Un Noyau de système réparti pour les applications gérées par un temps virtuel

Philippe Ingels, Carlos Maziero, Michel Raynal

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992

25ème anniversaire

N° 1605

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

UN NOYAU DE SYSTEME REPARTI POUR LES APPLICATIONS GEREES PAR UN TEMPS VIRTUEL

Philippe INGELS
Carlos MAZIERO
Michel RAYNAL

Février 1992

RR_1605

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Un Noyau de Système Réparti pour les Applications Gérées par un Temps Virtuel

Philippe INGELS, Carlos MAZIERO, Michel RAYNAL
IRISA – Campus de Beaulieu
35042 RENNES cedex – FRANCE
<*Name*>@irisa.fr

Programme 1, Projet ADP (Algorithmes Distribués et Protocoles)
Publication Interne n°644, Janvier 1992, 20 pages

Résumé

L'arrivée des machines parallèles à mémoire répartie rend envisageables la conception et l'implémentation d'exécutifs spécialisés pour supporter des classes d'applications réparties. Cet article décrit une telle réalisation, en présentant un noyau réparti (appelé FLORIA), dont le but est de fournir un environnement de temps virtuel à des applications réparties (la simulation parallèle à événements discrets est la plus connue des applications utilisant une telle notion de temps virtuel - le *temps simulé* - et par conséquent constitue le paradigme de cette classe). Cet article décrit d'abord les caractéristiques fondamentales du concept de temps virtuel et ensuite les aspects liés à l'implémentation de ce concept dans un noyau réparti dédié aux applications gérées par un temps virtuel. Des mesures relatives au prototype réalisé sont également présentées.

Mots-clés: noyau réparti, temps virtuel, horloge virtuelle, synchronisation répartie, simulation répartie, contrôle pessimiste de la concurrence, implémentation.

A Distributed Kernel for Virtual Time Driven Applications

The advent of distributed memory parallel machines turns feasible the design and implementation of specialized environments for distributed computations. This paper addresses such a realization by presenting a distributed kernel, called FLORIA, whose aim is to provide a virtual time environment for application programs (distributed discrete event simulation is the most known of the applications based on such a virtual time - the so called *simulation time* - and consequently constitutes the paradigm of this class). This paper presents first basic features of the virtual time concept and then describes the time-related aspects of the implementation of a distributed kernel dedicated to virtual time driven applications. Measures about the implementation are also given.

Un Noyau de Système Réparti pour les Applications Gérées par un Temps Virtuel*

Philippe INGELS, Carlos MAZIERO†, Michel RAYNAL
IRISA
Campus de Beaulieu
35042 RENNES cedex - FRANCE
<*Name*>@irisa.fr

Resumé

L'arrivée des machines parallèles à mémoire répartie rend envisageables la conception et l'implémentation d'exécutifs spécialisés pour supporter des classes d'applications réparties. Cet article décrit une telle réalisation, en présentant un noyau réparti (appelé FLORIA), dont le but est de fournir un environnement de temps virtuel à des applications réparties (la simulation parallèle à événements discrets est la plus connue des applications utilisant une telle notion de temps virtuel - le *temps simulé* - et par conséquent constitue le paradigme de cette classe). Cet article décrit d'abord les caractéristiques fondamentales du concept de temps virtuel et ensuite les aspects liés à l'implémentation de ce concept dans un noyau réparti dédié aux applications gérées par un temps virtuel. Des mesures relatives au prototype réalisé sont également présentées.

Mots-clés: noyau réparti, temps virtuel, horloge virtuelle, synchronisation répartie, simulation répartie, contrôle pessimiste de la concurrence, implémentation.

1 Introduction

Les applications informatiques peuvent être grossièrement divisées en deux classes: celles dans lesquelles le temps intervient et celles dans lesquelles il n'intervient pas. Les applications appelées "temps-réel" constituent l'exemple type de

*Ce travail a été en partie financé par le Greco C^3 du CNRS, consacré à l'étude du parallèlisme et de la distribution.

†Cet auteur est financé par une bourse du gouvernement brésilien.

la première classe, les applications numériques constituent un exemple de la seconde classe. Les spécificités de chacune de ces deux classes doivent être prises en compte par les systèmes d'exploitation qui les supportent: en effet, dans la première classe le temps physique est un objet de programmation qui pilote l'application et que celle-ci peut utiliser [6, 8], alors que dans la seconde le temps n'est qu'une des ressources qui servent de support à l'implémentation [26].

Dans les applications temps-réel le temps physique défini par l'environnement constitue le point de référence commun à l'ensemble des processus, qui leur permet de se synchroniser entre eux, et avec l'environnement, de manière à réaliser le contrôle voulu. Cette utilisation du temps, étendue au temps logique, a donné naissance à la notion de temps virtuel [18, 24]. Dans un tel contexte il existe une horloge virtuelle globale à l'ensemble des processus, dont l'évolution est cadencée par des règles propres à l'application et non par un phénomène physique extérieur. Cette horloge permet d'organiser le calcul, d'en contrôler l'évolution, de dater des événements, de mesurer des durées dans le temps virtuel, etc [27].

On s'intéresse dans cet article à la définition et à la mise en œuvre d'un noyau de système réparti, implémenté sur une machine parallèle à mémoire répartie, qui construit un tel temps virtuel. L'article est divisé en 5 sections. La section §2 caractérise les propriétés offertes par le temps virtuel; la section §3 présente le noyau de système réparti réalisé (appelé FLORIA) qui offre ce temps virtuel; la section §4 en présente des utilisations possibles et donne des résultats de mesures réalisées pour évaluer le prototype implémenté, afin d'en connaître les performances.

Dans les applications concernées par la définition et l'utilisation d'un temps virtuel on trouve notamment la simulation à événements discrets [11, 15, 17, 18, 19, 22]. La machine support étant asynchrone, on peut également voir un tel noyau comme construisant une machine virtuelle synchrone, sur laquelle on peut exécuter des algorithmes répartis synchrones (en particulier des algorithmes sur les graphes [3, 16, 30]); un tel noyau est alors appelé synchroniseur [1, 2].

2 Le Temps Virtuel Offert par le Noyau

2.1 Propriétés du Temps Virtuel

Les applications concernées par ce noyau se présentent comme un réseau de processus connectés par des canaux de communications FIFO (i.e. sur lesquels l'ordre de réception des messages est le même que l'ordre de leurs émissions). Chaque processus est ainsi doté d'un certain nombre de canaux d'entrée sur lesquels il peut attendre des messages et de canaux de sortie sur lesquels il peut en envoyer [4].

Le temps virtuel offert est caractérisé par les deux propriétés temporelles fondamentales suivantes, sur lesquelles peuvent s'appuyer les applications :

P1: Tous les processus ont la même perception du temps virtuel (synchronisme

logique des processus); en d'autres termes, le temps virtuel est unique et progresse donc logiquement à la même vitesse pour tous.

P2: Dans le temps virtuel tout message émis à une date t est reçu par son destinataire à la date t. [1]

Les interactions entre les processus ne se font que par messages. Le temps virtuel constitue donc le seul mécanisme permettant aux processus de se synchroniser.

2.2 l'Interface du Noyau

Le noyau peut être vu par un processus P_i comme une machine offrant le temps virtuel; il fournit notamment une horloge globale hv qui donne la date virtuelle courante. Il met pour cela à la disposition de chaque P_i une représentation locale de l'horloge hv, ainsi que tous les messages qui lui ont été envoyés jusqu'à l'instant présent (qui par définition est la valeur de hv).

Tous les messages envoyés à un processus P_i à une date inférieure à hv sont ainsi connus de P_i, il est libre de les utiliser ou non (la consommation effective de ces messages dépend de sa logique propre) : ces messages sont placés dans une file appelée $Fconnus_i$, ordonnée par dates d'émission croissantes. Le noyau offre des primitives permettant de parcourir cette file (**Vide**, **Premier**, **Dernier**, **Prochain** (msg), **Antérieur** (msg)) et d'y prélever des messages (**Prendre** (msg)); ces opérations ont une durée nulle dans le temps virtuel. A l'instant virtuel t la file $Fconnus_i$ contient donc tous les messages émis vers P_i avant ou à la date t et que celui-ci n'a pas encore prélevé avec la primitive **Prendre** (msg).

Outre la primitive d'envoi de messages, le noyau offre les primitives suivantes, qui ont une durée non nulle dans le temps virtuel :

- **Attendre** $(hv = t)$: le processus P_i progresse, sans pouvoir être interrompu, jusqu'à ce que hv ait atteint la date t.
- **Attendre msg** : la progression du processus P_i est bloquée jusqu'à l'arrivée d'un message (lors de cette arrivée l'horloge virtuelle a pour valeur la date d'émission de ce message - propriété **P2** - et le message est placé dans $Fconnus_i$; il est alors accessible à l'aide de la primitive **Prendre** (msg)).

[1] On pourrait considérer à la place de **P2** la propriété **P2'** suivante :

P2': Lors qu'un message m est émis à la date virtuelle t, son émetteur précise la durée virtuelle δ_m associée à sa transmission; le message m est reçu à la date virtuelle $t + \delta_m$ par son destinataire.

Les propriétés **P2** et **P2'** sont équivalentes en ce qui concerne la puissance d'expression. En prennant $\delta_m = 0$ $\forall m$, la propriété **P2'** se réduit à **P2**. Si l'on considère **P2**, on peut toujours rajouter entre un processus émetteur et le destinataire associé un processus intermediaire qui sert de tampon et y conserve le message m pendant la durée δ_m; on obtient alors la propriété **P2'** entre l'émetteur et le destinataire.

- **Attendre** ($hv = t$) **ou msg** : le processus reprend son exécution à la date t si aucun message n'arrive avant cette date, à la date d'arrivée du message sinon. Il s'agit d'une attente de message avec échéance dans le temps virtuel.

Toutes le autres opérations exécutées par un processus ont une durée nulle dans le temps virtuel [5, 17].

Construire un noyau consiste à offrir des outils de base suffisamment généraux pour pouvoir traiter des problèmes divers. Les primitives du noyau ne sont pas destinées à être utilisées directement au niveau utilisateur. Par exemple, si on souhaite réaliser un simulateur de réseaux de files d'attente, l'utilisateur aura à sa disposition un langage adapté à la description du réseau, qui sera ensuite traduit en terme d'appels au noyau [10, 13]. De manière générale, les opérations offertes à l'utilisateur sont implémentés à l'aide des primitives de base énoncées ci-dessus. Ainsi, par exemple, l'attente par un processus P_i de l'application d'un message selon une discipline FIFO de consommation serait implémentée en utilisant les primitives **Vide**, **Attendre msg**, **Prendre** (msg) et **Premier** (on examinera à la section 4.1 des exemples d'utilisation des primitives du noyau FLORIA).

3 La Réalisation du Noyau

3.1 Objectif

Deux problèmes importants se posent dans la réalisation d'un tel noyau sur une machine parallèle à mémoire répartie: le placement des processus sur les processeurs [23] et la mise en œuvre du temps virtuel. Dans la réalisation actuelle nous avons choisi un placement calculé a priori, et nos efforts ont été axés sur le deuxième point: effectuer une mise en œuvre cohérente du temps virtuel du point de vue du programme utilisateur, tout en ayant des processus qui évoluent physiquement de façon la plus indépendante possible les uns des autres, ceci afin d'obtenir la meilleure efficacité possible (mesurée en temps de calcul effectif).

L'architecture sur laquelle est construit le noyau FLORIA est un hypercube d'INTEL avec 64 processeurs. Le langage utilisé pour son implémentation est C. FLORIA se décompose en deux niveaux assurant des fonctions distinctes:

- la mise en œuvre de la communication dans le temps virtuel (qui permet d'assurer la propriété **P2**);

- la gestion de l'évolution des processus dans le temps virtuel; celle-ci est fonction à la fois de leur logique propre et de l'état de leurs canaux d'entrée.

3.2 La Mise en Œuvre des Communications dans le Temps Virtuel

Il s'agit ici de réaliser la propriété **P2**, c'est à dire, de faire en sorte que tout message émis à la date virtuelle t soit délivré à son récepteur à cette même date virtuelle. Pour cela chaque message est estampillé par sa date virtuelle d'émission et les messages reçus par un processus lui sont délivrés dans l'ordre de leurs dates d'émission: ceci permet de garantir que la propriété de sûreté ne peut être violée, en d'autres termes, l'évolution du temps virtuel sur chaque processus ne peut être incohérente (un processus ne peut alors "remonter le temps virtuel" en recevant un message émis à t puis un autre à t' avec $t' < t$) [15, 18, 22, 29].

Comme les communications réelles sont asynchrones, la livraison des messages dans un tel ordre nécessite un contrôle particulier [15, 17, 18, 22]. Considérons en effet la figure 1, dans laquelle le processus P_i, doté de 2 canaux d'entrée, reçoit 3 messages pourvus d'estampilles telles que $t_1 < t_2 < t_3$.

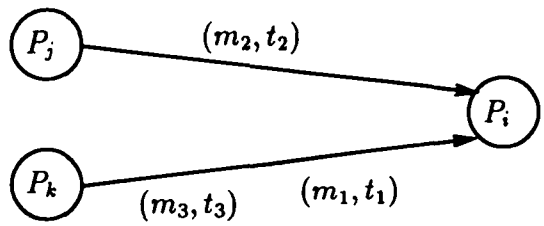

Figure 1 : Livraison de messages dans le temps virtuel

On peut délivrer les messages (m_1, t_1) et (m_2, t_2), mais on ne peut pas délivrer (m_3, t_3); en effet le processus P_j peut envoyer un message (m'_2, t'_2) à P_i tel que $t'_2 < t_3$, auquel cas ce dernier doit être délivré avant (m_3, t_3). Pour résoudre ce problème deux classes de techniques ont été proposées: les techniques optimistes [19] et les techniques pessimistes [11].

Le noyau FLORIA utilise la technique pessimiste avec prévention de l'interblocage [22]; celle-ci consiste à rajouter des messages de contrôle: quand un processus P_j sait qu'il n'enverra pas de messages sur un canal d'ici la date t ($t \geq hv$), il émet sur ce canal le message $(NULL, t)$. Cette prévision sur le futur du canal de sortie est un facteur qui a une influence importante sur l'efficacité de la méthode [14, 15, 17, 29] (on verra au §4.2.2 des résultats de mesures concernant l'influence de ce paramètre). Le noyau offre une primitive permettant à un processus de fixer, à tout moment, une prévision sur le futur d'un canal de sortie.

Considérons les variables suivantes, le processus P_i ayant k_i canaux d'entrée: $ce_i[1]$, $ce_i[2]$, ..., $ce_i[k_i]$:

- *temps_canal*$(ce_i[x])$ est par définition l'estampille du dernier message ($NULL$

ou de l'application) reçu par P_i dans son canal d'entrée $ce_i[x]$.

- $he_i = min_{1 \leq x \leq k_i}(temps\_canal(ce_i[x]))$: on peut considérer cette valeur comme une horloge associée à l'ensemble des canaux d'entrée de P_i; cette horloge est gérée par la couche de communication dans le temps virtuel.

La couche de communications dans le temps virtuel peut alors délivrer à P_i tous les messages (m, t) tels que $t \leq he_i$; cette valeur représente la date jusqu'à laquelle le contenu des canaux d'entrée du processus P_i est entièrement connu et par conséquent peut lui être délivré (mis dans $Fconnus_i$) au fur et à mesure de l'avancement du temps virtuel. Il est montré dans [11, 22] que, si chacun des processus progresse dans le temps virtuel, ce mécanisme garantit la progression de l'ensemble, i.e. la propriété de vivacité (ce schéma d'implémentation est trés proche de celui que mettent en œuvre les synchroniseurs de réseaux [3, 16]).

3.3 Le Contrôle de l'Exécution des Processus

Chaque processus P_i est doté d'une représentation locale hp_i de l'horloge globale hv; en d'autres termes, si $hp_i = t$ le processus P_i a progressé jusqu'à la date virtuelle t. Il est possible en fait d'avoir, à un instant donné de l'exécution, $hp_i \neq hp_j$ sans mettre en défaut la propriété **P1**; il suffit pour garantir cette propriété que tout processus P_i parvenu à la date t ($hp_i = t$) ne puisse pas à cet instant en percevoir un autre à une date différente de t.

Si P_i a une perception t de hv ($hp_i = t$) nous avons $he_i \geq t$, grâce à la couche de communication dans le temps virtuel (§3.2); celle-ci a alors placé dans $Fconnus_i$ tous les messages (m, t') tels que $t' \leq hp_i$. Rappelons de plus que P_i est libre de consommer ou non les messages de $Fconnus_i$ (cela ne dépend que de son texte de programme).

En fait les deux horloges hp_i (du processus) et he_i (de l'entrée) d'un processus P_i peuvent être désynchronisées; ceci favorise l'efficacité de l'exécution, le noyau ne faisant que les resynchronisations strictement nécessaires pour garantir les propriétés **P1** et **P2** dans le temps virtuel. Cela se traduit par les schémas d'exécution suivants pour les trois primitives qui font intervenir le temps virtuel.

3.3.1 Mise en Œuvre de l'Attente sur Horloge

Un processus P_i qui effectue **Attendre** ($hv = t$) poursuit son exécution avec $hp_i = t$. Rappelons que par sa définition la propriété suivante est associée à $Fconnus_i$: cette file contient, à l'instant t, tous les messages émis vers P_i avant ou à la date t et non encore consommés. Avec cette mise en œuvre de **Attendre** ($hv = t$) (qui consiste simplement à effectuer $hp_i := t$), la propriété précédente peut être mise en défaut. Mais comme le montre l'analyse des deux situations suivantes, elle peut toujours être rétablie de façon à ce que cette incohérence temporaire ne soit jamais perceptible par le processus P_i.

- $hp_i < he_i$: tous les messages (m, t') du passé du processus $(t' \leq hp_i)$ non encore retirés par **Prendre** (msg) sont réellement disponibles dans $Fconnus_i$ (figure 2), il peut donc les accéder et les consommer s'il le désire.

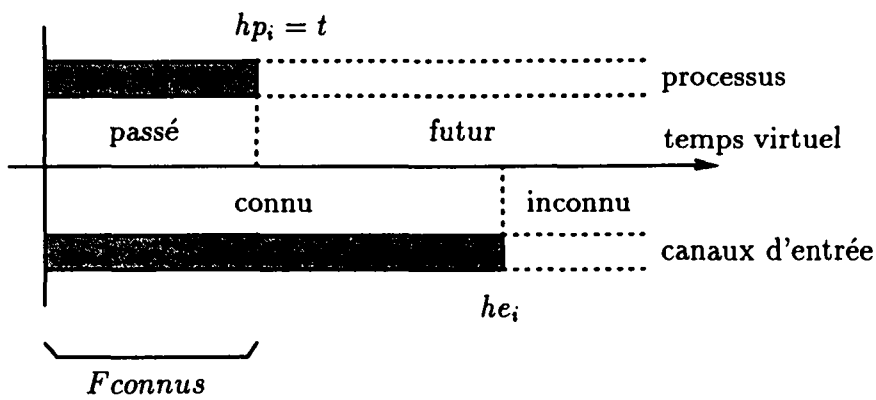

Figure 2 : situation $hp_i < he_i$

- $hp_i \geq he_i$: une partie des messages relatifs au passé de P_i (les messages (m, t') dont il est destinataire tels que $he_i < t' \leq hp_i$) n'est pas réellement disponible, la file $Fconnus_i$ n'est pas entièrement connue (la partie inconnue est représenté par la zone blanche de la figure 3). Seuls les messages (m, t') tels que $t' \leq he_i$ sont réellement disponibles dans $Fconnus_i$.

  On pourrait, dans cette deuxième situation, bloquer l'exécution de P_i jusqu'à ce que, grâce à l'arrivée de messages et à la mise à jour de he_i par la couche de communication, on ait $he_i \geq hp_i$, mais cela n'est pas toujours nécessaire (par exemple lorsque P_i n'utilise pas des messages émis entre he_i et hp_i, c'est le cas dans un programme de simulation d'un serveur FIFO qui a encore à traiter des messages émis avant he_i). Nous avons choisi de laisser le processus continuer son exécution dans cette situation. Mais, c'est lors d'éventuels appels aux primitives d'accès à $Fconnus_i$ (**Premier**, **Dernier**, etc) qu'il sera bloqué, s'il tente d'accéder à la partie non connue de $Fconnus_i$ (partie blanche dans la figure 3). Il reprendra son exécution lorsque la couche de communication dans le temps virtuel lui aura délivré le message attendu : la propriété associée à la définition de $Fconnus_i$ sera alors vérifiée.

3.3.2 Mise en Œuvre de l'Attente sur Message

L'exécution de la primitive **Attendre msg** implique toujours l'avancement de l'horloge virtuelle du processus. Elle peut imposer ou non un blocage du processus, suivant qu'il existe ou non un message disponible sur le canal d'entrée :

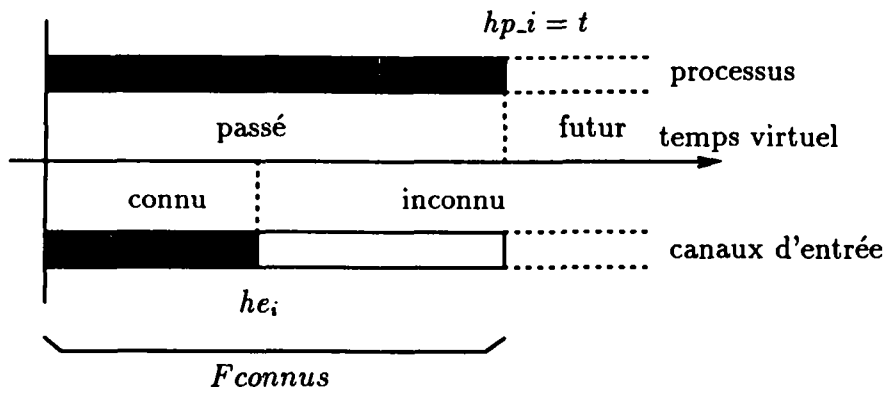

Figure 3 : situation $hp_i \geq he_i$

- $\exists (m,t) \mid hp_i < t \leq he_i$: dans ce cas le plus ancien de ces messages (m,t) est intégré à la file $Fconnus_i$ et hp_i est avancée à t.
- $\nexists (m,t) \mid hp_i < t \leq he_i$: le processus est bloqué; il ne reprendra son exécution que quand un nouveau message sera délivré par la couche de communication dans le temps virtuel; on appliquera alors le traitement décrit ci-dessus.

3.3.3 Mise en Œuvre de l'Attente sur Message avec Échéance

L'implémentation de la primitive **Attendre** $(hv = t)$ **ou msg** est obtenue par une combinaison des deux schémas d'exécution précédents (§3.3.1 et §3.3.2).

4 Utilisation du Noyau

4.1 Comment Utiliser ce noyau?

Considérons, pour illustrer l'utilisation du noyau FLORIA, la mise en œuvre des deux instructions suivantes, offertes par un langage à l'utilisateur : *Consommer_Temps* (δ), pour attendre l'écoulement de δ unités de temps virtuel et *Prendre_LIFO* (m), qui délivre au processus P_i qui l'invoque le dernier message arrivé. On implémente la première par la séquence suivante :

$t_{fin} := hv + \delta$;
Attendre $(hv = t_{fin})$;

La seconde est réalisée par la séquence d'appels suivante :

```
/* si Fconnus; est vide, attendre un nouveau msg */
si Vide alors Attendre msg fsi;
/* prendre le dernier msg de Fconnus; */
m := Prendre (Dernier);
```

De telles instructions peuvent par exemple être utilisées pour définir un simulateur de serveur de files d'attente selon la discipline LIFO, sans préemption; celui-ci s'exprime alors par:

```
boucle
    Prendre_LIFO (client);
    Consommer_Temps (temps de service pour ce client);
    Envoyer (client) sur le canal Sortie;
fin boucle
```

Les primitives fournies par le noyau FLORIA permettent de définir des services d'attente de message satisfaisant un prédicat ou appartenant à un type donné, avec ou sans échéance, anologues à ceux proposées dans [5]. La mise en œuvre de ce type d'extension est simple avec les primitives d'attente de message décrites dans les paragraphes précédents; il faut alors boucler sur une attente de message et tester les messages reçus jusqu'à ce que: soit un message reçu satisfait le prédicat donné, soit la date d'échéance définie est arrivée.

4.2 Evaluation de Performances de FLORIA

Comme nous l'avons indiqué, la simulation à événements discrets constitue la classe d'applications typique pour l'utilisation du temps virtuel. Nous avons donc, à l'aide des expériences suivantes, testé le fonctionnement du noyau et mesuré ses performances pour ce type d'applications.

4.2.1 Un Réseau de Serveurs

Le modèle de simulation, inspiré de [14], est constitué de 256 processus interconnectés par une structure régulière torique 16×16 (fig. 4). Chacun de ces processus, doté de 2 canaux d'entrée et de 2 canaux de sortie, simule un serveur FIFO selon le comportement suivant (les clients à servir sont représentés par les messages):

```
boucle
    envoyer msg sur une des 2 sorties en les alternant;
    recevoir msg;
    attendre temps de service Ts;
fin boucle
```

Dans cette application, la population de clients (messages) présents à un instant donné (en transit ou en traitement) est constante et égale à 256.

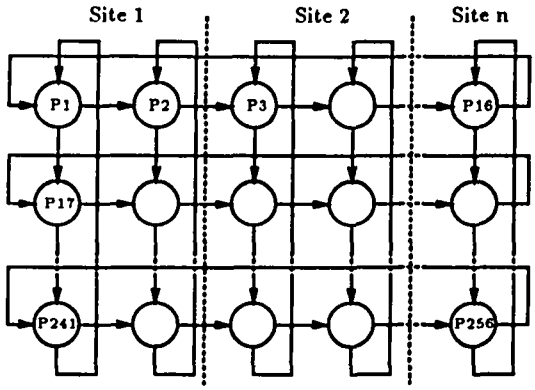

Figure 4 : structure de l'application

Le premier paramètre intéressant est le nombre de processeurs physiques sur lesquels les processus sont placés. Les valeurs étudiées sont 1 (tous les processus s'exécutent alors sur le même processeur), 2, 4, 8 et 16 (chaque processeur exécute alors 16 processus). Pour le placement sur une architecture avec n processeurs le tore des processus est découpé en n tranches verticales, chacune affectée à un processeur (fig. 4).

Les seconds paramètres étudiés concernent le temps virtuel: la durée des temps Ts de service de chaque processus et la prévision $Prev$ sur son comportement futur (*lookahead*) que peut faire un processus. Les cas suivants ont été étudiés :

Courbe	Temps de service Ts	Prévision $Prev$
$\diamond$	10.0	Ts
$+$	$1.0 + uniforme\ (1.0\ ..\ 19.0)$	1.0
$\square$	$1.0 + uniforme\ (1.0\ ..\ 19.0)$	Ts
$\times$	$1.0 + expn\ (moyenne = 9.0)$	1.0
$\triangle$	$1.0 + expn\ (moyenne = 9.0)$	Ts

Dans les cas où $Prev = Ts$, le processus fournit comme prévision sur le futur la valeur de son prochain temps de service. Rappelons que si un processus parvenu à l'instant virtuel t est doté d'une prévision sur le futur $Prev$, cela signifie qu'il connaît alors les messages qu'il peut envoyer sur ses canaux de sortie jusqu'à l'instant $t+Prev$ (nous avons vu dans §3.2 que cette connaissance peut faciliter la mise en œuvre des communications dans le temps virtuel). Les courbes notées $\diamond$, $\square$ et $\triangle$ sont donc relatives à la meilleure prévision sur le futur possible.

L'ensemble des résultats obtenus est représenté à la figure 5, qui donne les temps effectifs de calcul en secondes en fonction du nombre de processeurs. On y constate une diminution du temps total d'exécution lorsque le nombre de processeurs augmente.

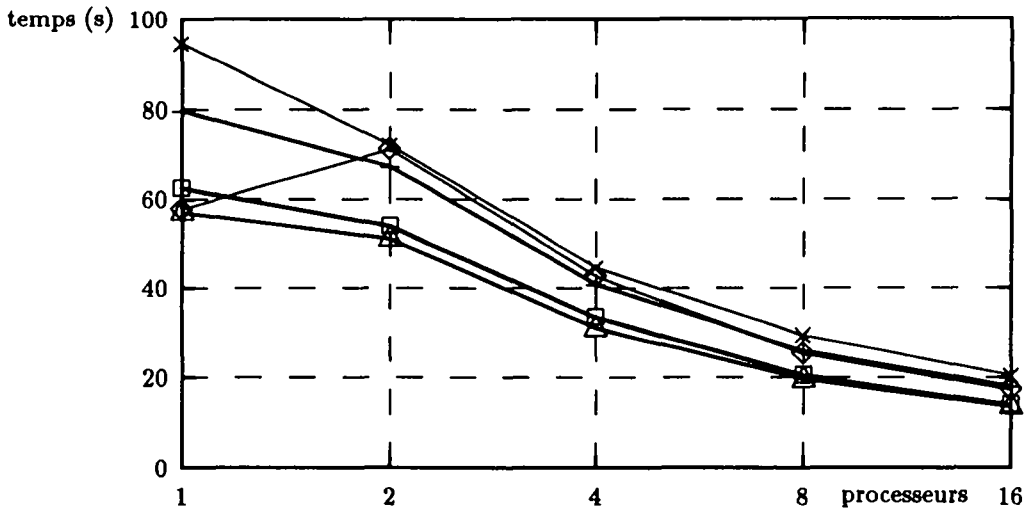

Figure 5 : temps effectifs de calcul vs. nombre de processeurs

Un autre paramètre étudié est le pourcentage de messages *NULL* rajoutés par le système par rapport au nombre total de messages utilisés; la figure 6 montre que ce pourcentage augmente peu par rapport au nombre de processeurs, en d'autres termes, rajouter des processeurs rajoute peu de messages NULL. La figure 7 montre le pourcentage total du temps d'exécution passé dans le noyau; on constate que le temps passé dans le noyau augmente avec le nombre de processeurs: cela s'explique par le fait qu'alors chaque processeur a moins de processus à exécuter et passe donc plus de temps à attendre des messages. Ces 2 figures illustrent en fait le coût du contrôle de l'application : on constate que celui-ci augmente légèrement avec le nombre de processeurs, mais la figure 5 indique que ceci ne se fait pas au détriment de l'efficacité globale.

4.2.2 Influence de la Prévision sur le Futur

Pour évaluer l'influence de la prévision sur le futur (cf. §3.2) sur les performances du noyau FLORIA, nous avons utilisé une application avec la même structure (tore 16 × 16), le même placement (en tranches verticales) et le même comportement (FIFO) que celle utilisée dans l'expérience précédente. Le temps de service

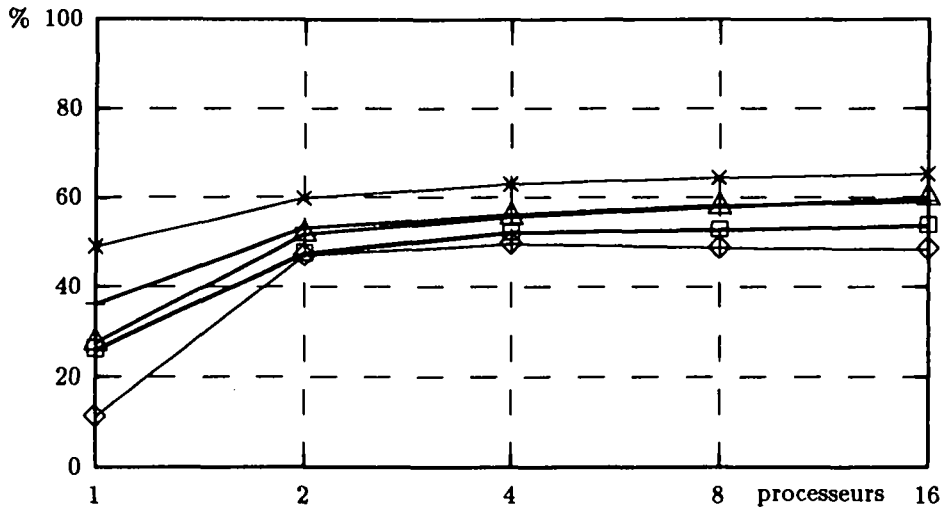

Figure 6 : pourcentage de messages *NULL* vs. nombre de processeurs

est défini par $Ts = 1.0 + uniforme\ (1.0\ ..\ \delta)$, et la prévision sur le futur est fixée à $Prev = 1.0$. La valeur δ constitue donc le paramètre des mesures.

Nous avons défini, pour cette expérience, la qualité Q d'une prévision comme le rapport entre sa valeur $Prev$ et la valeur moyenne du temps de service Ts: $Q = Prev/((1.0 + \delta)/2)$, c'est à dire, $Q = 2.0/(1.0 + \delta)$. Donc, si $Q = 1.0$, la prévision sur le futur $Prev$ est la meilleure possible; par contre, lorsque $Q \rightarrow 0.0$, la prévision devient de plus en plus pauvre.

La figure 8 présente les temps effectifs d'exécution en secondes obtenus sur une architecture cubique avec 8 processeurs, en attribuant à δ des valeurs entre 1.0 ($Q = 1.00$) et 100.0 ($Q = 0.02$). On constate que, s'il est possible d'avoir une prévision sur le futur de bonne qualité, le temps de calcul global peut être réduit de façon substantielle (ce qui confirme les résultats de [14], obtenus sur une machine à mémoire partagée).

Comme dans ces expériences les temps de service varient d'une expérience à l'autre, il n'a pas été possible de les conduire toutes avec la même durée dans le temps simulé. Afin de pouvoir comparer ces différentes exécutions, elles ont été faites avec le même nombre (5000) de messages traités par chaque processus de l'application: la charge (en nombre de messages traités) a donc été indépendante des valeurs de temps de service et la même pour toutes les expériences.

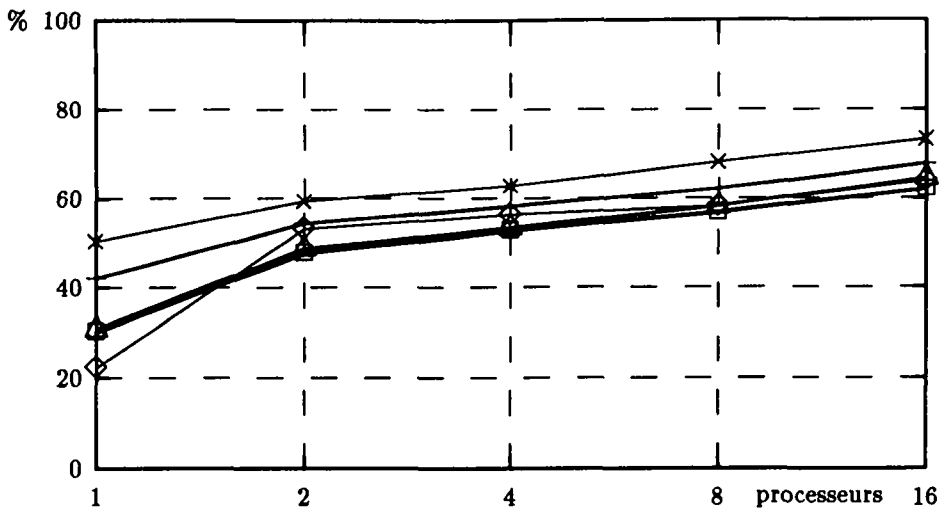

Figure 7 : pourcentage de temps dans le noyau vs. nombre de processeurs

5 Conclusion

Une version du noyau FLORIA est actuellement réalisée. Dans le cadre d'une simulation de réseaux de files d'attente, des processus réalisant la mise en œuvre de divers types de files ont été écrits (FIFO, LIFO, Quantum, avec ou sans préemption, etc). L'expérience montre que les primitives offertes par le noyau permettent d'exprimer facilement l'évolution de processus dans le temps virtuel. La version actuelle du noyau est réalisée directement sur le système d'exploitation de l'iPSC d'INTEL (un sous-ensemble d'UNIX); nous envisageons sa réécriture en utilisant un micro-noyau (Chorus ou Mach) plus adapté à la gestion de la distribution.

Sur le plan plus général, la conception d'un noyau réparti pour les applications pilotées par un temps virtuel est très intéressante par les contraintes de synchronisation qu'elle définit sur les communications: tout message émis à la date virtuelle t est reçu à cette même date t (nous appelons cette propriété VTO, *virtual time ordering*). On trouve une approche analogue dans les mécanismes de contrôle des accès aux données utilisant un ordre créé par l'estampillage [7, 21]. De manière générale, considérons la relation de causalité potentielle entre événements définie par Lamport [20] et notée "$\rightarrow$" (cette relation, également "happened before", permet de modéliser un calcul réparti comme un ordre partiel sur un ensemble d'événements qui ne laisse apparaître que les causalités qui les lient). Un certain nombre de noyaux de système offrent la communication causale (ordre causal, *CO*) comme contrôle de base [9, 25, 28]; ces systèmes sont caractérisés par la propriété suivante: si *envoyer* (m_1)

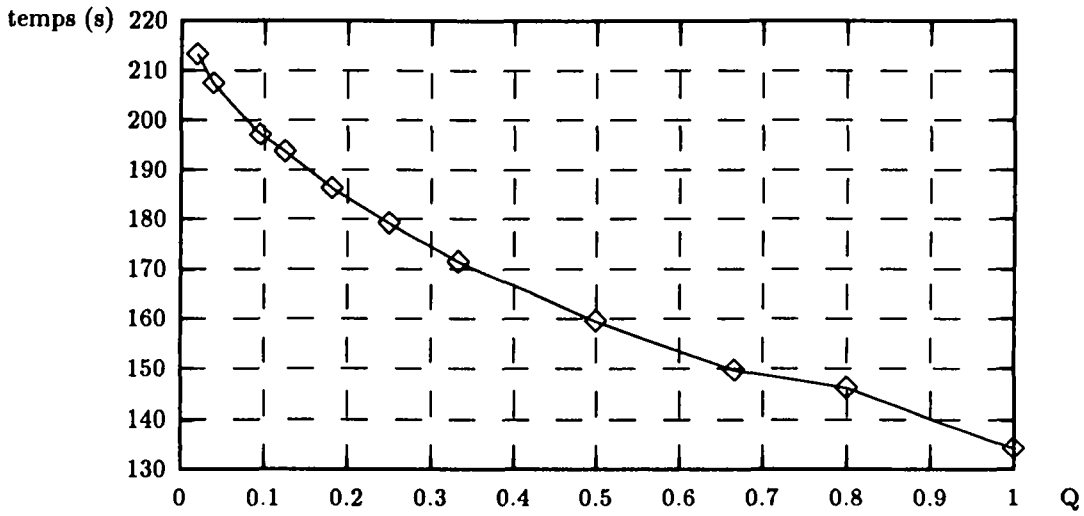

Figure 8 : influence de la prévision sur le futur

et *envoyer* (m_2) sont deux émissions de message vers la même destination, et si *envoyer* (m_1) $\rightarrow$ *envoyer* (m_2), alors m_1 doit être délivré avant m_2 (dans le cas particulier où m_1 et m_2 ont le même émetteur, la communication causale se confond avec la propriété FIFO). Il est facile de voir que $VTO \Rightarrow CO$.

Ceci suggère une étude plus approfondie sur les propriétés des communications qu'offrent les noyaux des systèmes répartis [12]. Depuis les schémas totalement asynchrones jusqu'au schéma très contraint qu'est VTO, il semble important de dégager des types de communication de base et de caractériser proprement les applications qui les nécessitent. Le noyau FLORIA est un pas concret dans cette direction.

Références

[1] M. Adam, Ph. Ingels, and M. Raynal. The meaning of synchronous distributed algorithms run on asynchronous distributed systems. In *The Third International Symposium on Computer and Information Sciences, Izmir*, pages 307–316, November 1988.

[2] B. Awerbuch. Complexity of network synchronization. *Journal of ACM*, 32(4):801–823, October 1985.

[3] B. Awerbuch. Reducing complexities of the distributed max-flow and breadth-first algorithm by means of network synchronization. *Networks*, 15:425–437, 1985.

[4] R.L. Bagrodia and K.M. Chandy. A micro-kernel for distributed applications. In *IEEE Proceedings of the 5th International on Distributed Computings Systems, New York*, pages 140–149, May 1985.

[5] R.L. Bagrodia, K. M. Chandy, and J Misra. A message-based approach to discrete event simulation. *IEEE Trans. on Soft. Eng.*, 13(6):654–665, June 1987.

[6] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79,9, September 1991.

[7] P.A. Bernstein and N. Goodman. Concurrency control in distributed data base system. *Computing Surveys*, 13,2:185–221, June 1981.

[8] G. Berry. Real time programming: special purpose or general purpose languages. In *Proc. IFIP congress*, 1989. invited talk.

[9] K.P. Birman and T.A. Joseph. Exploiting virtual synchrony in distributed systems. *Proc.* 11^{th} *ACM Symposium on Op. Systems Principles*, 14(4):123–137, 1987.

[10] G.M. Birtwistle, O.J. Dahl, B. Myhrhaug, and K. Nygaard. *Simula Begin.* Chartwell-Bralt Ltd., 1973. 391 p.

[11] K.M. Chandy and J. Misra. Distributed simulation: a case study in design and verification of distributed programs. *IEEE Trans. on Soft. Eng., Vol. 5, No 5*, 440–452, September 1979.

[12] B. Charron-Bost, F. Mattern, and G. Tel. *Synchronous and Asynchronous Communication in Distributed Computations.* Research report 91-55, LITP - Paris 7, September 1991. 33 p.

[13] J.B. Evans. *Structure of Discrete Event Simulation: an introduction to engagement strategy.* Ellis Horwood Ltd., 1988. 279 p.

[14] R. M. Fujimoto. Lookahead in parallel discrete event simulation. *Proceedings of the 1988 International Conference on Parallel Processing, Pennsylvania*, 34–41, August 1988.

[15] R. M. Fujimoto. Parallel discrete event simulation. *Communications of ACM*, 33:31–53, October 1990.

[16] J.M. Hélary and M. Raynal. *Synchronization and Control of Distributed Systems and Programs.* John Wiley & Sons, 1990. 124 p.

[17] Ph. Ingels and M. Raynal. Simulation répartie : schémas d'exécution pour un modèle à processus. *Technique et Science Informatiques*, 9(5):383–397, 1990.

[18] D. Jefferson. Virtual time. *ACM Toplas, Vol. 7, No 3*, 404–425, July 1985.

[19] D. Jefferson et al. Distributed simulation and the time warp operating system. In 11^{th} *ACM Symposium on Operating systems principles, Austin - Texas*, pages 77–93, Operating system Review, ACM Press, November 1987.

[20] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[21] M. Maekawa, A.E. Oldehoeft, and R.R. Oldehoeft. *Operating Systems : Advanced Concepts.* The Benjamin/Cummings Pub. Co, 1987. 414 p.

[22] J. Misra. Distributed discrete-event simulation. *Computing Surveys, Vol. 18, No 1*, 39–65, March 1986.

[23] T. Muntean and E.G. Talbi. Méthodes de placement statique de processus sur architectures parallèles. *Technique et Science Informatiques*, 10(5):355–374, 1991.

[24] G. Neiger and S. Toueg. Substituting for real time and common knowledge in distributed sytems. In *6th Annual ACM Symposium on Principles of Distributed Computing*, pages 281–293, August 1987.

[25] L. W. Peterson, N. C. Bucholz, and D. Schilchting. Preserving and using context information in interprocess communication. *ACM TOCS*, 7(3):217–246, 1989.

[26] M. Raynal. A simple taxonomy of distributed mutual exclusion algorithms. *ACM Operating Sytems Review*, 25:47–51, april 1991.

[27] M. Raynal. *Synchronisation et Etat Global dans les Systèmes Répartis.* Eyrolles, Collection EDF, Janvier 1992. 228 p.

[28] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Inf. Proc. Letters*, 30:343–350, 1991.

[29] R. Righter and J.C. Walrand. Distributed simulation of discrete event systems. In *Proc. of the IEEE*, pages 99–113, January 1989.

[30] G. Tel. *Topics in Distributed Algorithms.* Cambridge Int. Series on Parallel Processing, 1991. 250 p.

LISTE DES PUBLICATIONS INTERNES IRISA 1992

PI 624 SIGNAL AS A MODEL FOR REAL-TIME AND HYBRID SYSTEMS
Albert BENVENISTE, Michel LE BORGNE, Paul LE GUERNIC
Janvier 1992, 22 pages.

PI 625 ON THE CENTRAL-LIMIT THEOREM FOR TRACKING ESTIMATORS WITH SMALL GAIN - INFINITE HORIZON CASE
Bernard DELYON, Anatoli JUDITSKY
Janvier 1992, 16 pages.

PI 626 A MONTE CARLO METHOD BASED ON ANTITHETIC VARIATES FOR NETWORK RELIABILITY COMPUTATIONS
Mohamed EL KHADIRI, Gerardo RUBINO
Janvier 1992, 28 pages.

PI 627 CONSTRAINED MULTISCALE MARKOV RANDOM FIELDS AND THE ANALYSIS OF VISUAL MOTION
Fabrice HEITZ, Patrick PEREZ, Patrick BOUTHEMY
Janvier 1992, 40 pages.

PI 628 ON ITERATIVE REFINEMENT FOR THE SPECTRAL DECOMPOSITION OF SYMMETRIC MATRICES
Alexander N. MALYSHEV
Janvier 1992, 26 pages.

PI 629 STRUCTURAL OPERATIONAL SPECIFICATIONS AND TRACE AUTOMATA
Eric BADOUEL, Philippe DARONDEAU
Janvier 1992, 36 pages.

PI 630 EREBUS, A DEBUGGER FOR ASYNCHRONOUS DISTRIBUTED COMPUTING SYSTEM
Michel HURFIN, Noël PLOUZEAU, Michel RAYNAL
Janvier 1992, 14 pages.

PI 631 PROTOCOLES SIMPLES POUR L'IMPLEMENTATION REPARTIE DES SEMAPHORES
Michel RAYNAL
Janvier 1992, 14 pages.

PI 632 L-STABLE PARALLEL ONE-BLOCK METHODS FOR ORDINARY DIFFERENTIAL EQUATIONS
Philippe CHARTIER, Bernard PHILIPPE
Janvier 1992, 28 pages.

PI 633 ON EFFICIENT CHARACTERIZING SOLUTIONS OF LINEAR DIOPHANTINE EQUATIONS AND ITS APPLICATION TO DATA DEPENDENCE ANALYSIS
Christine EISENBEIS, Olivier TEMAM, Harry WIJSHOFF
Janvier 1992, 22 pages.

PI 634 UN NOYAU DE SYSTEME REPARTI POUR LES APPLICATIONS GEREES PAR UN TEMPS VIRTUEL
Janvier 1992, 20 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399