An efficient fault-tolerant scheduling algorithm for precedence constrained tasks in heterogeneous distributed systems

M. Nakechbandi[*], J.-Y. Colin[*], J.B. Gashumba[**]

LITIS Université du Havre, 25 rue Philippe Lebon, BP 540, 76058, Le Havre cedex, France
[(*)]{moustafa.nakechbandi, jean-yves.colin}@univ-lehavre.fr
[(**)] jb.gashumba@free.fr

Abstract

In this paper, we propose an efficient scheduling algorithm for problems in which tasks with precedence constraints and communication delays have to be scheduled on an heterogeneous distributed system with one fault hypothesis. Our algorithm combines the DSS_OPT algorithm and the eFRCD algorithm. To provide a fault-tolerant capability, we employ primary and backup copies. In this scheme, backup copies can overlap other backup copies, and backup copies are not established for tasks that have more than one primary copy. The result is a schedule in polynomial time that is optimal when there is no failure, and is a good resilient schedule in the case of one failure.

Keywords: Scheduling, Heterogeneous Distributed System, Fault-tolerance, Precedence Constraints, Communication Delays, Critical-Path Method.

I. INTRODUCTION

Heterogeneous distributed systems have been increasingly used for scientific and commercial applications, 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, because the general problem of optimal scheduling of tasks is NP-complete [13].

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 meta-computing systems are described in [1, 4, 6, 13]. Most of them suppose that tasks compete 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] is studied a static scheduling problem where the tasks execution times are positive independent random variables, and the communication delays between the tasks are perfectly known.

This review shows that a lot of works has been done concerning heterogeneous distributed scheduling. However, the problems with fault tolerant aspect are less studied. Reliable execution of a set of tasks is usually achieved by task duplication and backup copy [16, 17, 18]. Inspired with the works of Xiao Qin, Hong Jiang on Real-Time Heterogenous Systems [16], we propose in this article a good algorithm to solve the problem of one fault tolerant system using tasks duplication and backup copies technics.

This paper has three main parts: In the first one, we present the problem, in the second part, we present a solution to the problem, and in the third part, we discuss the advantages and disadvantages of the proposed solution.

II. THE CENTRAL PROBLEM

In this part, we present the following problem : Given an application T, we want to build an off-line scheduling of the tasks of this application in an heterogeneous distributed System with some possibilities of limited failures. We suppose that only one permanent failure can occur without any possibility of a temporary failure. This constitute one hypothesis which we call "1-failure". Note that the processing time of the application has to be minimized.

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 supposed 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.

T. Sobh (ed.), Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, 301–307.

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.

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, $i \in I$, 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, $i \in I$, toward a task j, $j \in I$, is noted (i, j). It is supposed in the following that the tasks are numbered so that if $(i, j) \in U$, 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 $P=\{G, \Sigma\}$.

Example: In the following example we consider a problem P of an application with nine tasks that has to be processed by a set of 3 servers on an heterogeneous distributed system. The architecture of distributed system and the task graph of this application are represented by the figure 1 and figure 2. It is supposed that the 3 servers are distributed on two different networks.

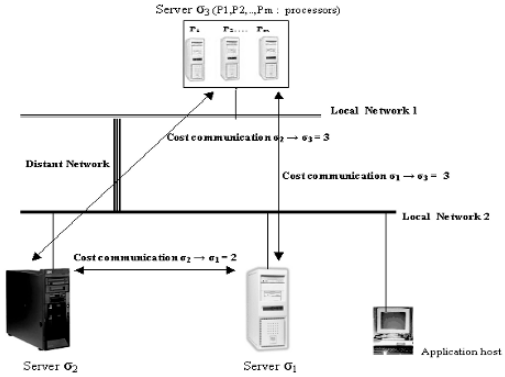

***Figure 1**: Distributed system architecture for the application example.*

The Matrix cost communication of our example is presented in table 1.

Network delay between $\sigma_i \rightarrow \sigma_j$	Server σ_1	Server σ_2	Server σ_3
Server σ_1	0	2	3
Server σ_2	2	0	3
Server σ_3	3	3	0

***Table 1**:* Cost communication *between servers* (*distance* $\sigma_r \rightarrow \sigma_p$ *)*