Calculateurs Parallèles, Vol. 8(2), June 96, pp229-243

Simulations Distribuées de Réseaux ATM

C.D. Pham

Laboratoire MASI
Université Pierre et Marie Curie
4 Place Jussieu
75252 Paris Cedex 05
Tél. : +33 (1) 44-27-75-12
Fax : +33 (1) 44 27 62 86
E-mail : pham@masi.ibp.fr

RÉSUMÉ. *L'évaluation des futurs réseaux haut-débit est une étape indispensable à un bon dimensionnement des ressources. Cette évaluation est difficile, longue et gourmande en puissance de calcul. En particulier, la simulation de grands réseaux de commutateurs au niveau cellule nécessite l'exécution de millions d'événements. Dans cet article, nous utilisons des techniques de simulation distribuée dans le but d'une part, de réduire le temps de simulation et d'autre part, d'augmenter la taille du système simulé. Un noyau de simulation utilisant l'approche conservatrice est réalisé à l'aide de la librairie PVM. Les résultats sur une CM-5 montrent des gains en vitesse très satisfaisants.*

ABSTRACT. *Performance evaluation of future high-speed networks is essential to correctly size its components. This evaluation is difficult, long and very computational expensive. In particular, the simulation of large switch networks requires the processing of million of events. In this article, distributed simulation techniques are used in order to reduce the simulation time and to increase the size of the system. A conservative simulation kernel is implemented using PVM and the results on a CM-5 show that very good speedups can be achieved.*

MOTS CLÉS : *Simulation distribuée, Réseaux ATM.*
KEYWORDS : *Distributed simulation, ATM Networks.*

1 Introduction

Les communications informatiques ont augmenté de manière exponentielle ces dernières années. Cette augmentation se poursuit à un rythme effréné et les réseaux de télécommunication seront marqués dans les prochaines années par une demande de plus en plus diversifiée, par une compétition accrue et par des évolutions technologiques considérables. Parmi ces techniques on retiendra surtout la technique de transfert asynchrone (ATM) qui a été adoptée comme la technique de transfert des futurs réseaux à large bande. Une lourde tâche lui incombe puisqu'elle devra supporter la grande variété des services demandés par les applications multimédia et coopératives. Il est par conséquent nécessaire d'obtenir une bonne adéquation entre cette technique et le besoin des applications, et cela ne pourra se faire qu'au travers d'un dimensionnement approprié des ressources du réseau. Il s'ensuit une activité intense au niveau international pour évaluer soigneusement les composantes du réseau afin d'en maîtriser le fonctionnement. Pour cela, de nouveaux outils d'évaluation sont nécessaires car la complexité des systèmes étudiés augmente ainsi que leur dimension, rendant leur étude délicate voire impossible dans le contexte actuel.

Dans ce travail, nous nous intéressons à l'évaluation de performance des réseaux ATM par des techniques de simulation distribuée. L'implantation sur une architecture parallèle (CM-5) est réalisée à l'aide de la librairie PVM [Pvm93] (Parallel Virtual Machine). L'organisation de l'article est la suivante: les chapitres 2 et 3 présentent les motivations du choix des techniques de simulation distribuée et leurs concepts de base. Le modèle de réseau et l'environnement de l'étude (PVM, les architectures utilisées et le noyau de simulation) sont respectivement présentés dans les chapitres 4 et 5. Enfin, les résultats font l'objet du chapitre 6 et nous concluons en présentant les perspectives de recherches futures dans le chapitre 7.

2 Évaluation de performance: la simulation

Les futurs réseaux ATM seront de grande dimension, impliqueront de multiples éléments souvent complexes et par conséquent poseront un grand nombre de problèmes. Parmi ceux-ci, on peut citer les problèmes de routage et de contrôle de congestion qui sont tous deux très complexes à résoudre à grande échelle. Des critères de performance tels que la longueur moyenne des files d'attente, le taux de perte de cellules ou la probabilité d'appels rejetés sont également très utiles pour bien dimensionner les éléments de commutation. Il est alors nécessaire de disposer d'outils d'évaluation permettant d'étudier tel ou tel mécanisme. La plupart des systèmes informatiques peuvent être considérés comme un ensemble de ressources et de clients concourant pour ces ressources. Des phénomènes d'attente inhérents à ces systèmes font que les méthodes de résolution analytique utilisent naturellement le formalisme des files d'attente. Les réseaux de files d'attente à forme produit ou non donnent la possibilité de modéliser un système donné puis d'en extraire des critères de performance à un faible coût puisque la résolution se fait analytiquement.

En contre partie, le champ d'application de ces méthodes est relativement réduit car celles-ci imposent souvent des hypothèses de fonctionnement très restrictives qui nécessitent de fortes simplifications pouvant rendre l'étude non réaliste. La simulation, contrairement aux méthodes analytiques, ne présente pas de limites théoriques quand au niveau de détail du modèle [Law91]. Ces avantages font que la simulation est devenue un outil beaucoup utilisé dans l'évaluation de performance des systèmes informatiques. Cependant cette fantastique souplesse se paye cher, et le principal facteur limitatif est le temps de simulation qui peut très vite devenir excessif lorsque le modèle est un tant soit peu réaliste. Par exemple, pour évaluer des algorithmes de contrôle d'admission, simuler au niveau appel peut être suffisant mais, dans la plupart des cas, il est nécessaire de descendre jusqu'au niveau cellule pour réellement obtenir des critères intéressants. Malheureusement, simuler à ce niveau implique l'exécution de millions d'événements, ce qui allonge dangereusement le temps de simulation. De manière générale, on peut prévoir que la simulation des futurs réseaux large bande nécessitera une grande quantité de calcul.

La situation est alors la suivante: d'un côté, la simulation séquentielle est dépassée par l'augmentation de la puissance de calcul requise, alors que d'un autre côté, le calcul distribué et les architectures parallèles offrent une formidable puissance très souvent sous-exploitée. Un grand effort de recherche pour utiliser cette puissance nouvellement disponible pour des tâches de simulation s'est donc déclaré depuis ces dix dernières années, et des techniques de simulation distribuée ont été proposées[1]. Ces techniques, encore méconnues du grand public et d'un usage délicat, pourraient s'avérer efficaces si on les applique spécifiquement aux réseaux ATM. Bien que la simulation distribuée soulève des problèmes de synchronisation et de respect des contraintes de causalité, qui rendent cette tâche ardue, il est envisageable, d'un autre côté, de réduire les temps de simulation et ainsi d'augmenter la taille du système étudié. Deux types d'environnement différent sont rencontrés lors de l'évaluation de réseaux de communication. Le premier consiste à s'intéresser aux performances d'un système possédant de nombreuses composantes différentes, il s'agit par exemple de l'étude d'empilement de couches de protocoles dans des stations d'extrémités. Dans ce cas, la simulation distribuée semble peu performante en raison, d'une part, de la forte synchronisation des processus et d'autre part, de leur faible nombre et de leur variété. Le second type est celui d'un grand réseau de commutateurs, tels que ceux rencontrés auprès des opérateurs publics ou des réseaux privés. Dans ce contexte, les commutateurs sont tous similaires et reproduits en grand nombre et on peut alors envisager une réplication du code associé à chaque commutateur dans un *processus logique* (PL) de la simulation distribuée.

[1] Habituellement, l'expression "simulation parallèle" est utilisée lorsque la simulation est exécutée sur une machine parallèle à mémoire partagée, le terme "distribuée" désignant alors plutôt l'utilisation d'une machine à mémoire distribuée. Nous utiliserons plutôt le deuxième terme puisque nos simulations sont effectuées sur une CM-5 à mémoire distribuée.

3 La simulation distribuée

3.1 *Le paradigme de l'échange de messages*

Avant toute chose, il est nécessaire de définir plus précisement le cadre de notre étude. On peut caractériser un réseau ATM comme étant essentiellement un *système asynchrone* car les cellules sont envoyées d'un commutateur à l'autre sans aucune relation avec une horloge globale. Par conséquent, les techniques de simulation *synchrone* ou par *incrément fixe* ne pourront pas représenter fidèlement le comportement temporel du système. De plus, de telles approches sont difficilement réalisables dans un contexte distribué sans la connaissance précise de l'état global du système. Nous nous intéressons donc aux approches *asynchrones* et en particulier celles dites *discrètes* où les changements apportés au système se produisent à des points discrets dans le temps. Dans ce cas, le modèle va évoluer par bonds variables dans le temps selon la date d'occurence des événements.

Les simulations sont supposées être exécutées sur une architecture parallèle ou distribuée composée de n processeurs. La parallélisation consiste alors à diviser le système global en un ensemble de sous-systèmes disjoints, chacun d'entre eux étant simulé par un processus logique s'exécutant sur un processeur physique $p_i, 1 \leq i \leq n$. Dans ce cas, les interactions qui peuvent exister dans le système réel sont représentées par des échanges de messages estampillés par une horloge locale du PL source. Les messages sont reçus par le PL destinataire et placés dans sa liste locale d'événements (LEV) en attendant d'être traduits en l'événement correspondant et d'être exécutés. L'exécution d'un événement consiste en une éventuelle modification de l'état du PL et une éventuelle libération de nouveaux messages vers d'autres PLs. D'autres méthodes de paral-lélisation existent mais leur champ d'application est relativement limité (cf. [Fuj90, Fer94]). Celle à laquelle nous nous intéressons offre de manière générale un niveau de parallélisme supérieur.

Cette distribution des horloges et des listes d'événements pose des problèmes de respect de la causalité. Inexistants dans le cas séquentiel, ces problèmes apparaissent maintenant car les différents PLs peuvent évoluer indépendamment et se trouver à des instants de simulation différents. En particulier, une *faute temporelle* se produit lorsqu'un message arrive à un PL avec une estampille inférieure à l'horloge locale du PL. Intuitivement ce cas représente une incohérence puisque le message appartient au passé. C'est pourquoi des mécanismes explicites de synchronisation sont nécessaires pour garantir l'intégrité de la simulation; ceux-ci peuvent être classés en deux catégories: *conservatrice* et *optimiste*.

3.2 *Protocoles*

La première méthode proposée fut l'approche dite conservatrice [Cha79]. Elle évite les fautes temporelles en autorisant un PL p à exécuter un événement que lorsqu'il peut être déterminé avec certitude que son exécution ne pourra pas produire de fautes temporelles dans le futur. De ce fait, chaque PL i communicant avec p se voit affecter une file d'événements et l'algorithme consiste à sélectionner la file possédant l'événement d'estampille minimale. Si cette file est vide, un blocage se produit. Pour éviter les situations d'inter-blocage, des *null-messages* n'ayant aucune signification du point de vue du modèle peuvent être utilisés pour faire avancer artificiellement le temps de simulation. La réception d'un *null-message* d'estampille T sur un lien informe le récepteur qu'aucun message d'estampille inférieure à T ne pourra être reçu sur ce lien. Une autre solution consiste à laisser la simulation se bloquer pour entreprendre alors une phase de déblocage [Cha81, Cha83]. Dans les deux cas, une bonne capacité à "prédire" l'avenir (*lookahead*) est nécessaire pour propager rapidement les valeurs d'horloge et ainsi obtenir de bonnes performances.

D'un fonctionnement totalement différent, les approches optimistes, dont le plus connu des réprésentants est *Time Warp* [Jef85], ne font aucune tentative pour éviter les fautes temporelles. Ils instaurent par contre un contrôle réactif qui consiste en un mécanisme de retour-arrière (*rollback*) leur permettant de corriger la trajectoire de la simulation lorsqu'une faute survient. Les inconvénients majeurs de ces approches optimistes sont d'une part, la nécessité de sauvegarder périodiquement l'état du PL pour pouvoir le restaurer et d'autre part, l'utilisation d'anti-messages pour corriger la propagation des erreurs de trajectoire. De manière générale, les approches optimistes sont beaucoup plus complexes à mettre en œuvre que leurs homologues conservateurs, mais peuvent d'un autre côté exploiter de manière transparente le parallélisme présent dans le modèle. De nombreuses améliorations ont été proposées ces dernières années et une excellente synthèse peut être trouvée dans l'article de Fujimoto [Fuj90]. La diversité des variations reflète très bien la difficulté de la tâche et il n'existe pas un protocole de simulation parallèle mais une multitude de protocoles adaptés pour des applications particulières. L'intérêt récent pour les protocoles dits *adaptatifs* est indicateur d'un besoin urgent de protocoles "multi-usages". Cependant, dans un premier temps, nous nous intéressons à une approche conservatrice et cet article décrit l'implantation d'une telle approche à l'aide de la librairie PVM.

4 Modèle de réseau

Un réseau de communication ATM est typiquement composé de sources de trafic et de nœuds réalisant le transfert à très haut-débit d'informations numériques. Ces informations seront véhiculées sous la forme de cellules de taille fixe de 53 octets par des commutateurs ATM dont les deux principales fonctions sont la réception et l'émission de cellules. De fait, au niveau cellule, deux événements seront considérés dans notre modèle: un événement $< rec, t_r >$ indiquant la réception d'une cellule à l'instant t_r et un événement $< env, t_d >$ indiquant l'envoi d'une cellule à l'instant t_d. Nous allons dans la suite préciser les modèles de commutateurs et de réseaux utilisés dans cette étude (les lecteurs pourront également consulter [Gbu95]).

4.1 *Le modèle de commutateur*

Un grand nombre d'architectures a été proposé dans la littérature pour les commutateurs ATM. Les lecteurs peuvent se référer à [Pry93, Han94] pour une complète description des architectures existantes. Pour notre part, nous adoptons un modèle de commutateur à architecture non-bloquante avec des tampons en sortie (figure 1). La stratégie de commutation est alors la suivante:

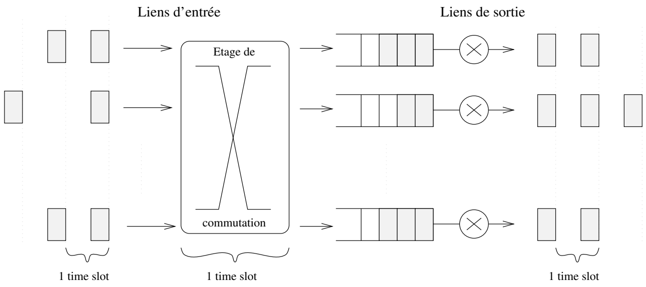

Figure 1. *Modèle de commutateur.*

A chaque *time slot*, une cellule arrivant sur un port d'entrée est commutée sur un port de sortie. Le choix du port de sortie est effectué de manière uniforme. Une seule cellule peut arriver sur un port d'entrée par *time slot* mais plusieurs cellules peuvent être commutées sur le même port de sortie au même instant. Dans ce cas, elles attendent dans les tampons de sortie. Parallèlement, les cellules en tête des tampons de sortie sont transmises sur le lien de sortie à chaque *time slot* en suivant une politique *First In First Out*. Au niveau de la simulation, l'arrivée d'une cellule sur un port d'entrée se traduit par la réception d'un événement $< rec, t_r >$. Le PL associé au commutateur programme ensuite pour lui-même un événement $< env, t_d >$ où t_d est l'instant de départ de la cellule du port de sortie. Un événement $< rec, t_a >$ sera ensuite envoyé au PL destinataire avec $t_a = t_d + delai\_lien$.

4.2 *Le modèle de réseau*

Le modèle de réseau que nous adoptons est un réseau de N_s commutateurs ATM complètement maillé. Une source de cellules est connectée à chaque commutateur et génère n_c cellules dont le temps d'inter-arrivée est distribué de manière exponentielle. Les liens entre deux commutateurs sont bi-directionnels, de ce fait chaque commutateur possède N_s entrées et $N_s - 1$ sorties. Lorsque les sources ont généré toutes leurs cellules, le nombre de cellulles dans le réseau reste constant et les commutateurs continuent d'envoyer les cellules jusqu'à la fin de la simulation. La figure 2 représente le modèle de réseau pour $N_s = 5$.

Le choix de ce modèle n'est pas arbitraire. En effet, comme nous l'avons dit dans le chapitre 2, nous nous intéressons surtout à l'évaluation de "grands

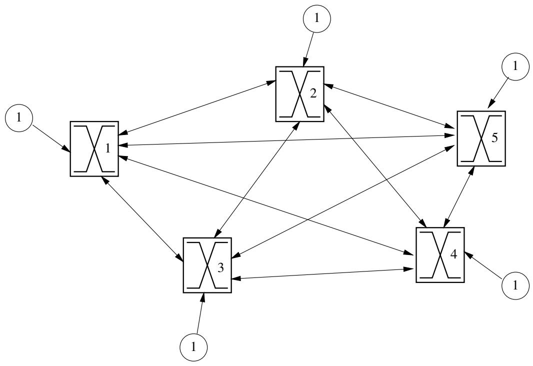

Figure 2. *Réseau complètement maillé pour $N_s = 5$*

réseaux de commutateurs, tels que ceux rencontrés auprès des opérateurs publics ou des réseaux privés". Nous pensons que la grande connectivité de notre modèle est représentative des réseaux fédérateurs futurs. Cette grande connectivité a des répercussions intéressantes sur le modèle que nous simulons. Premièrement, le nombre important de messages dans le système engendre un niveau de synchronisation très élevé entre les processus, ce qui constitue un cas très "stressant" pour un protocole de simulation distribuée. Deuxièmement, et c'est favorable, le routage uniforme des messages se traduit par une répartition équitable de la charge, qui nous évite par conséquent les problèmes liés à la sous-utilisation de certains processeurs tandis que d'autres sont surchargés.

5 Environnement d'étude

5.1 *PVM et CM/PVM*

Nous utilisons la librairie PVM v3.3 pour développer notre noyau de simulation. Cette librairie, pour langage C ou Fortran, offre des fonctionnalités d'échange de messages entre différents processus. Ces processus pouvant être situés soit sur le même processeur (exécution parallèle sur un seul processeur) soit sur des processeurs différents inter-connectés par un réseau quelconque de communication (machines reliées par un réseau Ethernet ou processeurs reliés par un réseau d'inter-connexion rapide). Le routage des messages est transparent à l'utilisateur qui ne connaît que le *tid* des processus.

Le grand nombre d'architectures supportées par PVM rend le portage d'une application très aisé d'une architecture à l'autre. En particulier, nous utilisons deux configurations pour exécuter nos simulations écrites en C. La première consiste en une SparcStation 20 monoprocesseur disposant de 64MB de mémoire centrale et la seconde en une Connection-Machine (CM-5) de 128 processeurs du *Centre National de Calcul Parallèle en Sciences de la Terre (CNCPST)*. La CM-5 est une machine parallèle à mémoire distribuée acceptant les modes de programmation SIMD et MIMD. Dans notre cas, c'est ce dernier mode qui est utilisé lorsque la synchronisation est effectuée par envoi de messages. Précisons que pour la CM-5 c'est l'implantation CM-PVM de PVM v3.3 qui est utilisée et qu'aucune modification du code source n'a été nécessaire pour passer d'une configuration à l'autre. Des précautions ont en outre été prises pour s'assurer que la simulation est bien la seule charge utilisatrice de la CM-5 lors des mesures.

Pour quantifier le gain en vitesse (speedup) réalisé en distribuant la simulation sur la machine CM-5, une implantation séquentielle optimisée est obtenue en utilisant une seule file d'événements et en remplaçant le code d'envoi des messages par des insertions dans cette file. Cette version séquentielle sera exécutée sur la machine SS20. Dans le reste de l'article, nous noterons par^{CM5} une simulation distribuée effectuée sur la CM-5 et seq^{SS20} la version séquentielle sur SS20.

5.2 *Initialisation de la simulation*

Le modèle de programmation est du type maître-esclaves. Un programme maître se charge de lancer l'ensemble des esclaves (PL commutateurs ATM) par un unique appel à la primitive `pvm_spawn()`. Il est important que l'appel à `pvm_spawn()` soit unique pour que l'ensemble des esclaves fasse partie du même processus afin de bénéficier de la vitesse maximale des communications sur CM-5 (figure 3). En effet, chaque appel à `pvm_spawn()` crée un nouveau processus avec des temps de communications inter-processus bien plus importants qu'entre deux tâches d'un même processus.

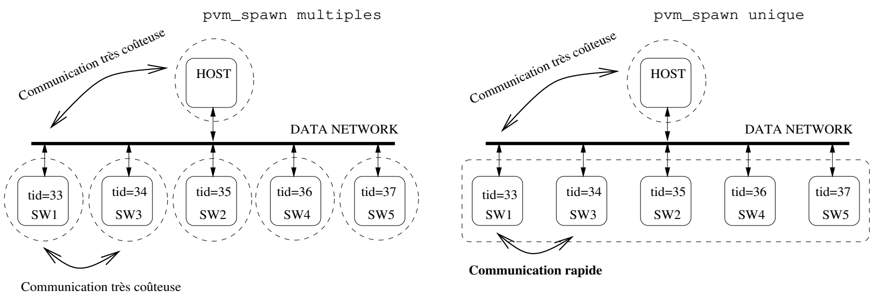

Figure 3. *Processus et tâches sur la CM5.*

Une fois lancée, chaque tâche esclave s'enregistre dans le groupe NETWORK et se voit alors attribuer un numéro d'instance dans ce groupe (étape 1, figure 4). La contrainte d'un appel unique à `pvm_spawn()` nous empêche de lancer les esclaves avec des paramètres en ligne différents. Les tâches esclaves se configurent donc en lisant un fichier unique de configuration et en cherchant les informations de la section correspondant à son numéro d'instance (étape 2). Ce fichier de configuration décrit la topologie du réseau et les caractéristiques de chaque composant; un exemple est décrit ci-dessous pour $N_s = 3$ et une configuration complètement maillée. On peut modifier la topologie du réseau (commandes `IN` et `LINK`) et les différents paramètres de manière quelconque.

```
## Fichier de configuration

[[ 0                      ## Numéro de section
SELF CONST 1.0 -1 -1 20   ## Type de la source
NODE SW0                  ## NODE nom
SWITCH 1 1000 3000        ## SWITCH debit nbrCell nbrTampons
LINK SW1 0 2.0            ## LINK nom sortie délai
LINK SW2 1 2.0
IN SW1                    ## IN nom
IN SW2
]]                        ## Fin de section

[[ 1                          [[ 2
SELF CONST 1.0 -1 -1 20       SELF CONST 1.0 -1 -1 20
NODE SW1                      NODE SW2
SWITCH 1 1000 3000            SWITCH 1 1000 3000
LINK SW0 0 2.0                LINK SW0 0 2.0
LINK SW2 1 2.0                LINK SW1 1 2.0
IN SW0                        IN SW0
IN SW2                        IN SW1
]]                            ]]
```

Une fois la configuration lue, les esclaves envoient au maître un couple $< tid_i, nom_i >$ (étape 3) composé de leur *tid*, obtenu en utilisant `pvm_mytid()`, et de leur nom symbolique (champ NODE du fichier de configuration, SW1 par exemple). Le maître récupère ces couples pour construire un tableau contenant tous les *tid*s et tous les noms symboliques des esclaves (étape 4). Il diffuse ensuite ce tableau à l'ensemble des esclaves qui peuvent alors récupérer les *tid*s qui leurs sont nécessaires (étape 5). Désormais toute communication avec le maître, qui sont très coûteuses, sont inutiles et celui-ci peut s'arrêter. Précisons que pour satisfaire aux contraintes imposées par le modèle de programmation de la CM-5, les sources de trafic sont encapsulées dans les commutateurs au lieu d'être lancées par ces derniers. A cette fin, chaque PL commutateur génère à l'initialisation n_c messages qu'il place dans la file d'entrée d'index 0. La figure 4 résume graphiquement l'ensemble des actions effectuées lors de l'initialisation. Une barrière de synchronisation à l'aide de `pvm_barrier()` est finalement placée pour s'assurer que tous les esclaves débutent leur simulation au même instant.

5.3 *Le noyau de simulation*

Notre moteur de simulation utilise l'approche conservatrice pour synchroniser les PLs entre eux. Des *null-messages* sont utilisés pour éviter les situations d'inter-blocage. Chaque PL possède N_s files d'entrées pour recevoir les messages émanant des $N_s - 1$ autres PLs et de leur source de trafic. La figure 5 illustre schématiquement la structure du moteur; l'algorithme général est décrit ci-après.

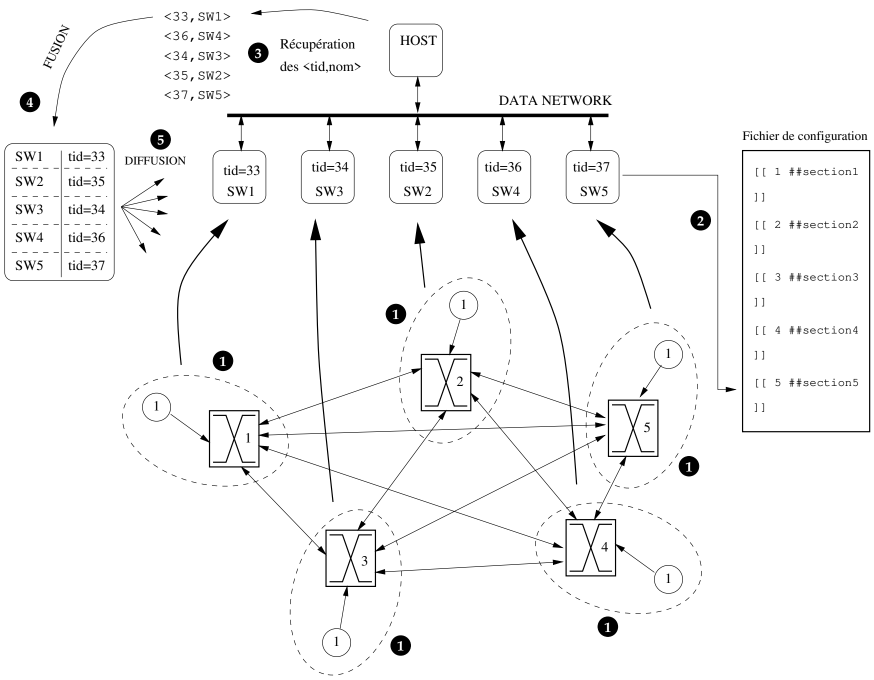

Figure 4. *Initialisation de la simulation.*

```
REPETER
 {
   récupérer_message();
   i=1;

   REPETER
    {
      SI NON vide(i) ALORS
        SI premier_message(i)=MNULL ALORS
          SI taille(i)>1 ET estampille_premier(i)<date_dernier_null ALORS
            enlève_premier_message(i);
            i=i-1;
        SINON
          h[i]=estampille_premier(i);
          i=i+1;

    } TANTQUE i<=n;

   hmin=h[1];
   j=1;

   POUR i=2 A n FAIRE
      SI hmin>h[i] ALORS
```

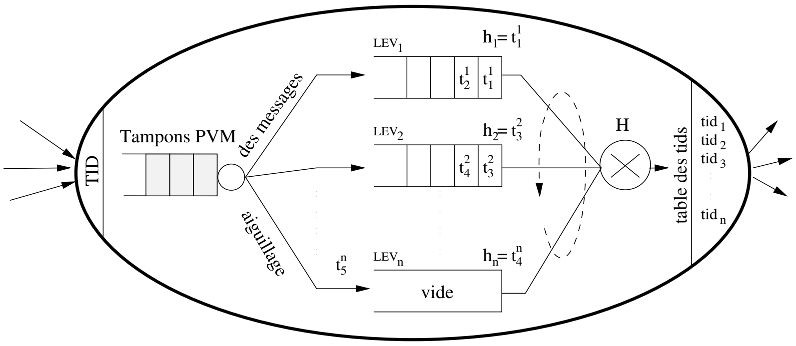

Figure 5. *Structure du moteur de simulation.*

```
23          hmin=h[i];
24          j=i;
25
26     SI vide(j) ALORS
27       envoyer_nullmessage();
28       attente_message(j);
29     SINON
30       executer(premier_message(j));
31
32   } JUSQUA fin_simulation();
```

La taille des messages échangés est relativement faible, inférieure à 100 octets, et ceux-ci sont envoyés en utilisant la primitive `pvm_send()`. Après chaque cycle d'exécution d'évènement, la récupération des messages se fait en utilisant la primitive `pvm_nrecv()` (ligne 3) et permet à tous les messages en attente dans les tampons de PVM d'être récupérés. Ils sont au fur et à mesure insérés dans la file d'entrée correspondant au PL émetteur. La détermination de la bonne file se fait grâce à une table indiquant la correspondance entre le numéro de la file et le numéro d'instance du PL émetteur, ce dernier étant contenu dans le message. Le PL peut alors déterminer la file j possédant l'horloge la plus petite (ligne 21-24) en réactualisant la valeur des horloges h_i pour les files i non vides (lignes 6-17). Si cette file j est vide, des *null-messages* sont envoyés vers les autres nœuds pour leur permettre d'avancer leurs horloges (ligne 26-28). A l'aide de la primitive bloquante `pvm_recv`, on attend alors qu'un nouveau message arrive (ligne 28). L'envoi et la réception des *null-messages* ne diffèrent pas de ceux des messages réels, cependant, pour réduire leur trafic, un filtrage est effectué lors de la phase de réactualisation des horloges. Un premier filtrage consiste à ôter les *null-messages* lorsque d'autres messages (*null-messages* ou messages réels) sont présents dans la file. Le second filtrage consiste à ne retenir que les *null-messages* dont l'estampille est supérieure à celui du dernier *null-message* exécuté (ligne 10-11).

6 Résultats

Le code parallèle a été exécuté sur la SS20 et la figure 6 représente le speedup de la version séquentielle sur la version parallèle tournant sur un processeur (notée par^{SS20}). Nous avons fait varier d'une part, la densité de messages (en abscisse) définie comme étant le rapport du nombre total de messages dans le système et du nombre de commutateurs (dans notre cas, on trouve n_c) et d'autre part, la taille N_s du réseau. Les résultats obtenus sont pour une

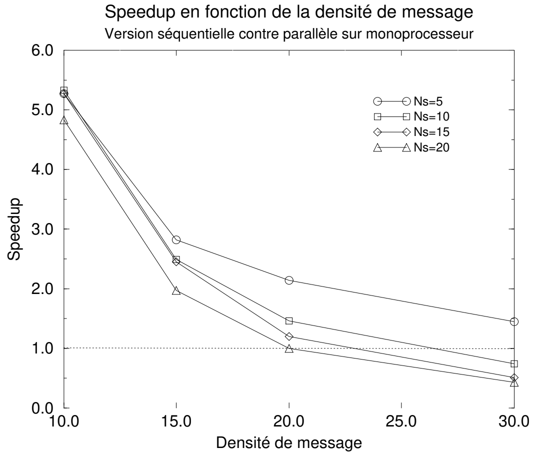

Figure 6. *Version séquentielle et version parallèle monoprocesseur*

grande part prévisibles: la version séquentielle va presque toujours plus vite que son homologue parallèle tournant sur un seul processeur puisque ce dernier comporte des mécanismes de synchronisation et de passage de messages très lourds. Néanmoins le "presque toujours" est étonnant car on s'attendrait plutôt à un "toujours". On constate en effet que pour $N_s = 20$ et $n_c = 20$, seq^{SS20} va aussi lentement que par^{SS20}. De plus lorsque l'on augmente n_c jusqu'à 30, seul le cas $N_s = 5$ permet à seq^{SS20} d'aller plus vite que par^{SS20}. En fait ces résultats illustrent très bien les limitations de la version séquentielle lorsque la taille du réseau en général devient grande. Ces limitations ont essentiellement pour origine l'explosion de la taille de la file d'attente des événements qui entraîne une chute des performances. La distribution de la simulation, bien qu'introduisant des mécanismes lourds, a l'avantage de distribuer les files d'événements, ce qui donne des files moins grandes. Ce cas, bien que rarement présenté, est néanmoins intéressant puisqu'il montre que la parallélisation peut être avantageuse même en utilisant une machine monoprocesseur. Cependant, il nous a été impossible de simuler plus de 20 commutateurs avec par^{SS20} à cause du nombre élevé de processus que cela demande.

Dans un deuxième temps la CM-5 est utilisée et les mesures de speedup sont obtenues avec seq^{SS20} et par^{CM5}. La figure (a) représente le temps d'exécution, exprimé en s, de seq^{SS20} et la figure (b) celui de par^{CM5}. Comme précédemment, nous avons fait varier la densité de messages et la taille du réseau. Ces temps d'éxécution ont été obtenus en simulant 3.000 cellules à chaque commutateur (le temps d'exécution devient alors linéaire en fonction du nombre de cellules).

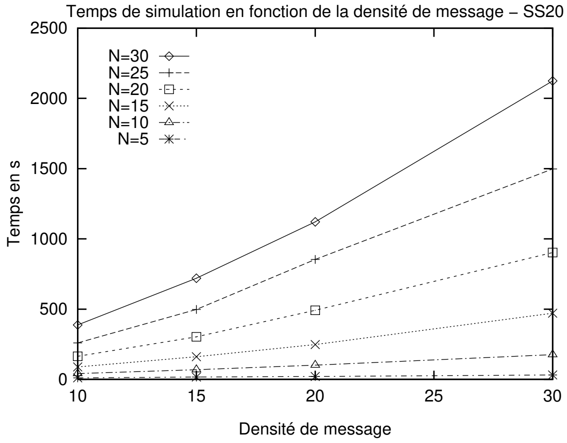

Figure 7. *Simulation séquentielle sur SS20*

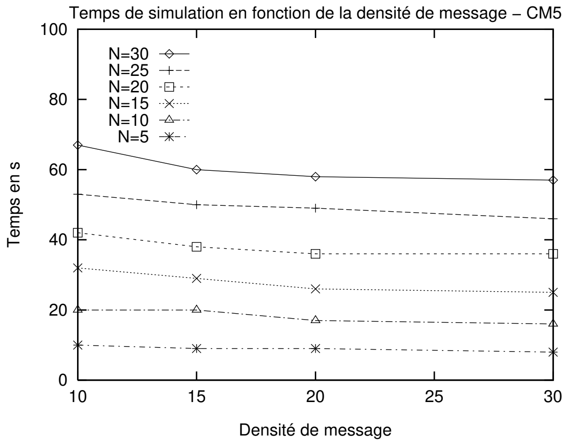

Figure 8. *Simulation parallèle sur CM5*

Les résultats sont les suivants: au fur et à mesure que la taille du réseau et la densité de messages augmentent, le temps d'exécution de seq^{SS20} augmente très rapidement. D'un autre côté, on constate que la version parallèle demeure pratiquement insensible aux variations de taille lorsque le nombre de PLs reste inférieur au nombre de processeurs physiques disponibles, comme c'est ici le cas. On constate même que par^{CM5} s'avère moins performante lorsque la densité de messages est faible. Ce phénomène s'explique simplement par l'augmentation du trafic des *null-messages* nécessaires pour éviter les blocages lorsque le nombre de messages dans le système est petit. La figure 9 représente les valeurs de speedup que l'on obtient. On peut alors constater que la version séquentielle est toujours plus lente que la version parallèle, même pour le cas d'un réseau de faible dimension ($N_s = 5$ et $n_c = 10$). On parvient même à obtenir pour une grande densité de messages, $n_c = 30$, des valeurs de speedup sur-linéaires à partir d'une taille de réseau de 10 noeuds. Cela s'expliquent encore une fois par une dégradation très nette des performances de la version séquentielle lorsque la densité de messages et la taille du réseau augmentent. Ces premiers résultats sont très encourageants et montrent que l'utilisation d'une approche conservatrice relativement simple nous permet déjà d'une part, de réduire considérablement le temps de simulation et d'autre part, d'augmenter la taille du système sans la contrepartie d'une dégradation majeure des performances.

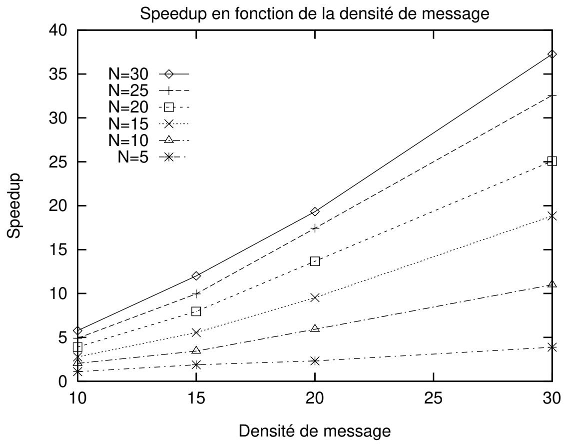

Figure 9. *Speedup entre parallèle multiprocesseur et séquentielle*

7 Conclusion

Dans cet article, nous avons montré la nécessité d'utiliser des techniques de simulation parallèle pour évaluer les futurs réseaux fédérateurs large bande. L'utilisation d'une approche conservatrice montre que de très forts speedup peuvent être obtenus. En particulier, il est clairement apparu qu'une approche séquentielle est très rapidement "débordée". Nous souhaitons étendre ces travaux et évaluer dans quelles situations (topologie du réseau, type de trafic, critères de performances étudiés) un "speedup" d'un ordre de grandeur peut être obtenu. Une application pour des problèmes réels (routage) sera étudiée en détail.

Pour obtenir de bonnes performances lors de la distribution de la simulation, il est bien sûr nécessaire que les procédures d'envoi et de réception de messages soient elles-mêmes performantes. Pour la CM-5, l'implantation originale de PVM ne profitait pas pleinement des possibilités de la machine et nous obtenions des temps de communications excessifs. La version CM/PVM remédie à ce problème et les performances ont pu être grandement améliorées. Dans ce cas, la souplesse apportée par l'utilisation d'une librairie comme PVM est fortement appréciable puisque de manière générale elle a permis de réduire considérablement le temps de développement.

Références

[Cha79] K. M. Chandy and J. Misra,"Distribution Simulation: A Case Study in Design and Verification of Distributed Programs", *Transactions on Software Engineering, Vol. 5(5), May 1979, pp440-452.*

[Cha81] K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations", *Communication of the ACM, Vol. 24(11), April 1981, pp198-206.*

[Cha83] K. M. Chandy, J. Misra and L. M. Haas, "Distributed Deadlock Detection", *ACM Transactions on Computer System, May 1983, pp144-156.*

[Fer94] A. Ferscha and S. K. Tripathi, "Parallel and Distributed Simulation of Discrete Event Systems", *CR-TR-3366, Department of Computer Science, University of Maryland, August 94.*

[Fuj90] R. M. Fujimoto, "Parallel Discrete Event Simulation", *Communication of the ACM, Vol. 33(10), October 1990, pp31-53.*

[Gbu95] P. Gburzynski, T. Ono-Tesfaye and S. Ramaswamy, "Modeling ATM Networks in a Parallel Simulation Environment: A Case Study", *Proceedings of the SCSC'95 Conference, Ottawa.*

[Han94] R. Händel, M. N. Nuber and S. Schröder, "ATM Networks: Concepts, Protocols, Applications", *Addison-Wesley, 1994.*

[Hop95] H. C. Hoppe, P. Ossadnik and W. Stüttgen, "CM-PVM: An efficient Implementation of PVM 3.3 for the CM-5", *Proceedings of the EuroPVM'95, pp53-58.*

[Jef85] D. R. Jefferson, "Virtual Time", *ACM Transaction on Programming Languages and Systems, Vol. 7(3), July 1985, pp405-425.*

[Law91] A. M. Law and W. D. Kelton, "Simulation Modelling & Analysis, Second Edition", *MacGraw Hill, 1991.*

[Pry93] M. DePrycker, "Asynchronous Transfer Mode : Solution for Broadband ISDN", *Second edition, Ellis Horwood, 1993.*

[Pvm93] A. Geist and al., "PVM 3 User's Guide and Reference Manual".