r} \geq 0 & \text{if PRED}(i) = \varnothing \\ \forall k \in \text{PRED}(i), \exists \sigma_p \in \Sigma_k / t_{i/\sigma_r} \geq t_{k/\sigma_p} + \pi_{k/\sigma_p} + c_{k/\sigma_p,i/\sigma_r} & \text{else} \end{cases}$$

It means that if a communication must be done between two scheduled tasks, there is at least one execution of the first task on a server with enough delay between the end of this task and the beginning of the second one for the communication to take place.

A feasible solution *S* for the problem *P* is therefore a set of executions i/σ_r of all i tasks, $i \in I$, scheduled at their dates t_{i/σ_r}, and verifying the Generalised Precedence Constraints GPC.

Note that, in a feasible solution, several servers may simultaneously or not execute the same task. This may be useful to generate less communications. All the executed tasks in this feasible solution, however, must respect the Generalized Dependence Constraints.

2.4 Optimality Condition

Let *T* be the total processing time of an application (also known as the makespan of the application) in a feasible solution *S*, with *T* defined as

$$T = \max_{i/\sigma_r \in S} (t_{i/\sigma_r} + \pi_{i/\sigma_r})$$

A feasible solution *S** of the problem *P* modelled by a DAG $G = \{I, U, \Pi(\Sigma), \Delta(\Sigma)\}$ is optimal if its total processing time *T** is minimal. That is, it does not exist any feasible solution *S* with a total processing time *T* such that $T < T^*$.

3 The DSS_OPT Algorithm

3.1 Presentation

Let *P* be a DSS scheduling problem, and let $G = \{I, U, \Pi(\Sigma), \Delta(\Sigma)\}$ be its DAG.

One can first note that there is an optimal trivial solution to this DSS scheduling problem. In this trivial solution, all possible tasks are executed on all possible servers, and their results are then broadcasted to all other tasks that may need them on all others servers. This is an obvious waste of processing power and communication resources, however, and something better and more efficient is usually needed.

So, we now present DSS_OPT(*P*), a new polynomial algorithm that builds an optimal solution for problem *P*.

DSS_OPT has two phases.

The first phase, DSS_LWB(*P*), computes the earliest feasible execution dates b_{i/σ_r} for all possible executions i/σ_r of each task *i* of problem *P*.

The second phase determines, for every task i that does not have any successor in *P*, the execution i/σ_r ending at the earliest possible date b_{i/σ_r}. If several executions of task i end at the same smallest date b_{i/σ_r}, one is chosen, arbitrarily or using other criteria of convenience, and kept in the solution. Then, for each kept execution i/σ_r that has at least one predecessor in the application, the subset L_i of the executions of its predecessors that satisfy GPC(i/σ_r) is established. This subset of executions of predecessors of i contains at least an execution of each of its predecessors in G. One execution k/σ_p of every predecessor task k of task i is chosen in the subset, arbitrarily or using other criteria of convenience, and kept in the solution. It is executed at date b_{k/σ_p}. The examination of the predecessors is pursued in a recursive manner until the studied tasks do not present any predecessors in G. The complete algorithm is the following

DSS_OPT (*P*)
1: DSS_LWB (*P*)
// first phase
2: $T = \max_{\forall i/\text{SUCC}(i)=\varnothing} \min_{\forall \sigma_r \in \Sigma_i} (r_{i/\sigma_r})$
3: for all tasks *i* such that SUCC(*i*) = ∅
// second phase
4: $L_i \leftarrow \{ i/\sigma_r / \sigma_r \in \Sigma_i \text{ and } r_{i/\sigma_r} \leq T \}$
5: $i/\sigma_r \leftarrow$ keepOnefrom(L_i)
6: schedule (i/σ_r)
end DSS_OPT

DSS_LWB(*P*)
1: **For** each task i where PRED(*i*) = ∅ **do**
2: **for each** server σ_r such that $\sigma_r \in \Sigma_i$ **do**
3: $b_{i/\sigma_r} \leftarrow 0$
4: $r_{i/\sigma_r} \leftarrow \pi_{i/\sigma_r}$
end for
5: mark (*i*)
end for

6: **while** there is a non marked task *i* such that
all its predecessors *k* in *G* are marked **do**

7: **for** each server σ_r such that $\sigma_r \in \Sigma_i$ **do**
$b_{i/\sigma_r} \leftarrow \max_{\forall k \in \mathrm{PRED}(i)} \min_{\forall \sigma_p \in \Sigma_k} (b_{k/\sigma_p} + \pi_{k/\sigma_p} + c_{k/\sigma_p, i/\sigma_r})$

9: $r_{i/\sigma_r} \leftarrow b_{i/\sigma_r} + \pi_{i/\sigma_r}$
end for

10: mark (*i*)
end while

end DSS_LWB(*P*)

schedule(i/σ_r)
1:execute the task *i* at the date b_{i/σ_r} on the server σ_r
2: **if** PRED(*i*) ≠ ∅ then
3: **for** each task *k* such that $k \in$ PRED(*i*) **do**
4: $L_k^{i/\sigma_r} \leftarrow \{ k/\sigma_q \ / \ \sigma_p \in \Sigma_\kappa$ and
$b_{k/\sigma_p} + \pi_{k/\sigma_p} + c_{k/\sigma_p, i/\sigma_r} \le b_{i/\sigma_r} \}$
5: $k/\sigma_q \leftarrow$ keepOneFrom(L_k^{i/σ_r})
6: schedule (k/σ_q)
end for
end if
end schedule

keepOneFrom(L_i)
return an execution i/σ_r of task i in the list of the executions L_i.
end keepOneFrom.

3.2 Example of application

Let *P* be a DSS scheduling problem, and its DAG G be the following DAG

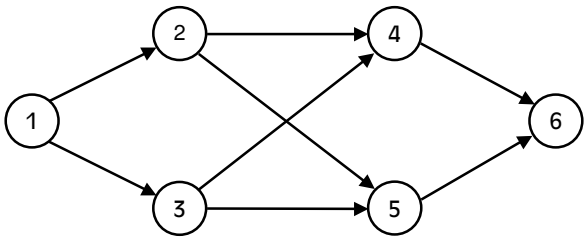

Figure 3: DAG of the problem P

Thus, the *P* problem has 6 distributed tasks, *I = {1, 2, 3, 4, 5, 6}*, we also suppose that four servers are available, $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, with the following processing times (*Table 2)*:

π_{i/σ_r}	σ_1	σ_2	σ_3	σ_4
1	3	∞	2	∞
2	2	4	3	∞
3	∞	2	2	2
4	4	∞	∞	2
5	2	3	2	4
6	∞	∞	2	3

Table 2: Processing times matrix for the problem P

And with the following communication delays (*Table 3)*:

(1, 2)	σ_1	σ_2	σ_3	σ_4
σ_1	0	3	2	∞
σ_2	∞	∞	∞	∞
σ_3	2	3	0	∞
σ_4	∞	∞	∞	∞

(1, 3)	σ_1	σ_2	σ_3	σ_4
σ_1	∞	2	4	1
σ_2	∞	∞	∞	∞
σ_3	∞	2	0	3
σ_4	∞	∞	∞	∞

(2, 4)	σ_1	σ_2	σ_3	σ_4
σ_1	0	∞	∞	3
σ_2	1	∞	∞	2
σ_3	3	∞	∞	3
σ_4	∞	∞	∞	∞

(2, 5)	σ_1	σ_2	σ_3	σ_4
σ_1	0	2	2	2
σ_2	3	0	1	2
σ_3	1	2	0	3
σ_4	∞	∞	∞	∞

(3, 4)	σ_1	σ_2	σ_3	σ_4
σ_1	0	∞	∞	∞
σ_2	2	∞	∞	1
σ_3	3	∞	∞	2
σ_4	1	∞	∞	0

(3, 5)	σ_1	σ_2	σ_3	σ_4
σ_1	∞	∞	∞	∞
σ_2	2	0	2	2
σ_3	3	1	0	4
σ_4	2	1	1	0

(4, 6)	σ_1	σ_2	σ_3	σ_4
σ_1	∞	∞	1	2
σ_2	∞	∞	∞	∞
σ_3	∞	∞	∞	∞
σ_4	∞	∞	2	0

(5, 6)	σ_1	σ_2	σ_3	σ_4
σ_1	∞	∞	3	2
σ_2	∞	∞	2	2
σ_3	∞	∞	0	1
σ_4	∞	∞	1	0

Table 3: Complete communication times matrix for the problem P

For example for the arc (5, 6), if the task 5 is executed on the server σ_1 and must send its result to the task 6 on the server σ_3 the communication delay $c_{5/\sigma_1, 6/\sigma_3}$ will be 3.

The algorithm uses DSS_LWB to compute the earliest possible execution date of all tasks on all possible servers, resulting in the following values *b* and *r* (*Table 4)*:

1	b_1	r_1
σ_1	0	3
σ_2	∞	∞
σ_3	0	2
σ_4	∞	∞

2	b_2	r_2
σ_1	3	5
σ_2	5	9
σ_3	2	5
σ_4	∞	∞

3	b_3	r_3
σ_1	∞	∞
σ_2	4	6
σ_3	2	4
σ_4	4	6

4	b_4	r_4
σ_1	7	11
σ_2	∞	∞
σ_3	∞	∞
σ_4	8	10

5	b_5	r_5
σ_1	7	9
σ_2	7	10
σ_3	5	7
σ_4	7	11

6	b_6	r_6
σ_1	∞	∞
σ_2	∞	∞
σ_3	12	14
σ_4	10	13

Table 4: The earliest possible execution date of all tasks on all possible servers for the problem P

It then computes the smallest makespan of any solution to the P problem :

$$T = \max_{\forall i/\mathrm{SUCC}(i)=\varnothing \forall \sigma_r \in \Sigma_i} \min (r_{i/\sigma_r}) = \max\min(r_{6/\sigma_3}, r_{6/\sigma_4}) = 13$$

In the example, only task 6 does not have any successor. The list L_6 of the executions kept for this task in the solution is reduced therefore to the execution $6/\sigma_4$. Thus $L_6 = \{6/\sigma_4\}$. The execution of task 6 on the server σ_4 is scheduled at date 10.

Next, The tasks 4 and 5 are the predecessors in G of task 6. For the task 4, only the execution $4/\sigma_4$ may satisfy the Generalised Precedence Constraints relative to $6/\sigma_4$. Therefore, this execution is kept and is scheduled at date b_{4/σ_4}. For task 5, execution $5/\sigma_3$ is kept and is scheduled at date b_{5/σ_3} …

The table 5 presents the final executions i/σ_r kept by the DSS_OPT(P) algorithm, with their date of execution, in an optimal solution S.

	$1/\sigma_3$	$2/\sigma_3$	$3/\sigma_3$	$4/\sigma_4$	$5/\sigma_3$	$6/\sigma_4$
b_{i/σ_r}	0	2	2	8	5	10
r_{i/σ_r}	2	5	4	10	7	13

Table 5: final executions i/σ_r kept by the DSS_OPT(P) algorithm

Finally, we obtain (figure 4) the following optimal scheduling :

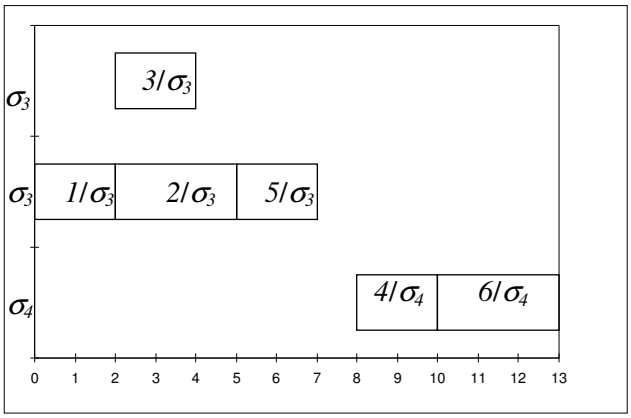

figure 4: An optimal scheduling S for the problem P*

3.3 Complexity

The most computationally intensive part of DSS_OPT(*P*) is the first part DSS_LWB(*P*). In this part, for each task *i*, for each server executing *i*, for each predecessor *j* of *i*, for each server executing *j*, a small computation is done. Thus the complexity of DSS_LWB(P) is $O(n^2s^2)$, where *n* is the number of tasks in *P*, and *s* is the number of servers in DSS.

Thus, the global complexity of the DSS_OPT(*P*) algorithm is $O(n^2s^2)$.

4 Discussion

As usual in all PERT or critical-path methods, the various processing times and communications delays of each task on each server are supposedly known. While these processing times and communication delays may easily be determined in some numerical applications, they may be much harder to estimate in others. This is a well-known problem of all PERT methods, and various means must sometimes be used to get estimates of these data [12].

Furthermore, an important hypothesis in our problem is that the concurrent executions of some tasks of the application on a server have none or a negligible effect on the processing time of any other task of the application on the same server. Although apparently far-fetched, this hypothesis may hold if the server is a multiprocessors architecture with enough processors to simultaneously execute all the tasks that are to be processed concurrently. Or it may be that the server is a time-shared, multi-user system with a permanent heavy load coming from other applications, and the tasks of an application on this server represent a negligible additional load compared to the rest. If this is not the case, then this hypothesis is still similar to the non limited number of available processors hypothesis present in all classical PERT problems. And as in the classical PERT problems, our earliest execution dates of each task may be used as priority values to build priority lists for list scheduling algorithms or heuristics.

Also, as already noted when introducing DSS_OPT, there is an optimal trivial solution to the DSS scheduling problem. In this trivial solution, all possible tasks are executed on all possible servers, and their results are then broadcasted to all other tasks that may need them on all others servers. For any real application, with many tasks and communications, this is a tremendous waste of processing power and communication resources, however. By contrast, our solution has the same total execution time but uses a much more limited number of tasks duplication, if any.

Additionally, one can note that the DSS_LWB(P) part itself is an extension of the VDS_OPT algorithm [1, 5], that are themselves extensions of the classical PERT algorithm to DAGs with communication delays. However, the hard condition that processing times must be superior or equal to communication delays in the VDS_OPT problem for the problem to be computationally tractable, even with a non limited number of processors, does not hold in the problem studied here. The reason is that we suppose that

several tasks can concurrently be executed on the same server with no effects, or with negligible effects on their processing times.

On a different aspect, note that the selection of the task to be kept from the list L_i of possible tasks is not attached to a particular policy or strategy. That is, any choice, even a random one, is possible in this list and will still result in an optimal solution. Thus, it is possible to use a more sophisticated policy to try to minimize a second criteria, if one is present. For example, if a money cost is associated to the execution of a given task on a given server, then a good choice will try to choose cheap executions to pay the smallest total execution price that still gives the minimal global execution time. The analysis of such multi-criteria problems needs more work however.

Finally, it may also be possible to improve the fault-tolerant aspects of a solution by keeping more tasks than necessary in the final solution, so as to have back-up tasks on other servers if the efficient ones fail. Worst-cases scenarios may then be studied and their additional costs and resulting loss of performances evaluated. Again, more work is needed on this subject.

5 Conclusions

In this paper, we studied the problem of scheduling the tasks of an application on the many heterogeneous servers of a multi-users system distributed on a heterogeneous network. The application is divided into communicating tasks that can be executed on at least one, and possibly several servers, with variable processing times depending on the chosen servers, and variable communication delays depending on the servers that communicate. We proposed an efficient algorithm that uses an extended DAG to build a static solution with a minimal makespan to this scheduling problem, with minimal number of task duplication or without task duplication at all. In our future work, we intend to study further both the multi-criteria problem and also the fault tolerant aspects evoked in the discussion part.

References

[1] J.-Y. Colin and P. Colin, “Scheduling tasks and communications on a virtual distributed system”, European Journal of Operational Research, Vol: 94, Issue:2, October 25, 1996.

[2] M. Maheswaran and H. J. Siegel, “A Dynamic matching and scheduling algorithm for heterogeneous computing systems”, Proceedings of the 7th IEEE Heterogeneous Computing Workshop(HCW '98), pp. 57-69, Orlando, Florida 1998.

[3] M. Iverson, F. Özgüner, “Dynamic, Competitive Scheduling of Multible DAGs in a Distributes Heterogeneous Environment”, Proceedings of the 7th IEEE Heterogeneous Computing Workshop (HCW '98), pp. 70 – 78, Orlando, Florida 1998.

[4] H. Topcuoglu, S. Hariri, and M.-Y. Wu., “Task scheduling algorithms for heterogeneous processors”. In 8th Heterogeneous Computing Workshop (HCW’ 99), pages 3–14, April 1999.

[5] J.-Y. Colin , M. Nakechbandi, P. Colin, F. Guinand, “Scheduling Tasks with communication Delays on Multi-Levels Clusters”, PDPTA'99 : Parallel and Distributed Techniques and Application, Las Vegas, U.S.A., June 1999.

[6] A. H. Alhusaini, V. K. Prasanna, C.S. Raghavendra, “A Unified Resource Scheduling Framework for Heterogeneous, Computing Environments”, Proceedings of the 8th IEEE Heterogeneous Computing Workshop, Puerto Rico, 1999, pp.156- 166.

[7] H. Chen, M. Maheswaran, “Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems”, Proceedings of the 11th IEEE Heterogeneous Computing Workshop, p. 88b-98b, Fort Lauderdale, 2002.

[8] M. Nakechbandi, J.-Y. Colin , C. Delaruelle, “Bounding the makespan of best pre-scheduling of task graphs with fixed communication delays and random execution times on a virtual distributed system”, OPODIS02, Reims, December 2002.

[9] Christoph Ruffner, Pedro José Marrón, Kurt Rothermel, “An Enhanced Application Model for Scheduling in Grid Environments”, TR-2003-01, University of Stuttgart, Institute of Parallel and Distributed Systems (IPVS), 2003.

[10] P. Palmerini, “On performance of data mining: from algorithms to management systems for data exploration”, PhD. Thesis: TD-2004-2, Universit`a Ca’Foscari di Venezia,2004.

[11] Srikumar Venugopal, Rajkumar Buyya and Lyle Winton, “A Grid Task Broker for Scheduling Distributed Data-Oriented Applications on Global Grids”, Technical Report, GRIDS-TR-2004-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, February 2004.

[12] Salah Elmagrgraby, “The Teory of networks and Management” Part II , Management Science Vol. 17, October 1970, pages B54-71.

[13] Yu-Kwong Kwok, and Ishfaq Ahmad, “Static scheduling algorithms for allocating directed task graphs to multiprocessors”, ACM Computing Surveys (CSUR), 31 (4): 406 - 471, 1999.