République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE AMAR TELIDJI

LAGHOUAT

FACULTE DES SCIENCES ET SCIENCES DE L'INGENIEUR

DEPARTEMENT DE GENIE INFORMATIQUE

PROJET DE FIN D'ETUDES

Pour l'Obtention Du Diplôme

D'INGENIEUR D'ETAT EN INFORMATIQUE

Option : Systèmes d'informations avancées

Thème :

Simulation et amélioration d'un algorithme réparti de point de reprise (Algorithme de Cao et Singhal)

Réalisé par :

Benmouiza Yamina

Chennouf Oumelkhier

Proposé et encadré par :

Melle. Zohra ABDELHAFIDI.

N° d'ordre : **08/2011-PFE/DGI**

Dédicaces

À nos parents,

qu'ils trouvent ici le fruit de leur patience et du soutien permanent et quotidien qu'ils nous ont prodigué pour affronter tous les moments difficiles.

À nos familles et nos amis

sans oublier ceux qui ont participés de prés ou de loin à la réalisation de ce travail.

Remerciement

Nous devons tout d'abord remercie Dieu

notre créateur, pour le courage et la patience qui nous a donné

afin de mener ce projet à terme.

Nous ne saurions oublier de trop remercier nos familles

pour leur encouragement et leur soutien tout au long de ce

parcours.

Nous tenons à exprimer

notre profonde gratitude et reconnaissance à l'égard de

notre encadreur Melle. ABDELHAFIDI Zohra pour son aide, sa

constante disponibilité et ses précieux conseils qui ont permis ce

travail de voir le jour.

Nous remercions les enseignants

qui nous avons fait l'honneur de participer au jury de ce mémoire.

Table des matières

Liste des figures

Liste des tableaux

Résumé

Dans ce mémoire, nous traitons le problème de calcul d'état global dans les systèmes répartis. Pour cela nous avons choisi des protocoles de la classe coordonnée.

A cet effet, nous avons simulé les algorithmes de Guohong Cao et Mukesh Singhal, le premier est bloquant et le deuxième est non bloquant. Pour voir des meilleures performances nous avons proposé des améliorations au deuxième algorithme.

Nous avons évalué les performances selon des critères tel que le nombre de requêtes, la durée de la première phase et le nombre des points de reprise mutable pour cela nous avons varié le nombre de processus, le nombre d'initiateur, et le nombre des messages.

Mots-clés : Système réparti, NS2, point de reprise, tolérance aux fautes.

Abstract

In this manuscript, we treat the problem of calculating global state of distributed systems. This is why we choose the class of coordinated protocols.

To this end, we have simulated the algorithms of Guohong Cao and Mukesh Singhal, the first one is blocking and the second is not. To get best performances we have proposed new improvements to the second algorithm.

We have evaluated the performances based on various criteria such as number of requests, the duration of the first phase and the number of mutable checkpoints, so we vary the number of processes, the number of initiator, and the number of messages.

Keywords: Distributed system, NS2, checkpointing, fault tolerance.

Introduction générale

Les progrès remarquables dans les systèmes informatiques pendant les dernières années ont permis une forte évolution des systèmes qui sont caractérisées par une tendance très forte vers la décentralisation, et alors un très nombre des problèmes tels que la tolérance aux fautes .

Par ailleurs, l'exigence de tolérance aux fautes est incontournable avec les systèmes répartis, Car la tolérance aux fautes est un besoin induit par la multiplicité des ressources et de nombreux travaux sur ces systèmes pour garantir la sûreté de fonctionnement.

L'absence d'état global est l'une des caractéristiques essentielle (et malheureuse) des systèmes répartis. Un site ou processus n'a qu'une connaissance approximative de l'état des autres sites ou processus puisque, seul, l'échange de messages permet d'obtenir des informations sur les partenaires distants (logiquement).

Parmi les techniques permettant d'assurer la tolérance aux fautes dans les systèmes répartis est le calcul d'état global « Checkpointing », cette technique peut être classifiées en trois catégories :

- Approche non coordonnée (Uncoordinated Checkpointing).
- Approche coordonnée (Coordinated Checkpointing).
- Approche induit par communication (Communication-Induced Checkpointing).

Notre objectif est de simuler des algorithmes de deuxième classe, ces algorithmes permettent de calculer l'état global dans les systèmes répartis, ces algorithmes (CSB, CSNB) sont proposés par G.Cao et M.Singhal [3], nous avons essayé proposé des améliorations dans CSNB.

Nous avons utilisé un outil de simulation appelé Network Simulator (NS-2), ce simulateur permet d'ajouter des protocoles. Il est exécutable tant sous Unix que sous Windows.

Ce mémoire est composé de trois chapitres et deux annexes:

- Le premier chapitre est donné sous forme de généralités sur les systèmes répartis avec leurs inconvénients et leurs avantages, et leurs problèmes, la tolérance aux pannes, les techniques de checkpointing, et ses classifications.

- Dans le second chapitre, nous présentons les algorithmes choisis pour la simulation, ainsi que les descriptions des améliorations.

- Le troisième chapitre regroupe les étapes de simulation et l'intégration des algorithmes simulés dans NS-2, les résultats de simulation obtenus sont représentés par des courbes et sont analysés et interprétés.

À la fin de ce mémoire, une conclusion résume ce mémoire.

Systèmes répartis et tolérance aux pannes 1

Dans ce chapitre on va présenter les systèmes répartis, leurs avantages et leurs inconvénients, leurs problèmes tels que la tolérance aux pannes.

1.1 Système réparti

1.1.1 Définitions d'un système réparti

Un système réparti [1] est défini comme un ensemble de nœuds de calcul ou processeurs interconnectés par un système de communication et communiquant uniquement par messages.

1.1.2 Les avantages et les inconvénients d'un système réparti

1.1.2.1 Avantages d'un système réparti

Le système réparti offre un ensemble d'avantages [7] :

- **Partage et Mise à disposition :**

La répartition est avant tout un moyen de mise à disposition et donc de partage de ressources et services. Cette idée de partage et de disponibilité constitue même la sémantique que l'on peut donner au mot répartition.

- **Répartition géographique :**

La répartition est un moyen essentiel pour mettre à disposition des usagers les moyens informatiques locaux dont ils ont besoin tout en gardant la possibilité d'accéder aux ressources et services distants de leurs collègues.

- **Puissance de calcul :**

La connexion de machines en réseau permet d'obtenir à moindre coût une puissance de calcul importante. Bien entendu, l'exploitation optimale de ces performances potentielles nécessite de paralléliser les algorithmes de calcul et de disposer d'un environnement d'exécution répartie adéquat.

- **Disponibilité :**

La disponibilité d'un service développé sur une architecture répartie peut être rendue plus grande que celle d'un service centralisé. La relative indépendance des défaillances qui peuvent survenir dans les nœuds, permet de continuer à offrir un service même dégradé. En particulier, la réplication d'un même service par l'installation de plusieurs serveurs équivalents est une solution classique mais de mise en œuvre délicate notamment si les serveurs sont à état rémanents.

- **Flexibilité:**

Une architecture répartie est par nature modulaire. Il est donc plus facile d'ajouter ou d'enlever un nœud connecté au réseau, cette flexibilité se situe aussi au niveau logique. Un nouveau service peut être installé sur un nœud sans nécessité d'une reconfiguration des autres nœuds.

1.1.2.2 Inconvénients d'un système réparti

Les principaux inconvénients d'un système réparti sont regroupés dans les points suivant [**7**] :

- **Pas d'état global :**

Dans une architecture répartie, un nœud ne possède pas une connaissance immédiate exacte de l'état d'un autre nœud. En effet, cette connaissance passe par l'échange d'un message qui introduit obligatoirement un délai. Un nœud connaît son état local et n'a qu'une connaissance du passé (certes parfois très proche) des autres. Cette difficulté conduira à la recherche d'algorithmes qui permettent de construire des états globaux qui représenteront un état passé possible de l'application.

- **Pas d'horloge globale :**

Chaque nœud possède sa propre horloge pour dater les événements qui lui sont locaux. Par conséquent, si les horloges indépendantes de chaque nœud ne sont pas parfaitement synchronisées, l'ordre des événements répartis dans l'application n'est pas déductible à partir des datations locales. Cette difficulté conduira à définir des datations logiques qui permettent de corriger ce problème.

Ces deux propriétés augmentent fondamentalement la difficulté de compréhension et d'analyse des applications réparties.

- **Sécurité relative :**

Une architecture répartie est plus difficile à protéger contre les malveillances qu'une architecture centralisée. Ceci pour deux raisons:

- les points d'accès aux ressources du système global sont multiples et souvent « hors les murs ». Autrement dit, l'utilisateur doit être authentifié.
- le réseau de communication est un point d'accès potentiel pour toute tentative d'intrusion.

1.1.3 Problèmes liés aux systèmes répartis

À cause de type de communication dans les systèmes répartis, on peut lister les problèmes suivants [6] :

- **L'élection :**

Certaines applications réparties imposent qu'un site donné parmi les sites du système joue un rôle particulier, ce site est appelé le coordinateur, et il assure la communication entre tous les sites. La panne du site coordinateur entraîne la fin de l'application utilisant ce coordinateur, les autres sites doivent désigner un autre site pour remplacer le coordinateur, ce nouveau site est choisi après une opération d'élection.

- **Détection de terminaison :**

Les sites d'un système réparti peuvent réaliser des calculs répartis, le problème de terminaison est celui de détection de la fin de ce calcul réparti. On dit qu'un calcul est terminé si tous les sites qui ont participé dans ce calcul ont terminé le calcul, et il n y a aucun message en transit entre ces sites.

- **L'exclusion mutuelle :**

Les sites peuvent utiliser les ressources partagées dans le système réparti, par exemple, plusieurs sites peuvent utiliser une seule imprimante, mais on ne peut pas avoir deux sites différents qui utilisent cette imprimante simultanément, l'accès aux ressources critiques doit être en exclusion mutuelle afin de garantir la cohérence de données, c'est à dire, à un moment donné, un seul site au plus peut utiliser la ressource partagée.

- **Calcul d'état global du système :**

Chaque processus dans le système n'a qu'une vue partielle du système, si un problème a lieu dans le système, il est difficile de revenir à une situation cohérente pour tous les sites, on doit donc avoir un état global cohérent.

La définition d'un état global cohérent permet de calculer et de mémoriser un point de reprise cohérent, le système va revenir à ce point en cas d'un problème.

1.2 La tolérance aux pannes

1.2.1 Définition

La tolérance aux pannes (fautes) est l'ensemble des techniques de conception des systèmes qui continuent de fonctionner même en présence de la panne de l'un de leurs composants.

D'une autre part, un ensemble d'architectures, considérée comme un tout, continue par l'utilisation de redondances de rendre le service attendu en dépit de l'existence de fautes. [12]

1.2.2 Les techniques de tolérance aux pannes

La tolérance aux pannes dans les systèmes répartis est toujours réalisée par l'emploi d'un mécanisme de redondance. Cette dernière peut être spatiale, temporelle ou bien informationnelle.

Dans un tel système à base de processus communicants, les techniques de tolérance aux pannes peuvent être séparées en deux classes: les techniques basées sur la duplication et les techniques basées sur une mémoire stable. [5]

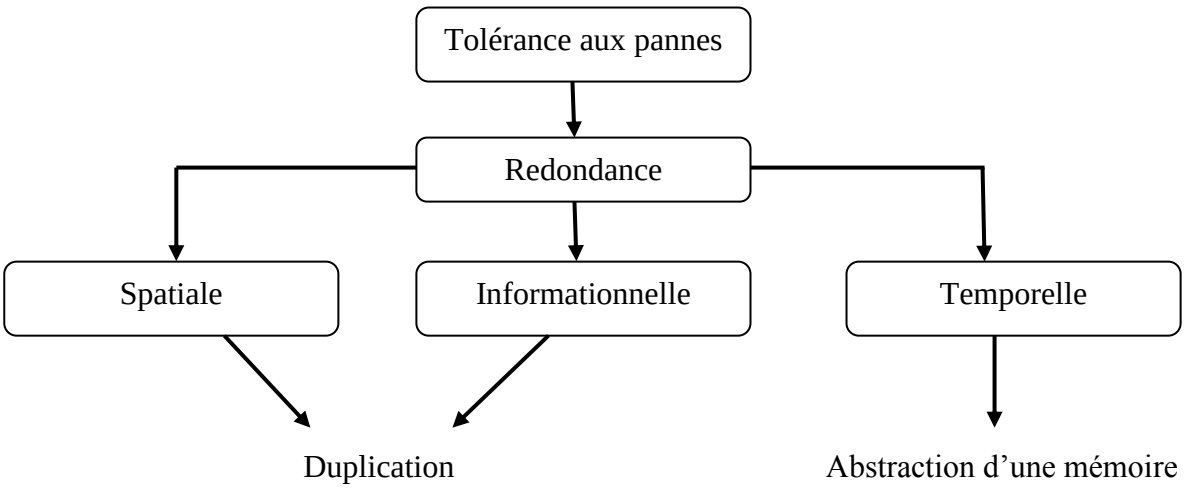


Figure 1. 1 Les techniques de tolérance aux pannes dans les systèmes répartis

1.2.2.1 Tolérance aux pannes par duplication

La tolérance aux fautes par réplication (duplication) consiste à utiliser des copies multiples d'un même composant ou processus. De cette manière, en cas de défaillance l'un des

composants, la défaillance peut être masquée par l'une des copies. La principale difficulté de cette approche est de conserver une cohérence forte entre les copies. [4]

1.2.2.2 Tolérance aux pannes par mémoire stable

L'existence d'une mémoire stable permet la réalisation de la tolérance aux pannes par une détection de défaillance suivie d'un recouvrement d'erreur. [4]

Définition :

La mémoire stable représente un support de stockage. Son rôle est de conserver les sauvegardes des informations du système qui permettront de reprendre l'exécution de l'application dans un état cohérent. Une mémoire stable doit préserver l'intégrité des données et les garder accessibles, même en cas de défaillance.

La réalisation physique d'une mémoire stable dépend essentiellement des types de défaillances auxquels on souhaite faire face. **[**11**]**

Principe :

Le principe de cette technique est de remplacer l'état d'erreur par un état correct en se basant sur l'abstraction d'une mémoire stable présentée dans le paragraphe ci-dessus. Ceci nécessite tout d'abord la détection de l'erreur, c'est à dire, identifier la partie incorrecte de l'état, afin de pouvoir ensuite effectuer le recouvrement d'erreur.

Il existe deux techniques de recouvrement:

- la reprise (rollback recovery): c'est la technique générale qui consiste à retourner vers un état antérieur dont on sait qu'il est correct. Cette technique nécessite donc la sauvegarde régulière de l'état du système ou de l'application.
- la poursuite (forward recovery) : c'est une technique spécifique qui consiste à reconstituer un état correct, sans retour en arrière. La reconstitution n'est souvent que partielle, d'où un service dégradé. [4]

1.2.3 Eléments de base sur les points de reprise

À partir que nous l'avons déjà dit, on peut tirer des notions fondamentales qui sont le point de reprise local , le point de reprise global.

1.2.3.1 Point de reprise local

Lorsqu'un processus enregistre son état dans la mémoire stable, on dit qu'il a pris un point de reprise. Le $x^{ième}$ point de reprise local d'un processus Pi est noté par $C_{i,x}$.

Intervalle: la séquence des événements produits par un processus Pi entre deux points de reprise successifs $C_{i,x-1}$ et $C_{i,x}$ est appelée intervalle noté $I_{i,x}$. [5]

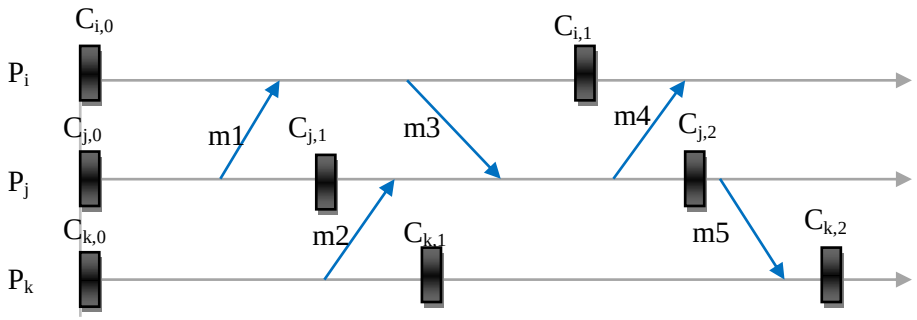


Figure 1. 2 Les points de reprise

1.2.3.2 Point de reprise global cohérent

Un point de reprise global est un ensemble des points de reprises locaux $\{C_{1,x1},....,C_{n,xn}\}$, un par processus.

La notion de cohérence du point de reprise global peut être formellement décrite par l'utilisation de la relation de précédence entre les points de reprise.

Définition 1 : Un point de reprise Ci,x du processus Pi, précède un point de reprise Cj,y de Pj, s'il existe un message m , tel qu'il est envoyé par Pi après Ci,x et reçu par Pj avant Cj,y. Ce message m est dit orphelin par rapport à la paire {Ci,x, Cj,y}.

Définition 2 : Un point de reprise global $\{C_{1,x1},....,C_{n,xn}\}$ est cohérent si et seulement si pour chaque message m tel que sa réception est enregistrée dans le point de reprise global, son émission est aussi enregistrée.[5]

Dans la figure 1.2, le point de reprise global $\{C_{i,1},C_{j,1},C_{k,1}\}$est cohérent, par contre le point de reprise global $\{C_{i,1}, C_{j,2}, C_{k,2}\}$n'est pas cohérent car le message m5 est orphelin.

1.2.4 Classification des protocoles de points de reprise

Les protocoles de points de reprise réalisent des calcules régulière de l'état des processus, pour redémarrer, ils utilisent le dernier ensemble de points de reprise formant un état global cohérent. **[**11**]**

Les protocoles de points de reprise peuvent être classés en trois catégories selon le mode de construction de l'état global cohérent de la reprise :

- Point de reprise non coordonnée (Uncoordinated Checkpointing).
- Point de reprise coordonnée (Coordinated Checkpointing).
- Point de reprise induit par communication (Communication-Induced Checkpointing).

1.2.4.1 Protocoles incoordonnés (Uncoordinated Checkpointing)

La technique de sauvegarde non coordonnées de points de reprise vise à maximiser les performances en fonctionnement normal. L'objectif est de minimiser le surcoût lié à la synchronisation lors d'une exécution sans fautes [9], pour cela chaque processus prend des points de reprise d'une manière indépendante.

Mais l'inconvénient majeur de cette méthode est le risque d'effet domino qui peut causer une perte importante du travail réalisé avant la panne et la sauvegarde inutile des points de reprise. [5]

- **L'effet domino**

Le problème de l'effet domino apparaît lorsque chaque processus prend des points de reprise sans coordination avec les autres processus, il est caractérisé par une cascade de retours en arrière lors de la reprise du système après une panne.

Lors de la panne du processus, le système va tenter de reprendre depuis le point de reprise global le plus récent [9]

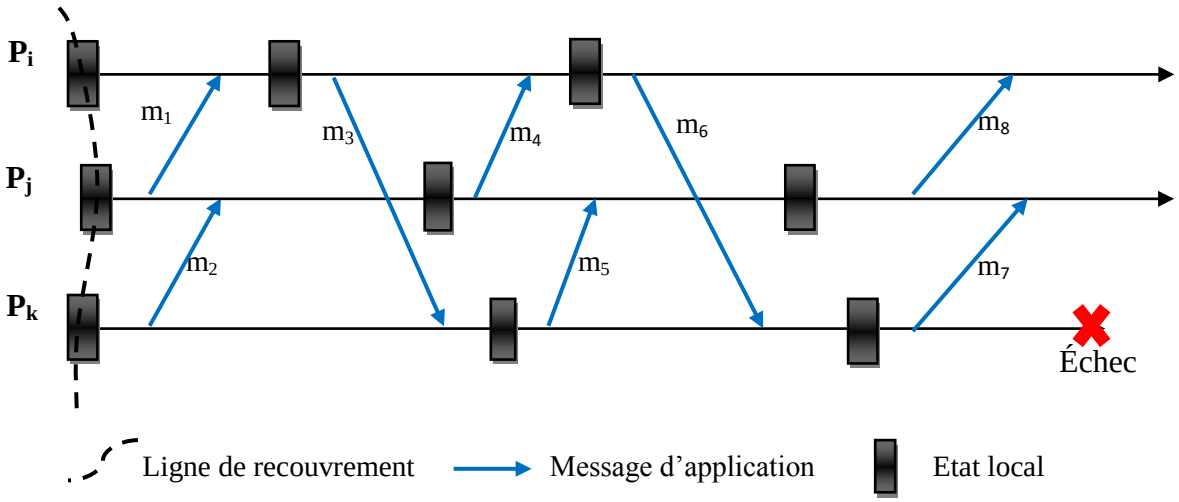


Figure 1. 3 L'effet domino

Pour éviter ce problème, les processus doivent coordonner leurs travaux pour construire un état cohérent du système.

1.2.4.2 Protocoles coordonnés (Coordinated Checkpointing)

Dans cette approche, les processus coordonnent leurs travaux pour assurer la cohérence de l'état global. Il existe différentes manières de coordonner les processus. On distingue en particulier [5] :

- Approche coordonnée bloquante.
- Approche non bloquante.

Les protocoles coordonnés bloquants :

Une manière pour calculer le point de reprise global (Figure 1.4) est de bloquer l'application pendant que les processus prennent les points de reprise.

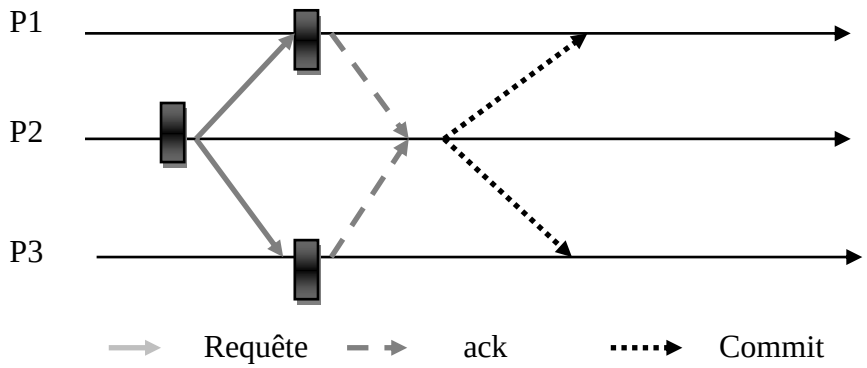


Figure 1. 4 Coordination entre les processus

1. Un initiateur prend un point de reprise et diffuse des requêtes (Requête) vers tous les processus.

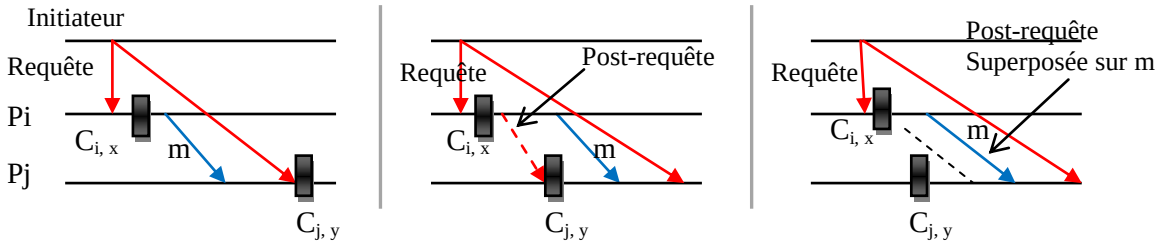
2. Un processus qui reçoit un tel message, il se bloque, prend un point de reprise et envoie un acquittement (ack) vers l'initiateur.

3. Après la réception de tous les acquittements, l'initiateur envoie des messages de validation (Commit) vers les processus.

4. Un processus qui reçoit un message de validation, il supprime l'ancien point de reprise et considère le nouveau point de reprise comme permanent. Ainsi il pourra poursuivre son exécution interrompue.

Les protocoles coordonnés non bloquant :

Une alternative à l'approche bloquante est la coordination non bloquante. Dans ce type de coordination les processus ne bloquent pas l'application courante pour prendre des points de reprise. Cette approche est basée sur la superposition des informations de contrôle aux messages d'application et l'utilisation des messages de contrôle.



(a) incohérence entre points de reprise (b) cas FIFO (c) cas non FIFO

Figure 1. 5 Approche non bloquante

- ***Cas des canaux FIFO***

Le problème est posé quand un processus Pj reçoit un message du processus Pi après que ce dernier ait reçu une requête, une telle situation est illustrée dans la figure 1. 5 (a).

Le message m est envoyé par le processus Pi au processus Pj après la réception de la requête, si ce message arrive à Pj avant la réception de la requête, il peut conduire à l'incohérence de point de reprise global qui contient $C_{i,x}$ et $C_{j,y}$ due à la présence de m qui vérifie la relation de dépendance entre $C_{i,x}$ et $C_{j,y}$.

Une stratégie adoptée par Chandy et Lamport, pour résoudre le problème précédent est faite de la manière suivante: un processus Pi est forcé d'envoyer une post-requête avant d'envoyer

un message m, tout processus Pj prend un point de reprise dès l'arrivée de la première requête. Dans ce cas un processus Pj peut consommer le message m sans risque de l'incohérence du point de reprise global (Voir Figure 1. 5(b)).

- ***Cas des canaux non FIFO***

Une adaptation de la solution de Chandy et Lamport pour les canaux non FIFO est fondée sur le principe de superposition de post-requête sur les messages d'application. Dans la figure 1. 5 (c), la post-requête (du processus Pi vers le processus Pj) est superposée au message m, le processus Pj prend un point de reprise avant la consommation d'un tel message. Il est possible de remplacer les post-requêtes par des nombres de séquence, un processus Pj prend un point de reprise lorsque son nombre de séquence est inférieur à celui superposé au message m.

L'avantage du point de reprise coordonné est qu'elle n'est pas sensible à l'effet domino lors de la reprise. Seule la dernière sauvegarde est nécessaire pour un redémarrage ce qui réduit le surcout de stockage. Le principal inconvénient est le surcoût induit par la synchronisation des processus.

Une amélioration de ce protocole présentée par Cao et Singhal minimise potentiellement le temps de blocage et de minimiser le nombre de points de reprise au cours de points de reprise. (Voir chapitre2)

1.2.4.3 Protocoles de type CIC (Communication-Induced Checkpointing)

Cette classe de protocoles est un compromis entre les protocoles coordonnés et les protocoles incoordonnés.

Chaque processus prend régulièrement des points de reprise de manière indépendante. La synchronisation se fait via l'utilisation des messages de l'application (superposition des informations supplémentaires).

Cependant, en fonction des messages reçus, envoyés et des informations qui sont superposées sur ces messages, le processus devra peut-être prendre un point de reprise additionnel. Ces protocoles assurent que le point pris fasse partie d'un point de reprise global cohérent, et donc qu'il existe une ligne de recouvrement toujours assez récente. [8]

Description des algorithmes simulés 2

***D**ans ce chapitre, nous présentons l'algorithme CSB qui bloque l'application lors du calcul d'état global, et l'algorithme CSNB qui corrige le problème de blocage et utilise un nouveau terme (point de reprise mutable), et on a essayé de proposer des améliorations dans CSNB tout en essayant de minimiser le nombre des requêtes et la durée de la première phase.*

2.1 Introduction

La Classe Coordonnée est une approche intéressante dans la tolérance aux pannes pour les applications distribuées, car elle évite l'effet domino et minimise les besoins de stockage stable.

Dans cette approche, l'état de chaque processus dans le système est sauvegardé sur le stockage stable, ce qui est appelé un point de reprise (Checkpoint) du processus, les processus doivent synchroniser leurs activités. En d'autres termes, quand un processus prend un point de reprise, il demande (en envoyant des requêtes) à tous les processus partenaires de prendre des points de reprise.

2.2 Algorithme de CAO et Singhal Bloquant (CSB)

Le protocole CSB (l'algorithme min-processus) est proposé par G.Cao et M.Singhal en 2003, cet algorithme force un nombre minimal de processus dans le système pour bloquer leurs calculs, pour prendre des points de reprise. [3]

2.2.1 Les messages utilisés

Dans cet algorithme, on utilise six types de messages :

- **R_request** : Message envoyé par l'initiateur à tous les processus pour collecter les dépendances.
- **Vecteur_R** : Message envoyé par le processus ayant reçu la demande R_request, il contient le vecteur de dépendance.
- **Request** : Après le calcul de dépendance l'initiateur envoie ce message aux processus dont il dépend pour demander de prendre un point de reprise.
- **Continue** : Ce message est envoyé par l'initiateur aux processus n'ayant pas une dépendance avec lui, il permet de débloquer les processus.
- **Reply** : Ce message envoyé par le processus ayant reçu le message requête.
- **Commit** : Message envoyé par l'initiateur pour finaliser le calcul d'état global.

Notez que Les informations de dépendance sont enregistrées par un vecteur booléen Ri pour le processus de Pi. Le vecteur a n bits. Ri [j] = 1 représente que Pi reçoit un message de calcul de Pj dans l'intervalle de point de reprise actuel. Ri est initialisée à 0 sauf Ri [i], qui est initialisée à 1, ce vecteur est appelé le vecteur de dépendance.

2.2.2 Description de protocole CSB

Cet algorithme est un protocole en deux phases et enregistre deux types de points de reprise sur le stockage stable.

- Dans la 1ière phase, l'initiateur envoie une demande de requête (**R_request**) à tous les processus pour demander les vecteurs de dépendance.

  - Chaque processus ayant reçu une **R_request**, il envoie son vecteur R dans un message **Vecteur_R** et bloque son exécution.

  - Après avoir reçu tous les vecteurs de dépendance, l'initiateur construit une matrice des dépendances D (N × N), et calcule les dépendances cela avec des multiplications entre son vecteur de dépendance et la matrice D, après certain itération, on a terminé si le résultat est stabilisé.

  - Après ça l'initiateur construit l'ensemble S_{forced} selon ce résultat, cet ensemble représente les processus ont une valeur 1.

  Il y a deux cas :

  - 1ière cas Pi ∉ S_{forced} (c'est-à-dire que l'initiateur ne dépend pas de Pi) :
    Envoyer un message **Continue** pour débloquer son exécution.

  - 2ième cas Pi ∈ S_{forced} :
    Envoyer un message **Request** pour lui demander de prendre un point de reprise tentative.

  - Un processus ayant reçu un message **Request** prend un point de reprise tentative et envoie un message **Reply**.

- 2ième phase :

  Si l'initiateur a reçu toutes les réponses, il envoie des messages **Commit** pour rendre les point de reprise tentatives en points de reprise permanents.

En peut résumer ces étapes dans la figure suivant :

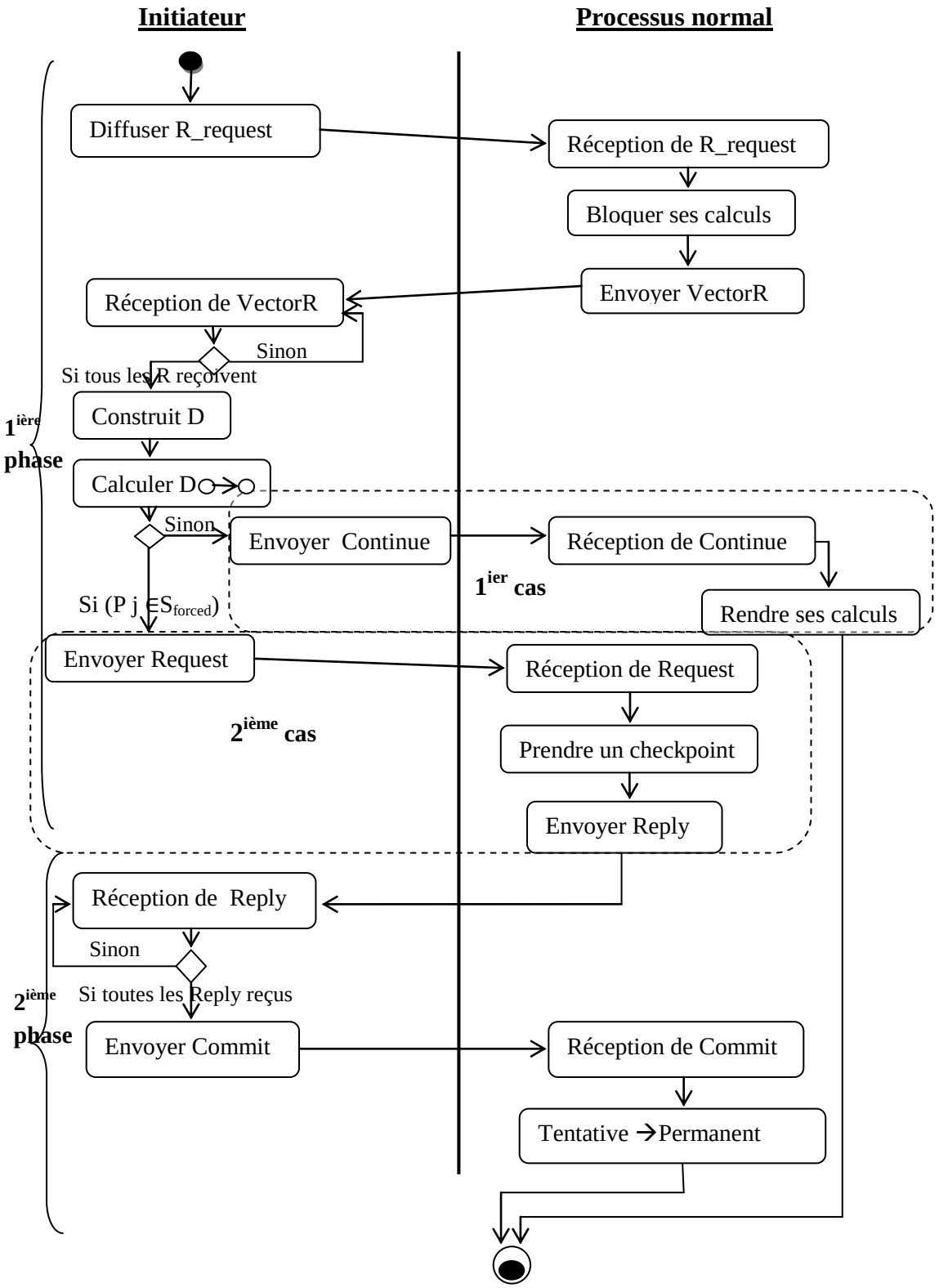


Figure 2. 1 Les étapes d'algorithme CSB

En peut détailler Calculer D dans la figure suivant :

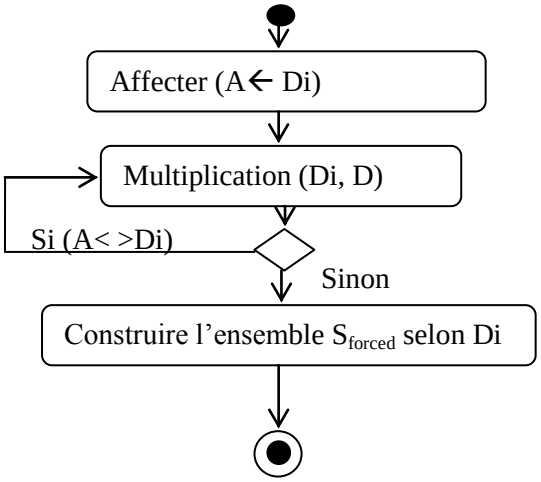


Figure 2. 2 Les étapes de Calculer D

Notez que chaque ligne de la matrice D est un vecteur de dépendance de processus P.

2.2.3 Exemple CSB

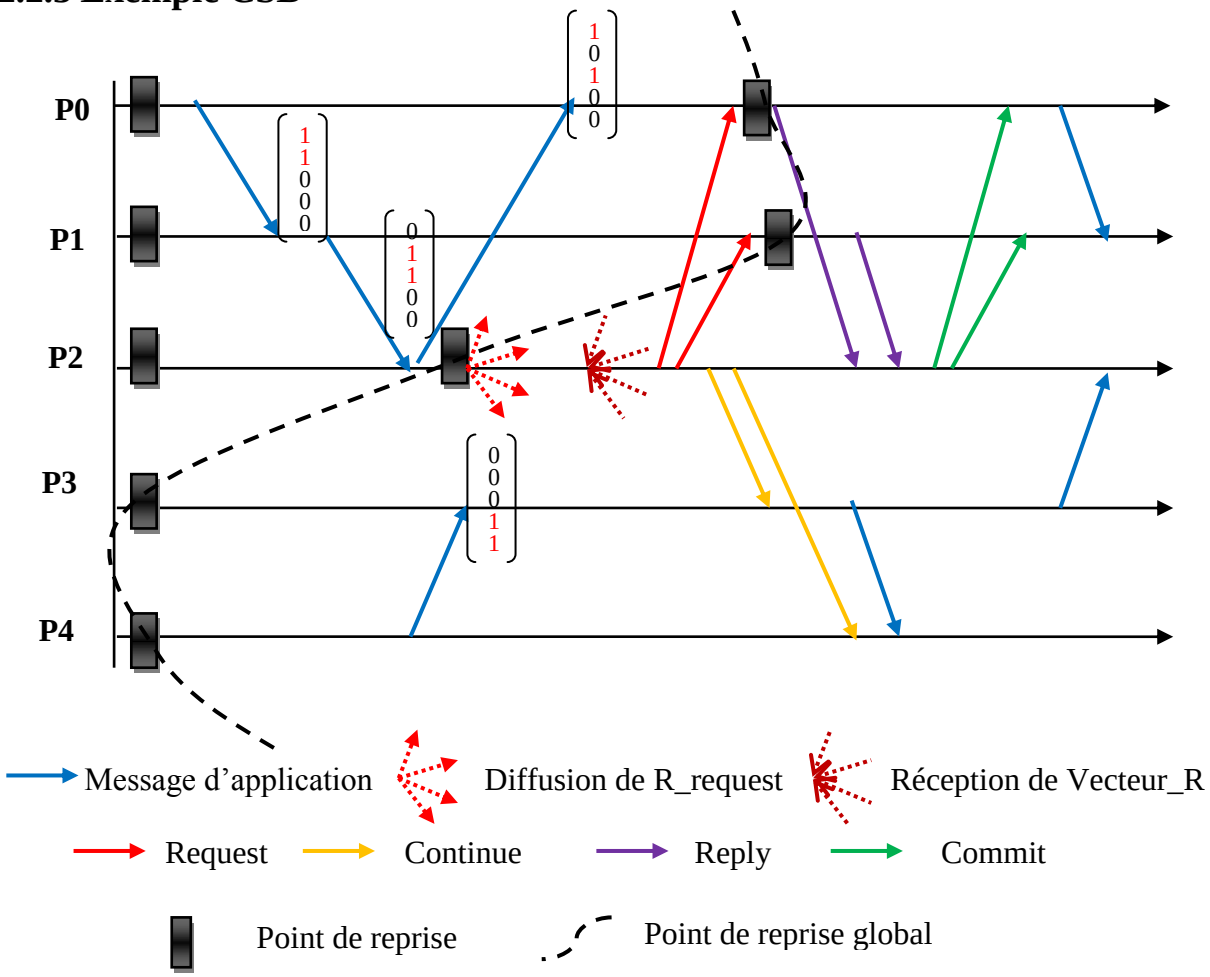


Figure 2. 3 Calcul de point de reprise global avec CSB

Dans cet exemple, P_2 souhaite calculer l'état global, il diffuse un message R_request ?

- Chaque processus ayant reçu R_request, il envoie son R vers l'initiateur P_2 via le message Vecteur_R.
- Depuis, P_2 va calculer D et construire S_{forced}.

$$\begin{bmatrix} 01100 \end{bmatrix} \times \begin{bmatrix} 10100 \\ 11000 \\ 01100 \\ 00011 \\ 00001 \end{bmatrix} = \begin{bmatrix} 11100 \end{bmatrix} \rightarrow S_{forced} = \{P0, P1, P2\}$$

- Puis, P_2 envoie Request aux P_1 et P_0 qui appartient à S_{forced} pour prendre un point de reprise tentative, et envoie continue aux autre processus qui sont P3 et P4.
- Lors de la réception de Request, P_1 et P_0 prennent des points de reprise et envoient des messages Reply.
- Après ça, P_2 peut détecter la fin de calcul d'état global et envoie une Commit vers P_1 et P_0.

2.3 Algorithme de CAO et Singhal non bloquant (CSNB)

Le protocole CSNB (l'algorithme **N**on-**B**loquant) est proposé par G.**C**ao et M.**S**inghal en 2003, cet algorithme est basé sur les points de reprise mutable, il ne bloque pas l'application pendant le calcul de point de reprise global.

2.3.1 Les messages utilisés

Dans cet algorithme, on utilise quatre types de messages :

- **Request :** Demande de prendre un point de reprise.
- **Reply :** Réponse envoyée par le processus ayant reçu un message Request.
- **Commit :** Envoyé par l'initiateur à chaque processus pour rendre les points de reprise tentative en point de reprise permanant.
- **Message :** Message d'application.

2.3.2 Les variables locales

- Ri [N] : booléen ; // Le vecteur de dépendance.

- csn_i [N] : entier ; // Les numéros de séquence initialiser à "0".
- weight: réel; // $0 \leq$ weight ≤ 1 il est utilisé pour détecter la terminaison de l'algorithme.
- trigger **:** enregistrement {
  - pid: entier ; // l'identificateur de l'initiateur.
  - inum: entier ; // la csn de l'initiateur.

  } ;
- $sent_i$: booléen ; // égal à faux, si le processus n'a envoyé aucun message.
- cp_state_i : booléen ; // égal à vrai, si Pi en cours de la phase de calcul de point de reprise.
- old_csn_i: entier; // dernier numéro de séquence.
- CPi **:** enregistrement {
  - mutable : point de reprise ;
  - Ri [N] : booléen ;
  - $trigger_i$: trigger ;
  - sent : booléen ;

  } ;

2.3.3 Description de protocole CSNB

- **envoyer un message d'application :**

  - Quand un processus Pi envoie un message d'application (Figure 2. 4), il y a deux cas :

    - S'il y a un calcul de point de reprise en cours alors P_i superpose sur le message les informations sur l'initiateur (trigger).
    - Sinon P_i envoie ce message sans ces informations (msg_trigger = NULL).

  - Ainsi, il a envoyé son csn sur ce message.

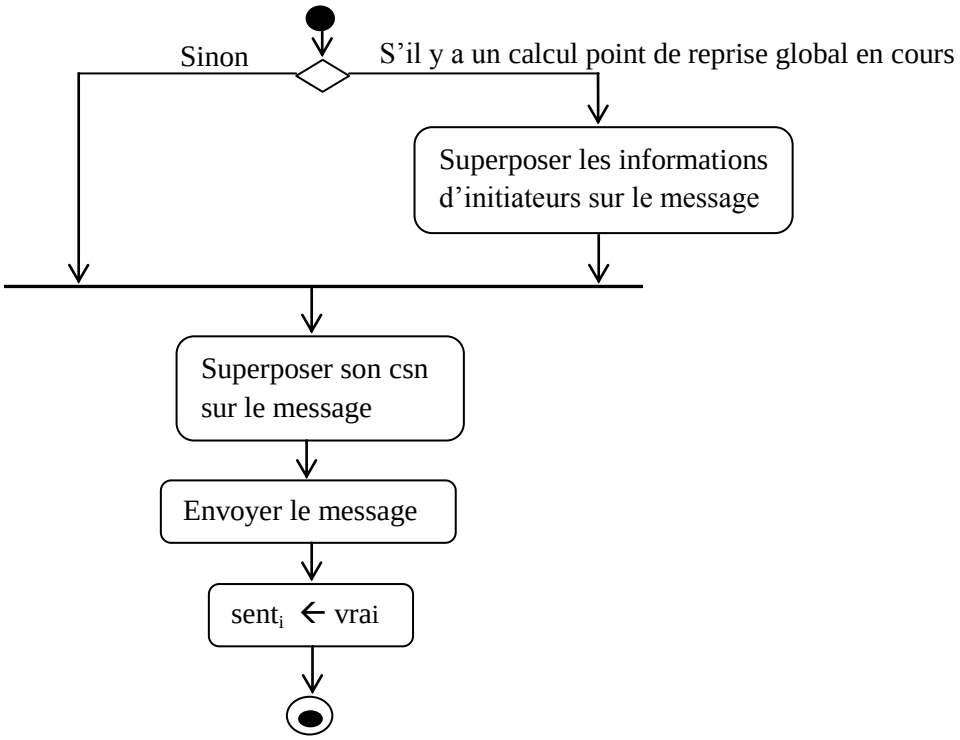


Figure 2. 4 Les étapes d'envoi des messages d'application

- **Lancement de calcul de point de reprise global :**

Quand un processus Pi souhaite lancer un calcul de point de reprise global, il prend un point de reprise tentative et envoie des requêtes (**Request**) aux processus dont il dépend (Figure 2. 5).

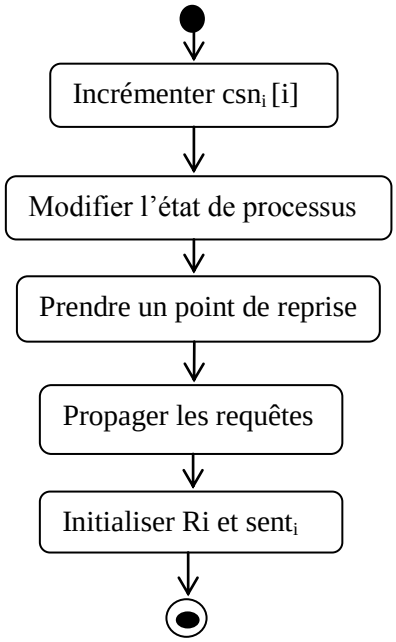


Figure 2. 5 Les étapes de lancement de calcul de point de reprise global

- **Propagation des requêtes :**

Cette procédure permet de propager les requêtes vers les processus dont il dépend (Ri[j]=1 où $i \neq j$).

Dans chaque envoie de requête, on diviser le variable weight sur deux, et superposer ce variable sur le message de requête. On a représenté ces étapes dans la figure suivant :

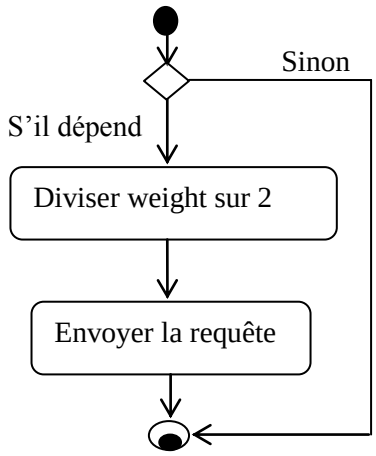


Figure 2. 6 L'envoi des requêtes

- **Réception d'une requête :**

Quand un processus Pi reçoit une requête de Pj, il compare le csn reçu dans la requête avec son ancien csn pour voir s'il a besoin propager la requête.

- Si ce n'est pas le cas, il envoie un message Reply à l'initiateur.
- Dans le cas contraire, on a plusieurs cas :

  ➢ **1ier cas** : Pi a reçu un message d'application qui porte des informations sur le calcul de point de reprise, alors Pi vérifie s'il a déjà pris un point de reprise mutable.si oui, il le rend tentative et propage la requête et envoie une réponse avec son valeur de weight.

  ➢ **2ième cas :** Pi a déjà reçu une requête, il envoie une réponse avec la valeur de weight reçu dans la requête.

  ➢ **3ième cas :** Pi n'a reçu aucune requête, alors il prend un point de reprise et propage la requête ensuite il envoie une réponse son valeur de weight.

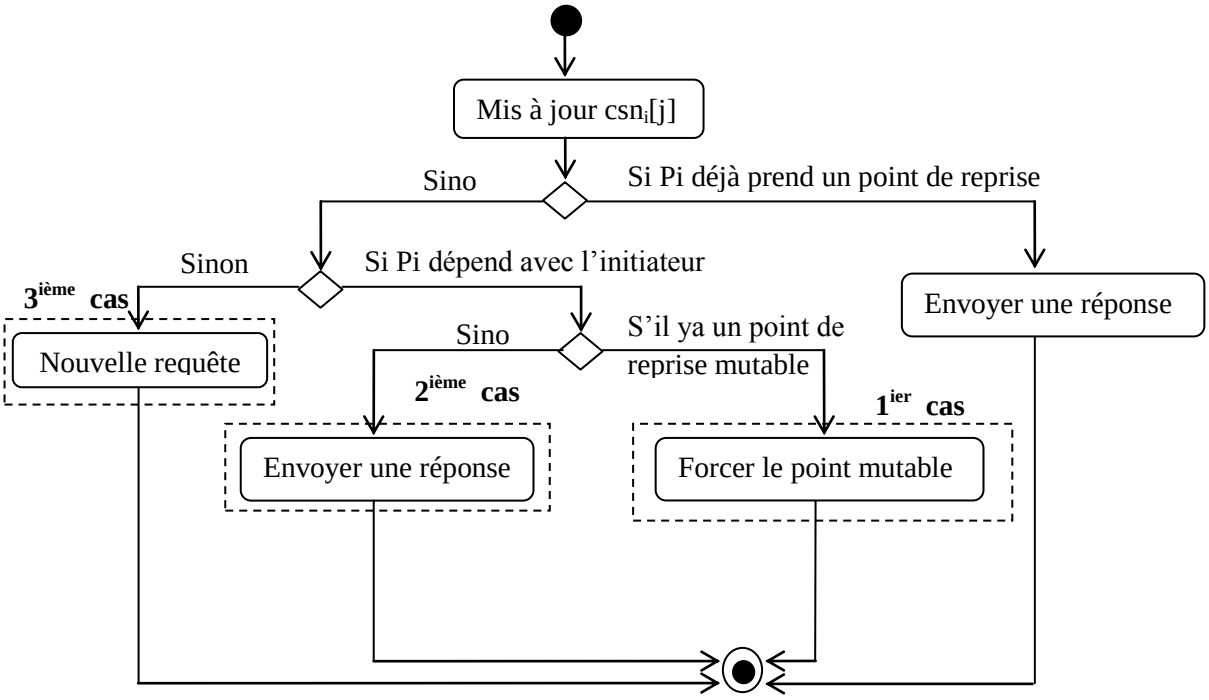


Figure 2. 7 Réception d'une requête

- **Réception des messages d'application :**

Lorsque Pi reçoit un message de calcul de Pj, Pi compare le csn du message reçu avec son $csn_i[j]$:

- **1ier cas :** Si csn_reçu < csn_i [j] alors Pi mis à jour $R_i[j]$ et consomme le message.
- **2ième cas :** Sinon, cela implique que Pj a pris un point de reprise avant d'envoyer m.

  Dans ce cas, Pi teste l'information portée par le message concernant l'initiateur.

  - Si Pi a déjà reçu cette information alors Pi mis à jour Ri et consomme le message.
  - Sinon, mettre à jour $csn_i[j]$ à recu_csn, et il vérifie les trois conditions possible.
    - ✓ **Condition 1 :** Pj a envoyé le message m après le processus de calcul d'état global.
    - ✓ **Condition 2 :** Pi a envoyé un message depuis le dernier point de reprise.
    - ✓ C**ondition 3 :** Pi n'a pas pris un point de reprise pour cet initiateur.

P_i teste les trois conditions précédentes :

- Si toutes ces conditions sont satisfaites, Pi prend un point de reprise mutable.
- Si la 1ière conditions est satisfaite, Pi incrémente $csn_i[i]$ change son état, et consomme le message.

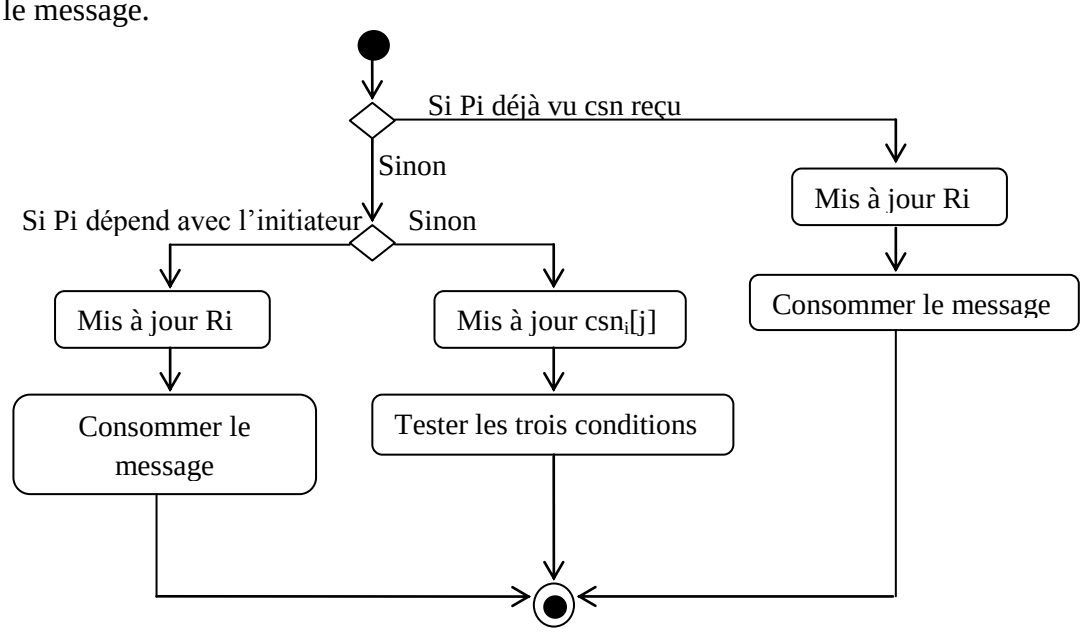


Figure 2. 8 Réception d'un message

- **Réception de messages Reply et vérification de terminaison :**

L'initiateur calcule la somme des valeurs weight reçus dans toutes les réponses. Lorsque weight égale à 1, il détecter la fin de la 1ière phase, et il diffuse des messages Commit.

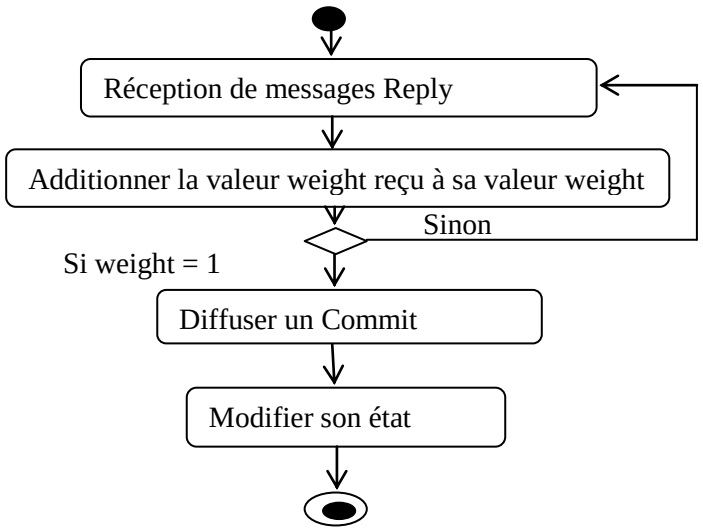


Figure 2. 9 Réception d'un message Reply

- **Réception d'un Commit :**

A la réception de Commit, tous processus qui a pris un point de reprise tentative le rendre permanant, et les point de reprise mutable seront éliminés.

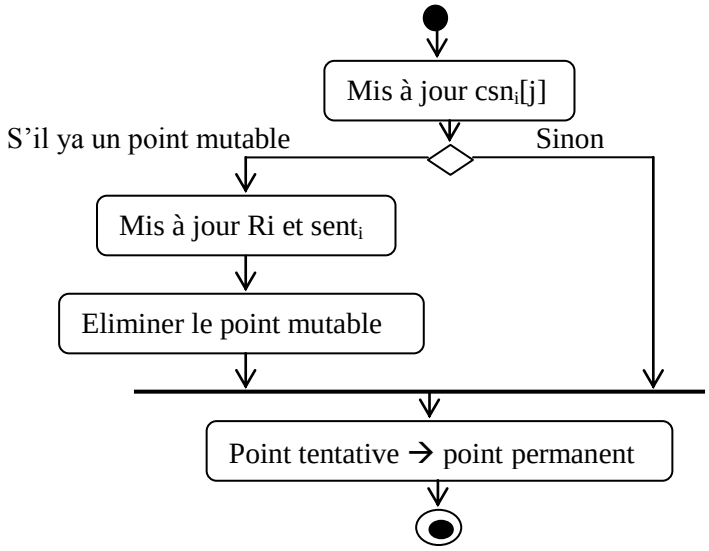


Figure 2. 10 Réception d'un Commit

2.3.4 Exemple CSNB

Dans cet exemple, P_2 lance le calcul d'état global et envoie les requêtes aux processus (P_1, P_3). A leurs tours, P_1 et P3 propagent aussi la requête vers (P_0, P_4).

- Si le message m9 est reçu par P_4 avant la requête. Après, P_4 rend le point de reprise mutable en tentative dés la réception de la requête, et aussi il va propager la requête vers P_5.

- Chaque processus ayant reçu une requête prend un point de reprise et il envoie un message Reply.

- Si P_2 a reçu tous les messages Reply de tous les processus, il diffuse les messages Commit.

On remarque que pour un exemple simple, il y a des requêtes dupliquées, par exemple P_4 a reçu une requête de P_3 et P_0 alors il va envoyer deux messages Réponse à l'initiateur, d'où un très grand nombre de message de synchronisation.

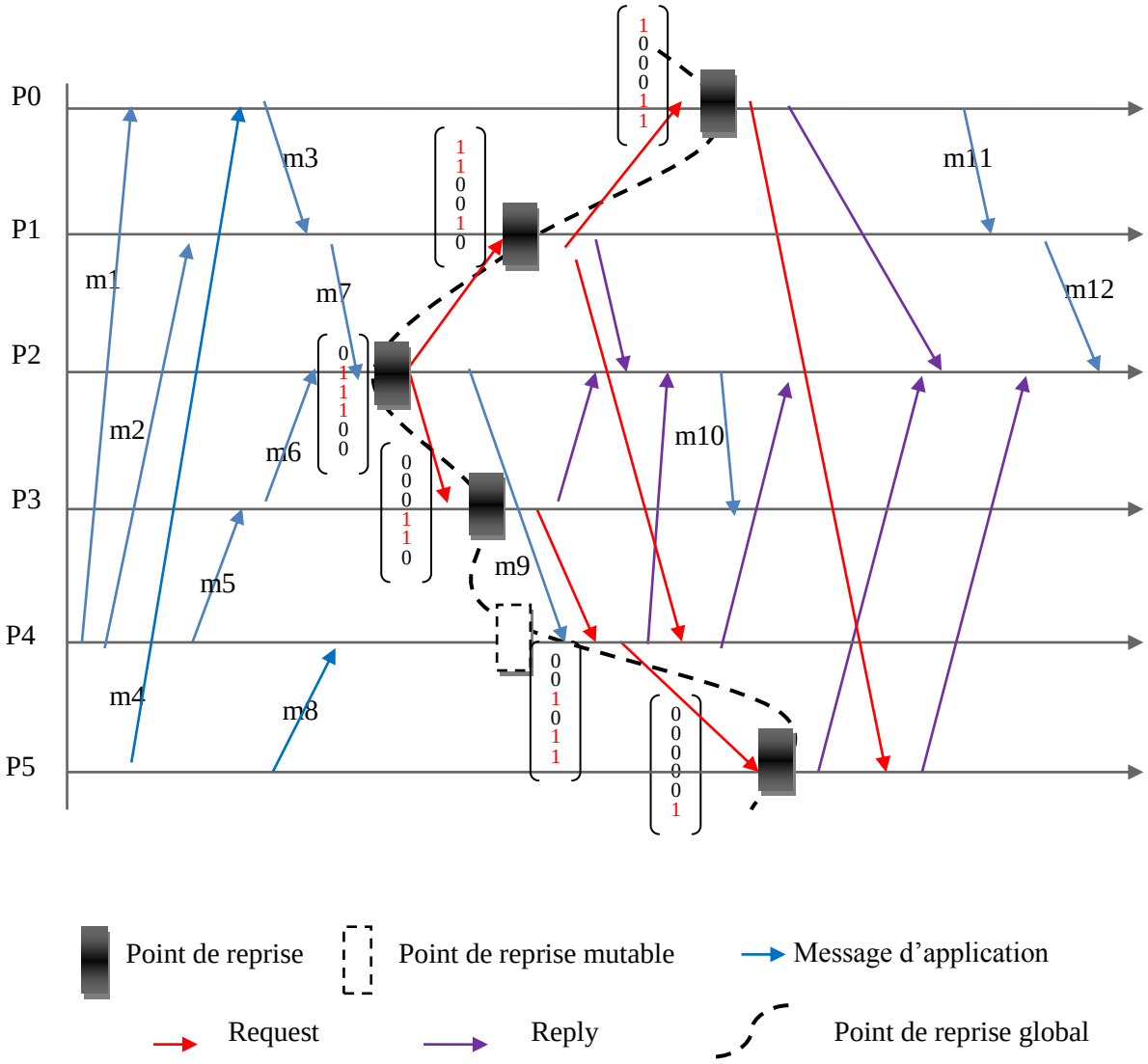


Figure 2. 9 Calcul de point de reprise global avec CSNB

2.3.5 La 1ière amélioration AM1

L'objectif de cette amélioration est de minimiser le nombre de requêtes et la durée de la première phase, et baser sur maximiser l'utilisation des messages d'application dans le calcule de point de reprise global ;

2.3.5.1 Description de protocole AM1 :

Comme le vecteur Ri contient les dépendances directes d'un processus Pi, on profite de ce vecteur pour calculer les dépendances indirectes, pour cela ce vecteur sera envoyé avec les messages d'application soit avant ou pendant la période de calcul de point de reprise global.

Lors de la réception du message d'application par P_i, P_i met à jour son Ri avec le max entre Ri et le vecteur R reçu dans le message.

2.3.5.2 Exemple AM1

Les vecteur de dépendance R sont envoyés avec les messages par exemple le vecteur R de P_4 est [000011] et le vecteur P_2 est [111111].

Dans cet exemple P_2 dépend de tous les processus alors si P_2 lance le calcul d'état global, il envoie des requête vers tous les processus dans ce cas il n'ya pas des requête dupliquée.

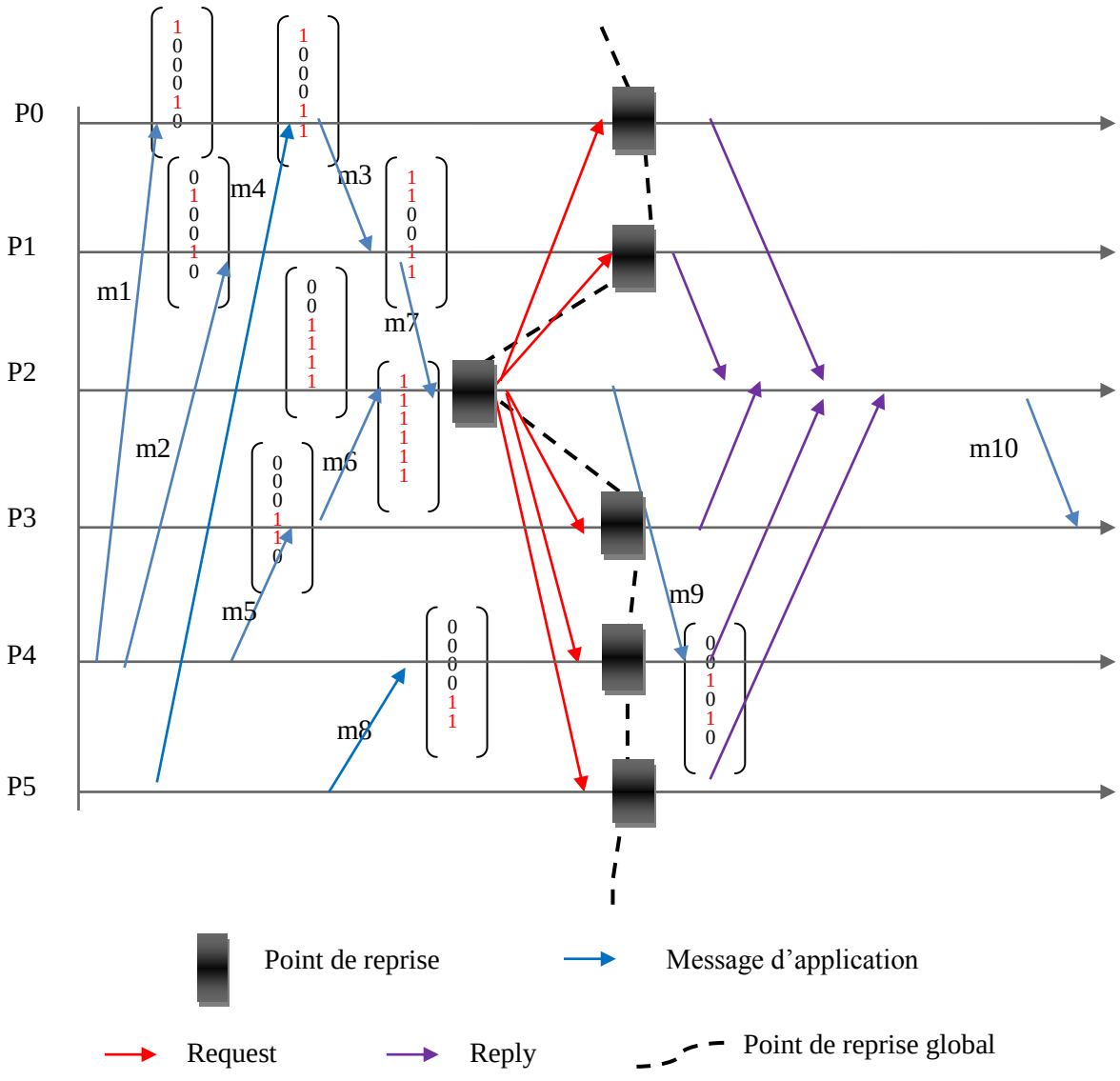


Figure 2. 10 Calcul de point de reprise global avec AM1

2.3.6 La 2ième amélioration AM2 :

L'objectif de la deuxième amélioration est de minimiser le nombre de requêtes.

2.3.6.1 Description de la 2ième amélioration AM2 :

Dans cette amélioration, on essaye de profiter les messages de réponse (Reply), pour cela on va superposer les vecteurs de dépendance sur ces messages. Dans ce cas l'initiateur s'occupe totalement l'envoie de requêtes.

2.3.6.2 Exemple AM2

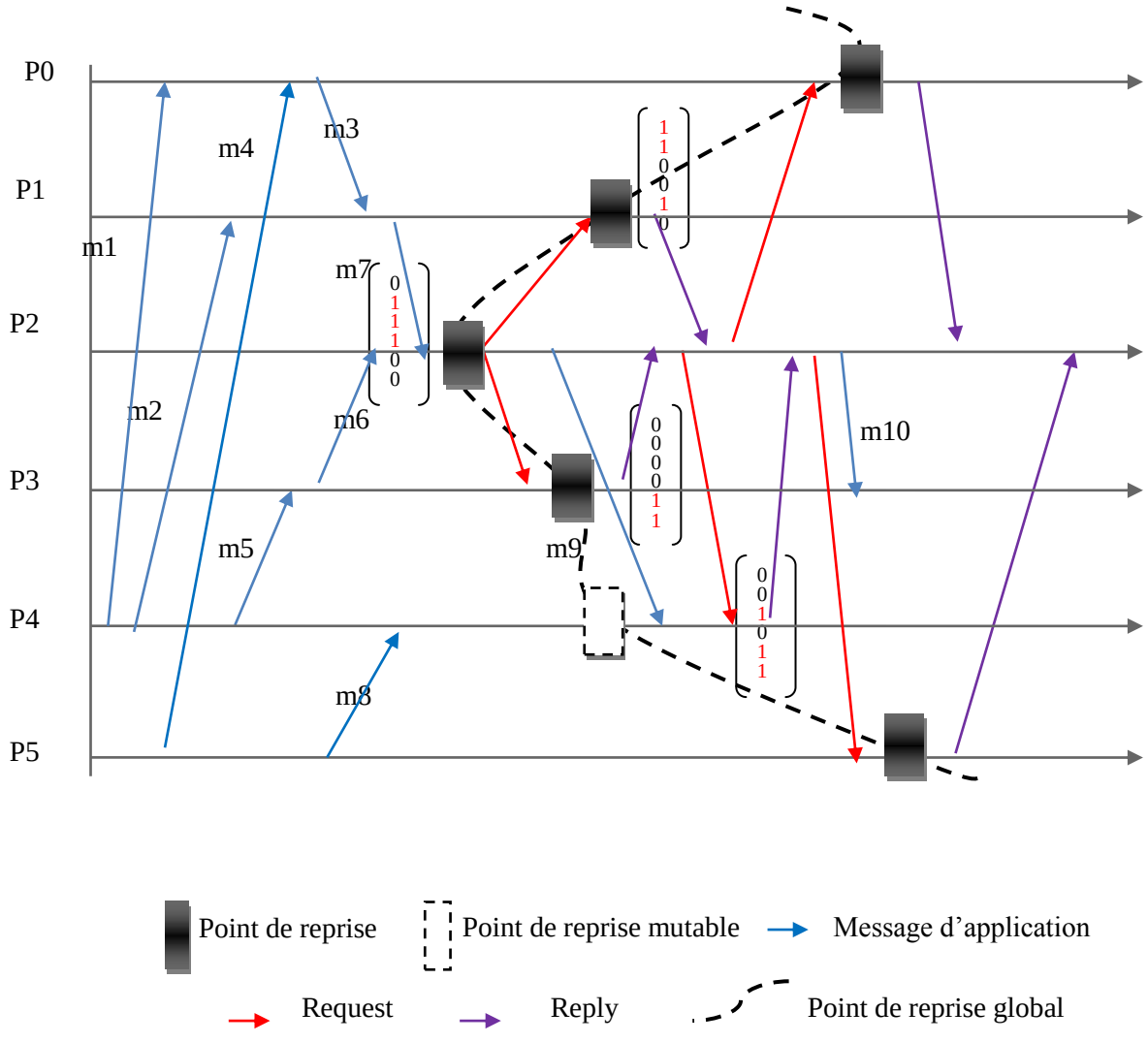


Figure 2. 11 Calcul de point de reprise global avec AM2

Dans cet exemple, lors de calcul d'état global, P_2 a envoyé des requêtes vers P_1 et P_3, puis, il a reçu les dépendances de P_1 et P_3 via des messages Reply. Après ça, P_2 a envoyé les requêtes vers P_0, P_4 et P_5.

Par conséquent, avec l'algorithme AM2, on peut remarquer clairement l'absence des requêtes et réponses (message Reply) dupliqués, c'est-à-dire, moins des messages de synchronisation dans AM2 par rapport CSNB.

2.3.7 La 3^{iéme} amélioration AM3 :

Pour avoir une meilleure performance la 3^{ième} amélioration combine les deux améliorations précédentes.

2.3.7.1 Exemple AM3

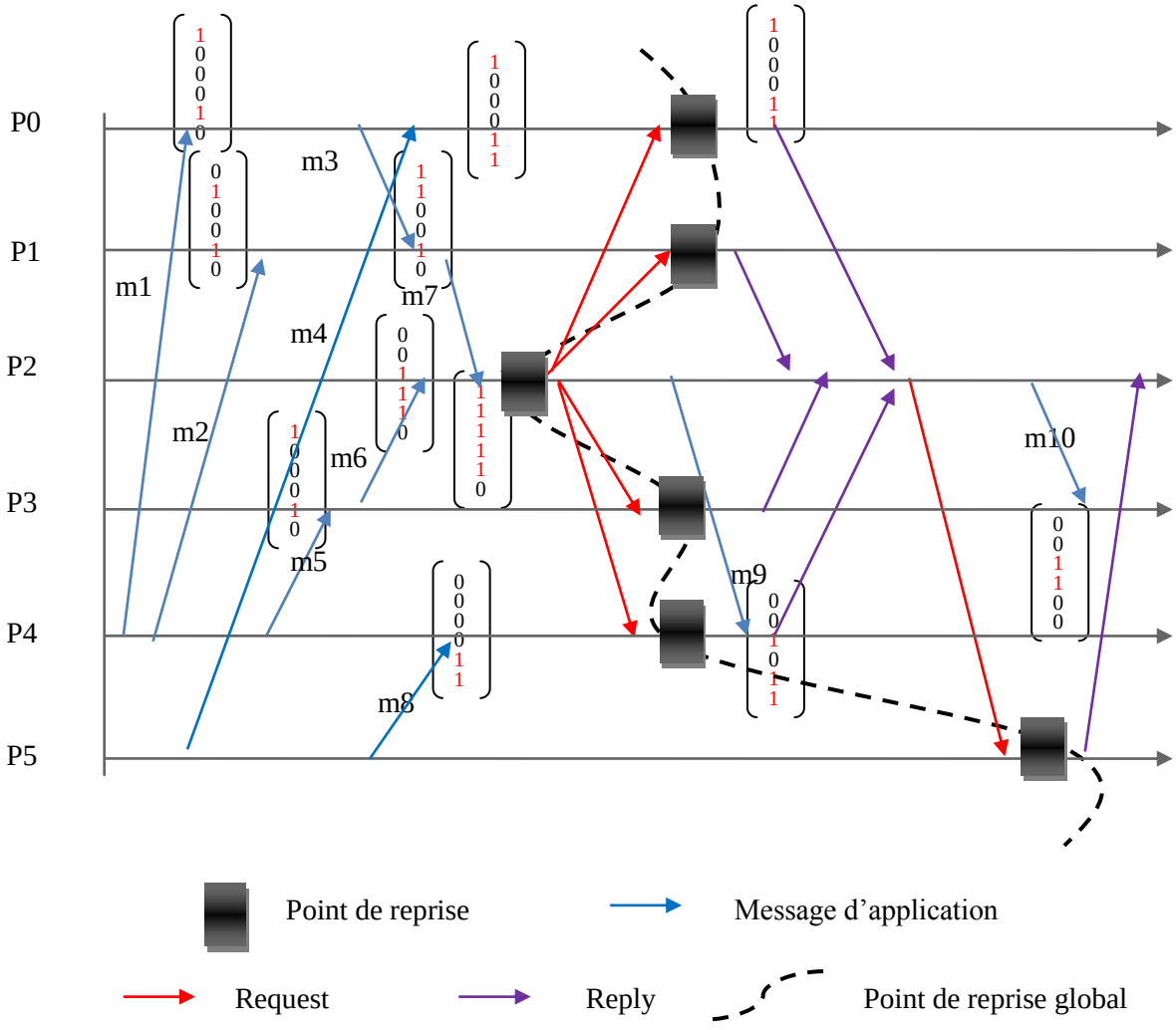


Figure 2. 12 Calcul de point de reprise global avec AM3

Dans cet exemple, si on utilise la 1ière amélioration le processus P_5 peut recevoir deux requêtes dupliquées de P_4 et P_0, cet inconvénient est éliminée par la 2ième amélioration, malgré que la 2ième amélioration aura une durée de la 1ière phase très grande, c'est pour cela on a combiné les deux améliorations.

2.4 Conclusion

Dans ce chapitre, nous avons présenté les algorithmes (CSB, CSNB), ainsi que les améliorations de CSNB qui sont AM1, AM2, AM3. Nous avons détaillé ces algorithmes et faire des exemples pour ces algorithmes.

Evaluation des performances 3

***L**'objectif de ce chapitre est de simuler les algorithmes (protocole CSB, protocole CSNB et ses améliorations) avec le simulateur NS-2, afin de faire une étude comparative pour évaluer la performance des algorithmes.*

3.1 Introduction

La simulation permet de tester à moindre coût les nouveaux protocoles et d'anticiper les problèmes qui pourront se poser dans le futur afin d'implémenter la technologie la mieux adaptée aux besoins.

Dans ce présent chapitre nous allons essayer de réaliser et analyser la simulation pour évaluer les performances des protocoles présentés dans le chapitre 2.

Lors de ces études on va interpréter également les courbes obtenues à partir des scénarios proposés pour chaque protocole.

3.2 L'outil de simulation

Nous avons utilisé NS-2 (Network Simulator 2) comme outil de simulation, cet outil est un logiciel de simulation de réseaux informatiques, il est principalement bâti avec les idées de la conception par objets, de réutilisabilité du code et de modularité, il est devenu aujourd'hui un standard de référence en ce domaine. [2]

Ce logiciel est open source. Son utilisation est gratuite et exécutable tant sous Unix et Windows.

Nous avons réalisé la simulation sous Linux Mandriva 2010, en utilisant la version ns-allinone-2.34, et on a utilisé un ordinateur dont la configuration suivante:

- Processeur : Pentium(R) Dual-Core CPU T4400 @ 2.20GHz.
- Mémoire : 3 Go.
- Disque dur : 250 GB /Go.

3.3 Réalisation de simulation

Afin de réaliser notre simulation des algorithmes (CSB, CSNB, AM1, AM2, AM3), nous avons intégré ces protocoles dans le simulateur NS-2.

Pour intégrer ces algorithmes, nous avons besoin de la création de deux fichiers sources écrits en langage c++ (*.h, *.cc):

- ✓ le fichier *.h est utilisé pour la déclaration des variables et contient la structure des messages utilisés dans chaque protocole.
- ✓ le fichier *.cc est utilisé pour définir l'algorithme qui contient ces procédures.

Les étapes d'intégration sont les suivantes :

- Modifier le fichier makefile, pour cela nous avons ajouté à la variable OBJ_CC : (diffusion3/apps/agents/agentCSNB/CSNB.o \).
- Modifier le fichier packet.h pour connaître notre agent (dans le répertoire common, ajouter : (*static const packet_t PT_CSNB = 62;)* à (*typedef unsigned int packet_t* ;) et (name_[PT_CSNB]="CSNB" ;) à (class p_info).
- Modifier le fichier ns-default.tcl (dans le répertoire tcl/lib, ajouter : Agent/CSNB set packetSize_ 64 et à la suite, initialiser les variables utilisées dans le programme).
- À la fin de ces modifications, il faut recompiler NS-2 par la commande *make*. Un fichier de type objet (*.o) sera généré après la recompilation de NS-2.

  On peut résumer ces étapes dans la figure suivante :

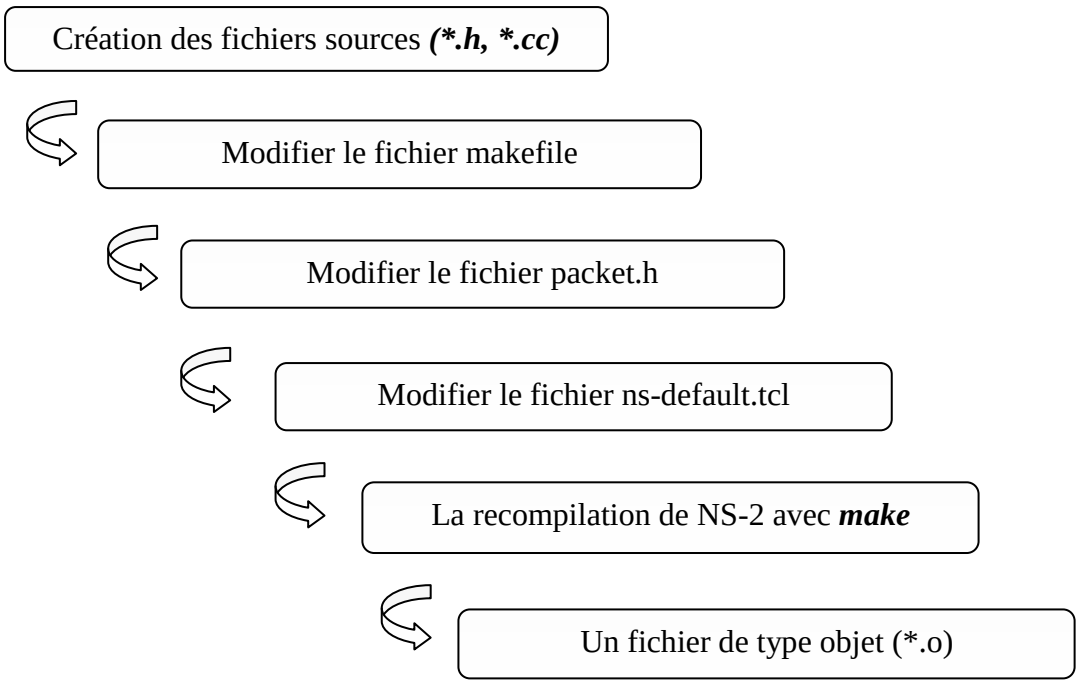


Figure 3.1 Les étapes de la création d'un protocole

3.4 Les étapes de simulation

Après l'ouverture du terminal et celle du répertoire où se trouvent les fichiers contenant le programme à exécuter :

- Le fichier (*scenario.tcl*) permet à l'utilisateur de saisir les informations nécessaires pour la simulation (le nombre de processus, le nombre de messages, le nombre d'initiateur…) ces informations sont enregistrées dans un fichier appelé *scenario.txt*.
- Le fichier *messages.txt* est devenu aussi comme résultat de premier fichier, contient les informations suivantes (le processus source, le processus destination, et le moment d'envoie).

Ces fichiers (*.txt) sont nécessaires pour l'exécution des autres fichiers (CSNB.tcl, *CSB.tcl …etc.)*, ces derniers fichiers permettent de:

- Lancer le Nam où on peut animer la simulation. Le lancement de simulation permet de voir les nœuds, les liens entre les nœuds, l'envoi et la réception des messages,…etc.
- Obtenir les résultats de la simulation pour tracer des courbes afin d'analyser la performance de l'algorithme simulé. [10]

On peut résumer ces étapes dans la figure suivant :

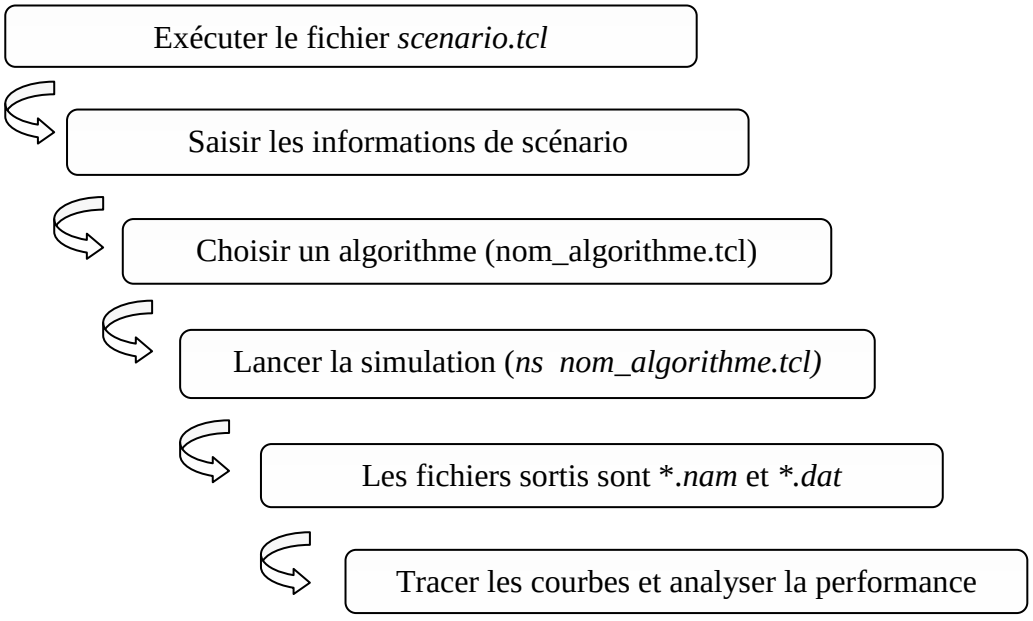


Figure 3.2 Les étapes de simulation

3.5 Les paramètres de simulation

3.5.1 Paramètres de contexte de simulation

Avant de lancer la simulation des scénarios, nous devons ajuster et fixer certains paramètres qui vont constituer le contexte de notre simulation. Ces paramètres sont illustrés dans le tableau suivant :

Les paramètres	**Descriptions**
Nbr_proc	Nombre de processus
Nbr_intr	Nombre d'initiateur
Nbr_msg	Nombre de message d'application

Tableau 3. 1 Les paramètres de simulation

Dans notre simulation, il ya trois scénarios :

Scénario1: Variation du nombre d'initiateur (Nbr_intr)

Dans ce scénario Nous avons fixé le nombre de processus (Nbr_proc) à 64 et le nombre de message (Nbr_msg) à 5000 et nous avons varié le nombre d'initiateur (Nbr_intr) de 1 à 64.

Scénario2 : Variation du nombre de processus (Nbr_proc)

Dans ce scénario Nous avons fixé le nombre de message (Nbr_msg) à 5000 et le nombre d'initiateur (Nbr_intr) à 16 et nous avons varié le nombre de processus (Nbr_proc) de 4 à 64.

Scénario3 : Variation du nombre de message (Nbr_msg)

Dans ce scénario Nous avons fixé le nombre de processus (Nbr_proc) à 64 et le nombre de initiateur (Nbr_intr) à 16 et nous avons varié le nombre de message (Nbr_msg) de 300 à 4800;

Le tableau suivant résume les différents paramètres utilisés dans la simulation :

	Nbr_proc	Nbr_intr	Nbr_msg
Scénario1	64	1<Nbr_intr<64	5000
Scénario2	4<Nbr_proc<64	16	5000
Scénario3	64	16	300<Nbr_msg<4800

Tableau 3. 2 Variation des paramètres de simulation

3.3.2 Paramètres à évaluer

Les scénarios que nous avons réalisés, nous ont permis de mettre on évidence des paramètres importants qui vont faire la différence dans l'évaluation et la comparaison des algorithmes simulés et qui sont :

- Nombre de requêtes (**Nbr_req**) :

Le nombre de requêtes est un paramètre très important dans la comparaison des performances.

$$Nbr\_req = \sum Nbr\_req_i \quad \text{avec } Nbr\_req_i \text{ est le nombre de requête de processus Pi}$$

- La durée moyenne de la première phase (**DMP**) :

$$DMP = \frac{\sum DP}{Nbr\_intr}$$ avec DP : La durée de la première phase pour un seul initiateur

- Le nombre de points de reprise de mutable (**Nbr_M**):

Dans les algorithmes non bloquants ce paramètre est important, ce type des points est utilisé pour éviter les messages orphelins.

$$Nbr\_M = \sum Nbr\_M_i$$

- Le nombre de message de synchronisation (**Nbr_msg_syn**): on peut définier ce paramètre comme élément clé pour évaluer les performances des algorithmes.

Nbr_msg_syn = Nbr_req + Nbr_Commit + Nbr_Reply + Nbr_Continu + Nbr_VectorR (pour les algorithmes bloquants)

Nbr_msg_syn = Nbr_req + Nbr_Commit + Nbr_Reply (pour les algorithmes non bloquants)

Ces paramètres sont illustrés dans le tableau suivant :

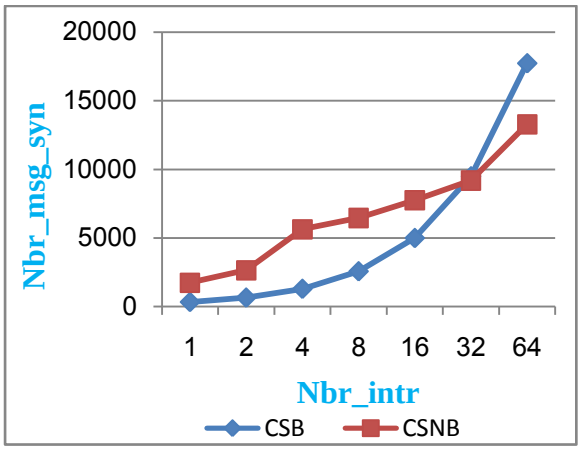
Les paramètres	Descriptions	Expression
Nbr_req	Nombre de requêtes	$\sum Nbr\_req_i$
DMP	La durée de première phase	$\sum DPi / Nbr\_intr$
Nbr_M	Nombre de points de reprise mutable	$\sum Nbr\_M_i$
Nbr_msg_syn	Nombre de messages de synchronisation	La somme total des messages de contrôle

Tableau 3. 2 Les paramètres d'évaluation

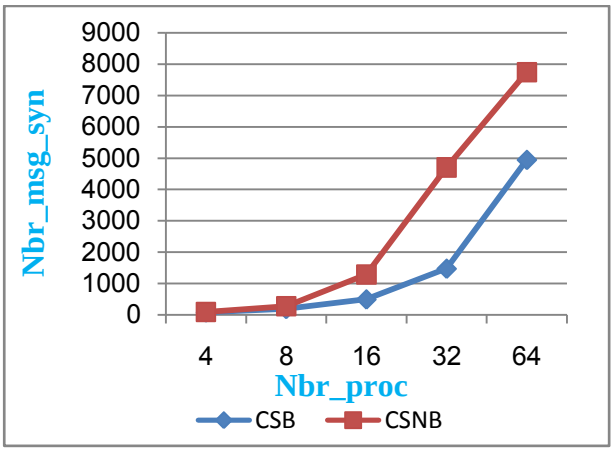
3.6 Résultats et interprétation

3.6.1 Comparaison entre CSB et CSNB

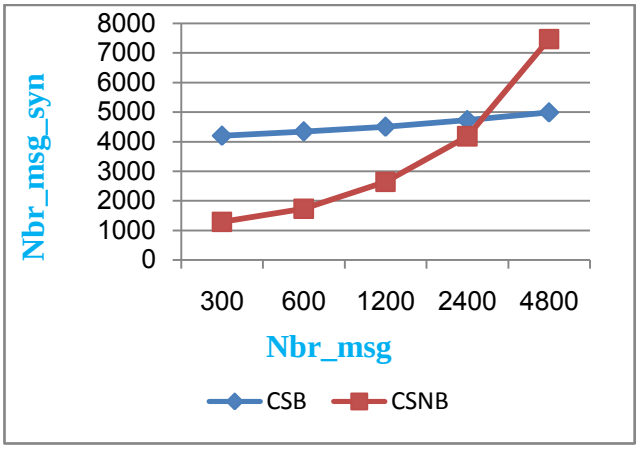
Après la simulation des deux algorithmes *CSB* et *CSNB*, on a obtenu les résultats ci-dessous :



a) *Scénario1*



b) *Scénario2*



c) *Scénario3*

Figure 3.3 Comparaison entre *CSB* et *CSNB*

Dans cette figure, on remarque que le nombre des messages de synchronisation augmente d'une manière progressive dû à l'augmentation du nombre de processus, le nombre d'initiateur et le nombre de message d'application.

Par exemple, dans le scénario1, le Nbr_msg_syn pour 2 initiateurs est inferieur à 5000, et pour 64 initiateurs le Nbr_msg_syn dépasse 10000 messages de synchronisation.

Ainsi, à partir de 1 jusqu'à 16 initiateurs l'algorithme *CSNB* a un grand Nbr_msg_syn par rapport au *CSB*, ce qui est expliqué par le faite que les informations de dépendances est très grand (5000 messages d'application).

Mais, si le nombre d'initiateurs est augmenté on remarque l'inverse. Ceci est expliqué par la diffusion de messages de contrôle pour chaque initiateur dans l'algorithme *CSB*.

Même remarque pour scénario2, le Nbr_msg_syn est élevé dans *CSNB* que dans *CSB*, par exemple, pour 32 processus le Nbr_msg_syn est inferieur à 2000 dans *CSB*, et pour *CSNB* le Nbr_msg_syn dépasse 4500 messages de synchronisation.

Dans scénario 3, on a obtenu des résultats différents, à partir de 300 jusqu'à 2400 messages l'algorithme *CSNB* a un petit Nbr_msg_syn par rapport *CSB*. Ceci expliqué par le principe de *CSNB* que les requêtes sont envoyées sauf vers les processus qui ont une dépendance (direct ou indirect) avec l'initiateur et il y a un seul message qui diffuser (Commit).

D'autre part, on remarque que Nbr_msg_syn dans *CSB* est presque stable. Ceci est expliqué par le faite que le Nbr_msg_syn est presque dépendant avec le nombre d'initiateurs et nombre

de processus (les messages diffusé). Si le nombre d'initiateurs est augmenté on remarque l'inverse. Ceci expliqué par le faite que les informations de dépendances est très grand.

On peut observer clairement le problème de messages de synchronisation pour *CSNB* dans les grands systèmes ayant grand nombre de processus, et dans les systèmes ayant un échange énorme d'information.

Malgré ce problème, les algorithmes non bloquants reste la meilleur par rapport aux les algorithmes bloquants, car les algorithmes bloquant peuvent dégrader les performances du système à cause de blocage de leurs applications.

3.6.2 Comparaison entre CSNB et les améliorations proposées

3.6.2.1 Nombre de requête

Dans ce scénario nous avons varié le nombre d'initiateur pour étudier leur effet sur le nombre de requêtes :

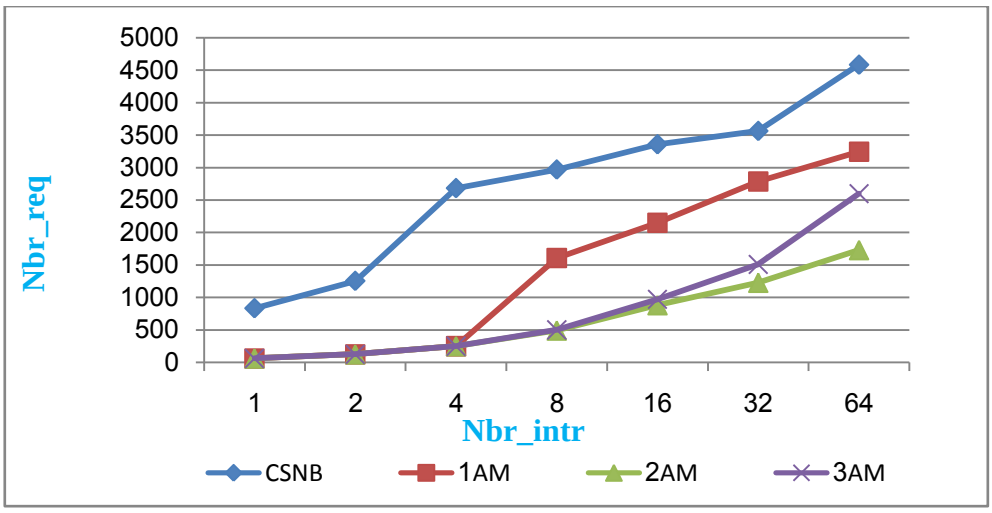


Figure 3.4 Influence de nombre d'initiateur sur le Nbr_req

	1	2	4	8	16	32	64
CSNB	836	1257	2686	2972	3360	3566	4586
AM1	63	126	252	1608	2151	2786	3244
AM2	63	126	252	493	883	1228	1730
AM3	63	126	252	504	974	1512	2601

Tableau 3. 3 Les valeurs de Nbr_req correspondent à scénario1

Premièrement, on remarque que le nombre des requêtes augmente d'une manière progressive dû à l'augmentation du nombre d'initiateurs.

Pour seulement 4 initiateurs le nombre de requêtes pour les trois améliorations est inferieur à 500 mais pour l'algorithme *CSNB* le nombre de requêtes dépasse 2500 requêtes.

Dans la figure 3.4, il est clair que le nombre de requêtes a été diminué dans les trois améliorations par rapport à l'algorithme original, car le but de l'amélioration était de réduire le nombre des requêtes dupliqués. Cela implique que le nombre de message Reply a été diminué.

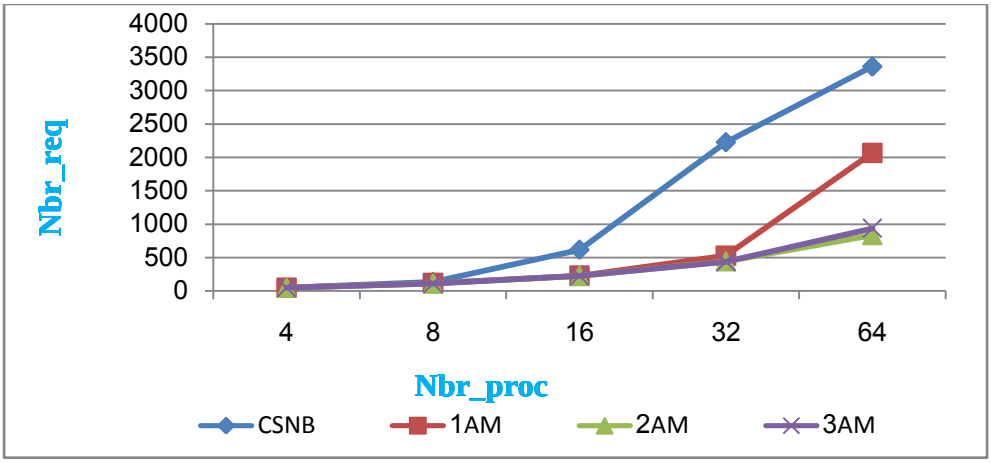


Figure 3.5 Influence de nombre de processus sur le Nbr_req

	4	8	16	32	64
CSNB	48	132	616	2227	3361
AM1	48	117	227	527	2066
AM2	48	112	225	444	836
AM3	48	112	225	437	937

Tableau 3. 4 Les valeurs de Nbr_req correspondent à scénario2

Dans le 2^{éme} scénario, avec l'augmentation de processus, on remarque l'augmentation de nombre de requêtes.

Les deux améliorations *AM2* et *AM3* ont pratiquement la même performance, on remarque aussi que pour 64 processus le nombre de requêtes pour ces deux améliorations reste inferieur à 1000 requêtes.

Mais pour la 1ère amélioration, on remarque l'augmentation énorme de nombre de requêtes qui atteint 2000 requêtes. Ceci est expliqué par le faite que les informations de dépendances est très peux quand le nombre de processus est grand.

Dans *scénario3* on a obtenu les résultats ci-dessous :

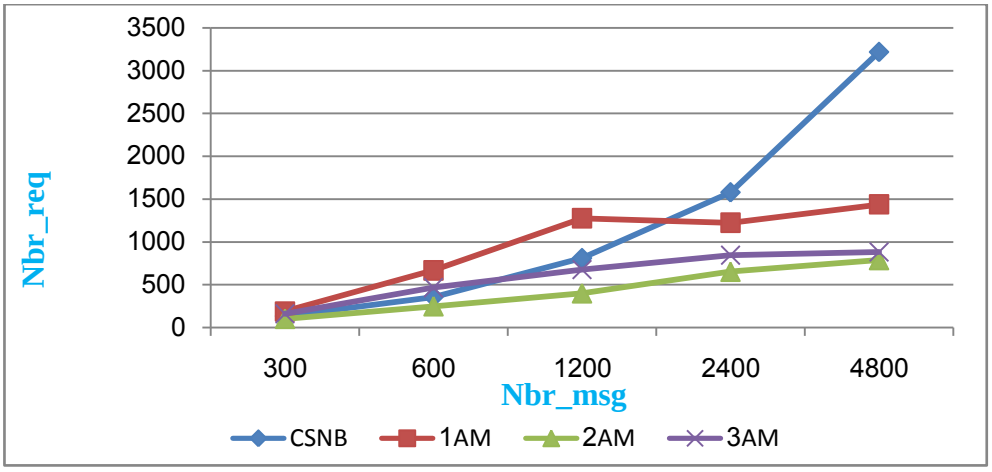


Figure 3.6 Influence de nombre de message sur le Nbr_req

	300	600	1200	2400	4800
CSNB	134	356	810	1579	3219
AM1	188	667	1276	1224	1437
AM2	101	248	399	651	787
AM3	159	464	676	845	881

Tableau 3. 5 Les valeurs de Nbr_req correspondent à scénario3

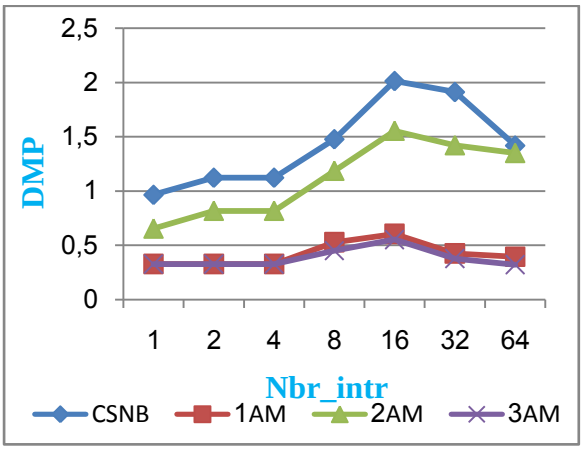
Dans ce scenario, on a varie le nombre de message pour étudier son influence sur le nombre requêtes, nous remarquons la croissance de nombre de requêtes avec l'augmentation du nombre de messages, à partir de 300 jusqu'à 1200 message l'algorithme *AM1* a un grand nombre de requêtes. Ceci expliqué par le faite qu'il y a très peu d'informations de dépendances sont envoyées via les messages.

Les deux améliorations *AM2*, *AM3* ont des meilleur performances par rapport a *CSNB*, et la différence atteint jusqu'à 2000 requêtes et alors que la différence dans les messages de synchronisation atteint jusqu'à 4000 (chaque processus a été reçu une requête, il envoi un message Reply).

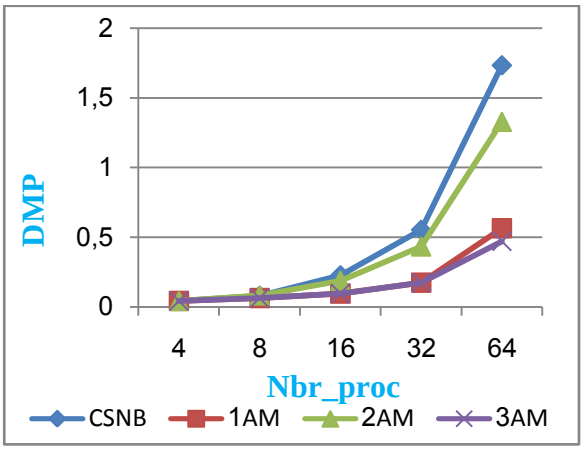
Les algorithmes *AM1*et *CSNB* dégradent ses performances dans les grands systèmes ayant grand nombre de processus, et dans les systèmes ayant un échange énorme d'information.

3.6.2.2 La durée moyenne de la première phase

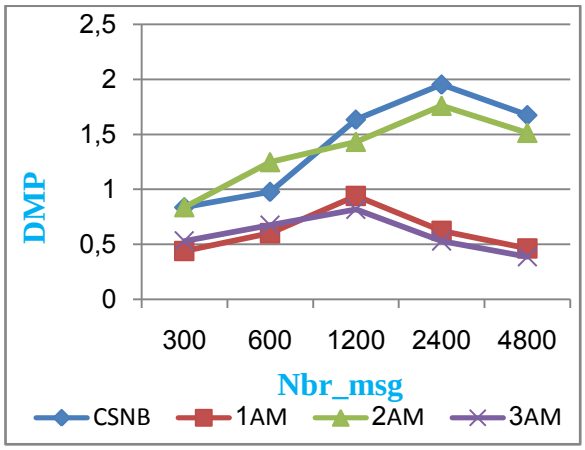
On a obtenu les résultats suivant :



a) ***Scénario1***



b) Scénario2



c) *Scénario3*

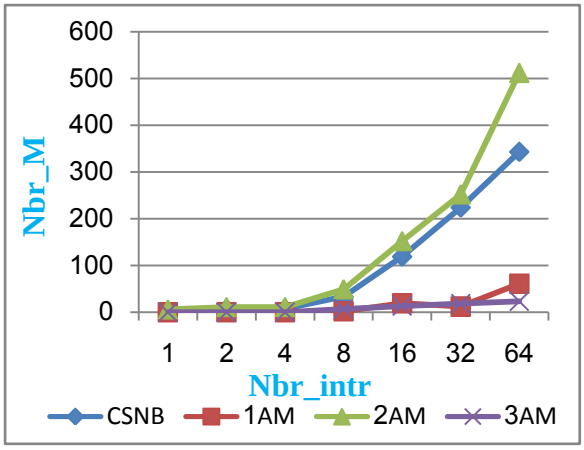
Figure 3.8 L'évaluation de DMP

On remarque que la durée moyenne de la premiere phase est presque stable dans les courbes de les scénarios 1 et 3 , mais il augmente d'une manière progressive dû à l'augmentation du nombre de processus dans le scénario2.

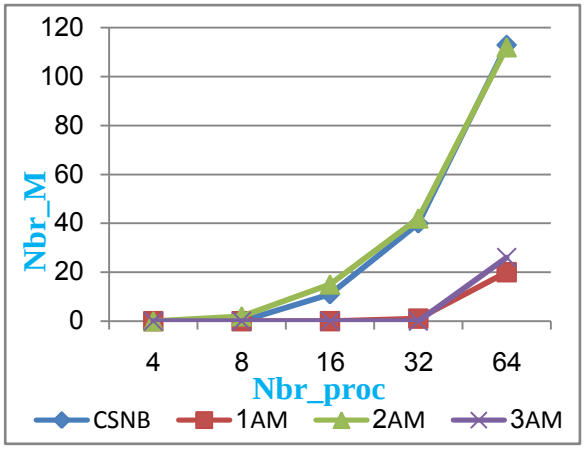
Généralment, dans les courbes de cette figure, la durée moyenne de la premiere phase pour les algorithmes *AM3* et *AM1* est nettement inférieur par rapport aux autres. Ceci est expliqué par l'envoi des information de dépandence avec les messages d'application et le mécanisme utilisé pour éviter les requêtes dupliquées.

3.6.2.3 Nombre de points de reprise mutable

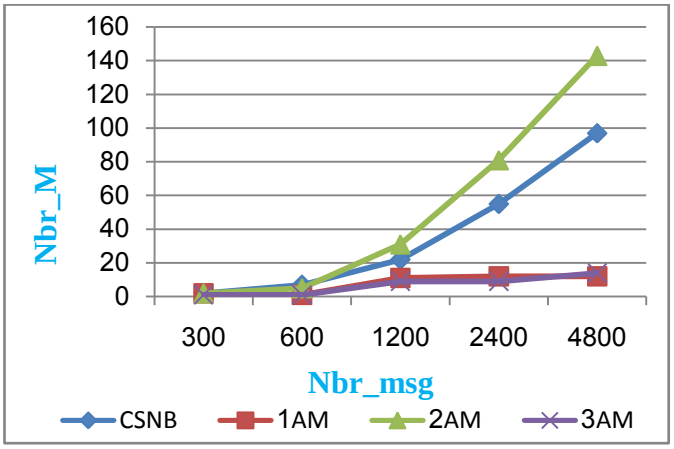
On a obtenu les résultats suivant :



a) Scénario1



b) Scénario2



c) Scénario3

Figure 3.7 L'évaluation de Nbr_M

Dans le scénario 1 : avec l'augmentation du nombre d'initiateurs, nous remarquons l'augmentation de nombre de points de reprise mutable pour l'algorithme *CSNB* et *AM2*.
Ceci est expliqué par le faite que la durée moyenne de la premiere phase très grande pour ces deux algorithmes, alors on peut avoir plus de nombre de point de reprise mutable.

Mêmes remarques pour les deux autres scénarios (scénario 2, 3), si on varie le nombre de processus et le nombre du message.

Les deux algorithmes *AM1* et *AM3* ont des meilleures performances, car ils enregistrent un petit nombre de points de reprise mutable dû un petit espace mémoire utilisé et un taux d'accès minimale au mémoire de stockage.

3.7 Conclusion

D'après les résultats obtenues et après une analyse approfondie, nous pouvons tirer ces remarques :

1. Les performances des algorithmes *CSNB* et *AM1* sont dégradées dans les grands systèmes ayant grand nombre de processus, et dans les systèmes ayant un échange énorme d'information.

2. Pour le nombre de requêtes et le nombre de messages Reply, les algorithmes *AM2* et *AM3* ont des meilleures performances,

3. Les deux algorithmes *AM1* et *AM3* ont des meilleures performances dans l'utilisation de mémoire et la durée de la première phase.

4. Nous pouvons dire que nos améliorations sont plus performantes que les algorithmes *CSB, CSNB*. On peut dire que l'algorithme *AM3* est le meilleur par rapport aux autres algorithmes.

Conclusion générale

Il existe trois approches pour le calcul du point de reprise global cohérent de la reprise des systèmes :

- L'approche coordonnée (Coordinated Checkpointing).
- L'approche non coordonnée (Uncoordinated Checkpointing).
- L'approche induit par communication (Communication-Induced Checkpointing).

À cet effet, Le but de notre travail est de simuler des algorithmes dans la classe coordonnée, on a choisi deux algorithmes (CSNB et CSB) et nous avons proposé des améliorations pour CSNB qui sont AM2, AM1, AM3.

L'évaluation de ces algorithmes a été réalisée sous NS-2, pour cela nous avons varie certain paramètres pour étudier leur influence sur les performances.

À partir de cette simulation, nous avons tiré les remarques suivantes :

1. Les performances des algorithmes CSNB et AM1 sont dégradées dans les grands systèmes ayant grand nombre de processus, et dans les systèmes ayant un échange énorme d'information.

2. Pour le nombre de requêtes et le nombre de messages Reply, les algorithmes AM2 et AM3 ont des meilleures performances,

3. Les deux algorithmes AM1 et AM3 ont des meilleures performances dans l'utilisation de mémoire et la durée de la première phase.

4. Nous pouvons dire que nos améliorations sont plus performantes que les algorithmes *CSB, CSNB*. On peut dire que l'algorithme *AM3* est le meilleur par rapport aux autres algorithmes.

Ce travail nous a permis de :

- Bien comprendre les problèmes des systèmes répartis, surtout le problème de calcul d'état global.

- Comparer par simulation entre les performances des différents algorithmes.

- Maitriser l'outil de simulation NS-2.

Bibliographie

[1]. G.Lelann. P.Minet. D.Powell. *tolérance aux pannes et systèmes répartis : Conception et mécanismes.* Rapport de recherche. Institut National de Recherche en Informatique et Automatique. Novembre 1993.

[2]. P. Anelli. E. Horlait. *NS-2: Principes de conception et d'utilisation (Manuel NS-2).* UPMC - LIP 6 : Laboratoire d'Informatique de Paris VI, 1999.

[3]. G. Cao, M. Singhal Checkpointing with mutable checkpoints, Theoretical Computer Science 290 :1127 – 1148 , 2003

[4]. S.Jafar. *Programmation des systèmes parallèles distribués : tolérance aux pannes, résilience et adaptabilité.* Thèse pour obtenir le grade de docteur de L'INPG (Institut National Polytechnique de Grenoble) Spécialité : "Informatique : Systèmes et Logiciels", le 30 Juin 2006.

[5]. Z.Abdelhafidi. *Points de reprise dans les systèmes répartis Etude basée sur la simulation des protocoles CIC assurant la propriété RDT.* Thèse Magistère de l'université Amar Telidji-Laghouat Spécialité informatique, 2007.

[6]. T.Allaoui. Une nouvelle solution du problème de la K-Exclusion Mutuelle dans les systèmes répartis, Thèse Magistère de l'université Amar Telidji-Laghouat Spécialité informatique, 2007.

[7]. S.Benkouider. S.Labgaa. *Etude des performances des algorithmes répartis d'exclusion mutuelle avec le simulateur NS-2* ,Thèse d'ingénieur de l'université Amar Telidji-Laghouat Spécialité informatique, PFE 2007.

[8]. C.Delbé. *Tolérance aux pannes pour objets actifs asynchrones : protocole, modèle et expérimentations.* Thèse de doctorat de l'université de Nice - Sophia Antipolis Spécialité informatique, 2007.

[9]. M.Kebir. A.Alliliche. *Etude comparative des protocoles de points de reprise de type CIC.* Thèse d'ingénieur de l'université Amar Telidji-Laghouat Spécialité informatique, PFE 2008.

[10]. F.Barkat. L.Ouled kouider. *Etude et simulation des algorithmes de recouvrement arrière : application des protocoles MS et BCS* Thèse d'ingénieur de l'université Amar Telidji-Laghouat Spécialité informatique, PFE 2009.

[11]. X.Besseron. Tolérance aux fautes et reconfiguration dynamique pour les applications distribuées à grande échelle. Thèse pour obtenir le grade de docteur de l'Université de Grenoble Spécialité : "Informatique ", le 28 avril 2010.

[12]. G.Florin. *La tolérance aux pannes dans les systèmes répartis.* Laboratoire CEDRIC CNAM.

Annexe I

Les procédures des algorithmes

1. Algorithme de CAO et Singhal bloquant (CSB)

1. 1 Procédure 1: Lancer le calcule d'état global

```
1. Début
2.    Diffuser (R_request) à tous les processus;
3. Fin.
```

1. 2 Procédure 2: Réception de R_request

```
1. Début
2.     continue := 0 ;    // bloquer leur calcule
3.     Envoyer (VectorR) à l'initiateur ;
4. Fin.
```

1. 3 Procédure 3: Réception de VectorR

```
 1. Début
 2.     Si reçoit tous les VectorR
 3.        Construit la matrice D ; //  chaque ligne (i) représente une Ri de Pi
 4.        Calculer(D);
 5.        pour  chaque processus  P j
 6.           Si (Sforced [j] =1) alors                 // P j ∈Sforced
 7.                Envoyer (Request) à  P j ;
 9.           Sinon   Envoyer (Continue) à P j ;
10.           FinSi ;
11.       Fait ;
12.    FinSi ;
13. Fin.
```

1.4 Procédure 4: Produit (Di(N), D (N, N))

```
1. Début
2.    pour J := 0 à Nbr_proc
3.       S :=0 ;
4.       pour I := 0 à Nbr_proc
5.             S: = S +Di [I]*D [I, J];
6.       Fait;
7.        Di [J]:=S;
8.    Fait ;
9. Fin.
```

1.5 Procédure 5: Calculer (D (N, N))

```
1. Début
2.    A=Di ;
3.    Di=Produit (Di, D) ;
4.    TQ (A<> Di) faire
5.       A :=Di ;
6.       Di=Produit (Di, D) ;
7.    Fait ;
8.    pour k := 0 à Nbr_proc            // Sforced = Ø ;
9.          Sforced [k] := 0 ;
10.   Fait ;
11.   pour k := 0 à Nbr_proc
12.         Sforced [j]:= Di[j];        // si Di[j] =1 alors  Sforced = Sforced  ∪ Pj;
13.   Fait ;
14. Fin.
```

1. 6 Procédure 6: Réception de Continue

```
1. Début
2.    continue := 1 ;    // reprend ses calculs;
3.  Fin.
```

1. 7 Procédure 7: Réception de Request

```
1. Début
2.     prend une checkpoint et note point tentative;
3.     Initialiser(R) ;              // Ri est initialisée à 0 sauf Ri [i] :=1
4.     Envoyer (VectorR) à l'initiateur ;
5.     continue := 1 ;               // reprend ses calculs;
6. Fin.
```

1. 8 Procédure 8: Réception de Reply

```
1. Début
2.     Si reçoit tous les VectorR
3.         Initialiser(R) ;          // Ri est initialisée à 0 sauf Ri [i] :=1
4.     FinSi ;
5. Fin.
```

1. 9 Procédure 9: Réception de Commit

```
1. Début
2.     point tentative → point permanent ;
3. Fin.
```

2. Algorithme de CAO et Singhal non bloquant (CSNB)

2. 1 Procédure 1: Pi envoyer message d'application à Pj

```
1. Début
2.     Si cp_state = 1 alors
3.         Envoyer (Pi, message, csn_i[i], own_trigger) ;
4.         sent_i := 1 ;
5.     Sinon
6.         Envoyer (Pi, message, csn_i[i], NULL) ;
7.         sent_i := 1 ;
8.     FinSi ;
9. Fin.
```

2. 2 Procédure 2: Actions pour l'initiateur Pi

```
1. Début
2.   increment (csn_i [i]) ;
3.   own_trigger:= (Pi, csn_i [i]) ;
4.   cp_state:=1;
5.   pour k := 0 à Nbr_proc
6.         MR [k]csn: = 0;
7.         MR [k] R: = 0;
8.   Fait;
9.   MR [i] csn: = 1;
10.  MR [i] R: = 1;
11.  Proc_cp (Ri, MR, Pi, own_trigger, 1.0);   // weight: = 1.0;
12.  prend un point de reprise local;
13.  old_csn_i:= csn_i [i];
14.  sent_i := 0 ;
15.   Initialiser Ri;     // Ri est initialisée à 0 sauf Ri [i] :=1
16. Fin.
```

2. 3 Procédure 3: proc_cp (Ri, MR, Pi, msg_trigger, recu_ weight)

```
1. Début
2.   weight:= recu_weight;
3.   pour k := 0 à Nbr_proc
4.         temp [k]csn :=max(MR [k]csn, csn_i[k]);
5.         temp [k]R :=max(MR [k]R, Ri[k]);
6.   Fait;
7.   pour k := 0 à Nbr_proc
8.     Si ((Ri[k] = 1) et (max (MR [k]csn, csn_i[i]) <> MR [k]csn)
                    et (Ri[k]) <> MR [k]R)    alors
9.        weight:= weight/2 ;
10.       Envoyer (Pi, request, temp, csn_i[i], msg_trigger, csn_i[k], weight);
11.    FinSi;
12.  Fait;
13. Fin.
```

2. 4 Procédure 4: réception de Request à partir Pj :

```
1. Début
2. réception (Pj, Request, MR, recu_csn,msg_trigger, req_csn,recu_weight) ;
3.  csn_i[j] := recu_csn ;
4.  Si (old_csn_i > req_csn) alors
5.      Envoyer (Pi, Reply, recu_weight);
6.  Sinon
7.      cp_state:= 1 ;
8.      Si (msg_trigger= own_trigger) alors
9.          Si (CPi.trigger= msg_trigger) alors
10.             Proc_cp (CPi.Ri, MR, Pi, msg_trigger, recu_weight);
11.             sauve checkpoint mutable sur le stockage stable ;
12.            old_csn_i:= csn_i[i] ;
13.            CPi := NULL ;
14.             Envoyer (Pi, Reply, weight);
15.         Sinon
16.             Envoyer (Pi, Reply, recu_weight);
17.         FinSi;
18.     Sinon
19.          increment (csn_i [i]) ;
20.          own_trigger= msg_trigger;
21.         Proc_cp (Ri, MR, Pi, msg_trigger, recu_weight);
22.         prend un point de reprise local;
23.         old_csn_i:= csn_i [i];
24.         Envoyer (Pi, Reply, recu_weight);
25.         sent_i := 0 ;
26.         Initialiser Ri;     // Ri est initialisée à 0 sauf Ri [i] :=1
27.     FinSi;
28.  FinSi;
29.Fin.
```

2. 5 Procédure 5: réception de Message à partir Pj :

```
1. Début
2.  réception (Pj, message, recu_csn, msg_trigger) ;
3.  Si  (recu_csn ≤ csn_i[j]) alors
4.     Ri[j]:= 1 ;
5.     Traiter le message ;
6.  Sinon
7.    Si (csn_i[msg_trigger.Pid]≥ msg_trigger.inum) alors
8.        csn_i [j] := recu_csn ;
9.        Ri[j]:= 1 ;
10.        Traiter le message ;
11.   Sinon
13.      csn_i [j] := recu_csn ;
12.      Si ((msg_trigger<>NULL)et(sent_i=1)et(msg_trigger<> own_trigger) )
8.          prend un point de reprise mutable;
9.          CPi.trigger := msg_trigger;
10.         CPi.R :=Ri ;
11.         CPi.sent := sent_i ;
12.         sent_i :=0 ;
13.         Initialiser Ri;     // Ri est initialisée à 0 sauf Ri [i] :=1
14.      FinSi;
15.      Si ((msg_trigger<>NULL)et(cp_state_i=0))
16.         cp_state_i:=1;
17.         increment (csn_i [i]) ;
18.         own.trigger := msg_trigger;
19.      FinSi;
20.      Ri[j]:= 1;
21.      Traiter le message ;
22.   FinSi;
23. FinSi;
24.Fin.
```

2. 6 Procédure 6: Réception de Reply

```
1. Début
2.    réception (Pi, Reply, recu_weight);
3.    weight:= weight+ recu_weight;
4.    Si (weight = 1) alors
5.       Diffuser (Commit, msg_trigger) ;
6.       cp_statei:=0;
7.       date_fin_ :=Date_système ;
8.    FinSi ;
9. Fin.
```

2. 7 Procédure 7: Réception de Commit

```
1. Début
2.    réception (Commit, msg_trigger) ;
3.    cp_statei:=0;
4.    Si ((CPi.trigger = msg_trigger) et (CPi < > NULL))
5.       senti := senti ∪ CPi. sent ;
6.       Ri :=Ri ∪ CPi. Ri ;
7.       CPi := NULL ;
7.    FinSi ;
8.    point tentative → point permanent ;
9. Fin.
```

Annexe II

Exemple de code source

1. Le fichier CSNB.h :

```
//*************************************************************************************//
//                                          CSNB .h                                    //
//*************************************************************************************//

#ifndef ns_CSNB_h
#define ns_CSNB_h

#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"

//*************************************************************************************//
typedef struct trigger   {

        int pid;                        // L'identifiant de l'initiateur
        int inum;                       // numéro de séquence de l'initiateur
}trigger;

typedef struct MRI{
        int  csn[64];                   //Les numéros de séquence
        bool R[64];                     //vecteur de dependance
}MRI;

typedef struct CPi {
        const char*    estmutable;     //mutable Check point de Pi
        bool    R[64];                  //Vecteur de dependance
        trigger CPi_trigger;            //Le numéro et csn de l'initiateur de point de reprise actuelle
        bool    sent;                   //Etat de processus si envoit ou non dans le point de reprise
actuelle
} CPi;

//*************************************************************************************//
//                          Les packet utiliser                                        //
//*************************************************************************************//
struct hdr_request {
        const char*    type;        // type de request
        int     id_proc;             // L'identifiant d'émetteur
        MRI     temp;                // Les numéros de séquence et vecteur de dependance commun
        int     Mcsn;                // (Mcsn) numéro de séquence d'émetteur
        trigger msg_trigger;         // L'identifiant et numéro de séquence de l'initiateur
        int     Vcsn;                // (Vcsn) numéro de séquence de récepteur
        double   weight;             // poit d'émetteur

                // La méthode pour accéder à l'entête du paquet

        static int offset_;
        inline static int &offset() { return offset_; }
        inline static hdr_request* access(const Packet* p) {
                return (hdr_request*) p->access(offset_);
        }
};
```

```
struct hdr_reply {
                    const char*   type;              // type de reply
                   int    id_proc;                  // L'identifiant d'émetteur
                   double  weight;                  // poit d'émetteur

                        // La méthode pour accéder à l'entête du paquet

                  static int offset_;
                  inline static int &offset() { return offset_; }
                  inline static hdr_reply* access(const Packet* p) {
                                return (hdr_reply*) p->access(offset_);
          }
};

struct hdr_messageApp {
                const char*    type;        // type de message
                int     id_proc;            // L'identifiant d'émetteur
                int     Mcsn;               // (Mcsn) numéro de séquence d'émetteur superposé avec le message
                trigger own_trigger;        // trigger pour l'initiateur de point de reprise actuelle

                        // La méthode pour accéder à l'entête du paquet

                  static int offset_;
                  inline static int &offset() { return offset_; }
                  inline static hdr_messageApp* access(const Packet* p) {
                              return (hdr_messageApp*) p->access(offset_);

          }
};

struct hdr_commit {

                  const char*    type;          // type de commit
                  trigger msg_trigger;          // L'identifiant et numéro de séquence de l'initiateur

                              // La méthode pour accéder à l'entête du paquet
                  static int offset_;
                  inline static int &offset() { return offset_; }
                  inline static hdr_commit* access(const Packet* p) {
                               return (hdr_commit*) p->access(offset_);
          }
};
//*************************************************************************************//
//                  Création de la classe CSNBAgent héritée de la classe Agent                  //
//*************************************************************************************//

class CSNBAgent : public Agent {

        public:

                CSNBAgent();
               int      nbr_proc;              // Nombre de processus
```

```
        virtual int command(int argc, const char*const* argv);
        virtual void recv(Packet*, Handler*);
        void Reset(bool D[],int id);
        void proc_cp(bool Ri[] ,MRI MR1 ,int id ,trigger ms_trigger,double recu_weigth);
        void Broadcast_Commit(trigger ms_trigger);
        void Rj_U_CPiR(bool R1[],bool R2[]);

    protected:
        int     nbr_mutable;        // Nombre de point de contrôle mutable
        int     nbr_mutable_forced;// Nombre de point de contrôle mutable forcé
        int     nbr_request;        // Nombre des requètes
          int     nbr_reply;        // Nombre de reply
        int     old_csn;            // dernier numéro de séquence
        double  date_fin_;          // date fin de première phase
        int     sent;               // Si envoit ou non dans le point de reprise actuelle
        int     cp_state;           // Si entrer ou non dans le point de reprise actuelle
        int     recv_m;             // Si reçoit un msg ou non dans le point de reprise actuelle
        int     recv_c;             // Etat de processus si reçoit une commit ou non
        bool    R[64];              // Vecteur de dependance
        int     csn[64];            // Les numéros de séquence
        MRI     MR;                 // Les numéros de séquence et vecteur de dépendance commun
        double  weight;             // Poit
        trigger own_trigger;   // L'identifiant de l'initiateur actuelle
        CPi     CP;
};
#endif
//*************************************************************************************//
//                                  Fin_CSNB.h                                         //
//*************************************************************************************//
```

2. Le fichier CSNB.cc :

```
//*************************************************************************************//
//                                  CSNB.cc                                            //
//*************************************************************************************//
#include "CSNB.h"
#include "agent.h"
int hdr_request::offset_;
int hdr_reply::offset_;
int hdr_messageApp::offset_;
int hdr_commit::offset_;

//*************************************************************************************//
//                                pour calculer offset de l'entête                     //
//*************************************************************************************//

static class requestHeaderClass : public PacketHeaderClass {
public:

        requestHeaderClass() : PacketHeaderClass("PacketHeader/CSNB",
                                sizeof(hdr_request)) {
                                        bind_offset(&hdr_request::offset_);
                                }
```

```
} class_requesthdr;
//*************************************************************************************//

static class replyHeaderClass : public PacketHeaderClass {
public:

        replyHeaderClass() : PacketHeaderClass("PacketHeader/CSNB",
                                sizeof(hdr_reply)) {
                                        bind_offset(&hdr_reply::offset_);
                                }

} class_replyhdr;
//*************************************************************************************//

static class messageHeaderClass : public PacketHeaderClass {
public:

        messageHeaderClass() : PacketHeaderClass("PacketHeader/CSNB",
                                sizeof(hdr_messageApp)) {
                                        bind_offset(&hdr_messageApp::offset_);
                                }

} class_messagehdr;
//*************************************************************************************//

static class commitHeaderClass : public PacketHeaderClass {
public:

        commitHeaderClass() : PacketHeaderClass("PacketHeader/CSNB",
                                sizeof(hdr_commit)) {
                                        bind_offset(&hdr_commit::offset_);
                                }

} class_commithdr;

//*************************************************************************************//
//                                pour créer une intance à partir de tcl                 //
//*************************************************************************************//
static class CSNBClass : public TclClass {
public:

        CSNBClass() : TclClass("Agent/CSNB") {}
        TclObject* create(int, const char*const*){
                                        return (new CSNBAgent());
                                }
} class_CSNB;




//*************************************************************************************//
//                        pour faire la lien entre les variables en C++ et en TCL (bind)  //
//*************************************************************************************//
```

```
CSNBAgent::CSNBAgent() :
Agent(PT_CSNB),nbr_proc(8),nbr_mutable(0),nbr_mutable_forced(0),nbr_request(0),nbr_reply(0),
old_csn(0),date_fin_(0),sent(0),cp_state(0),recv_m(0),recv_c(0)
{
        bind("packetSize_", &size_);
        bind("nbr_proc",&nbr_proc);
        bind("nbr_mutable",&nbr_mutable);
        bind("nbr_mutable_forced",&nbr_mutable_forced);
        bind("nbr_request",&nbr_request);
        bind("nbr_reply",&nbr_reply);
        bind("old_csn",&old_csn);
        bind("date_fin_",&date_fin_);
        bind("sent",&sent);
        bind("cp_state",&cp_state);
        bind("recv_m",&recv_m);
        bind("recv_c",&recv_c);

}
//*************************************************************************//
//                    Initialisation de vecteur de dépendance              //
//*************************************************************************//

void CSNBAgent::Reset(bool D[],int id)
{
int i;
for( i = 0 ; i < nbr_proc ; i++ ) {
     D[i]=0;                                  // Ri est initialisée à 0
}
D[id]=1;                                       // Sauf Ri[i]
}

//*************************************************************************//
//                          Fonction de max entre deux valeur              //
//*************************************************************************//

int max(int a,int b)
{
int c;
  if (a<b)
     c=b;
    else
      c=a;
return c;
}




//*************************************************************************//
//                          Diffusion de message Commit                    //
//*************************************************************************//
void CSNBAgent::Broadcast_Commit(trigger ms_trigger)
```

```
{int k;
for( k = 0 ; k < nbr_proc ; k++ )
            {
    if (k!= here_.addr_ )
       {
            dst_.addr_ = k;                                  // Adresse destination est processus(k)
            size_ =600*nbr_proc;                             // Pour la taille de paquet
            fid_ = 2;                                        // Couleur vert
            Packet* pkt = allocpkt();                        // Création d'un nouveau paquet
            hdr_commit* Chdr = hdr_commit::access(pkt);      // L'accès à l'entête des paquets
            Chdr->type="Commit";
            Chdr->msg_trigger=ms_trigger;
            send(pkt, 0);
       }
       }
}
//*************************************************************************************//
bool sentj_U_CPjsent(int sent1,int sent2)
{
if (sent2==1)
   sent1=sent2;
return sent1;
}
//*************************************************************************************//
void CSNBAgent::Rj_U_CPiR(bool R1[],bool R2[])
{
int k;
for( k = 0 ; k < nbr_proc ; k++ )
 {
  if (R2[k]==1)
    R1[k]=R2[k];
 }
}
//*************************************************************************************//
void CSNBAgent::proc_cp(bool Ri[] ,MRI MR1 ,int id ,trigger ms_trigger,double recu_weight)
{
MRI temp;
int k;
weight=recu_weight;
          for( k = 0 ; k <  nbr_proc ; k++ ) {
                                temp.csn[k]=max(MR1.csn[k],csn[k]);
                                temp.R[k]=max(MR1.R[k],Ri[k]);
                               }

          for( k = 0 ; k < nbr_proc ; k++ ) {
           if((Ri[k]==1)&&(max(MR1.csn[k], csn[id])!= MR1.csn[k]) && (Ri[k]!= MR1.R[k]))
            {
              weight=weight /(double  )2 ;                   // Chaque envoit en divise weight sur deux
              size_ =600*nbr_proc;                           // Pour la taille de paquet
              fid_ = 1;                                      // Couleur rouge
              dst_.addr_ = k;                                // Adresse destination est processus(k)
              Packet* pkt = allocpkt();                      // Création d'un nouveau paquet
              hdr_request* RQhdr = hdr_request::access(pkt); // L'accès à l'entête des paquets
              RQhdr->type="Request";
```

```
            RQhdr->id_proc=id;
            RQhdr->temp=temp;
            RQhdr->Mcsn=csn[id];
            RQhdr->msg_trigger=ms_trigger;
            RQhdr->Vcsn=csn[k];
            RQhdr->weight=weight;
            send(pkt, 0);                          // Envoyer le paquet
           }
                                  }
}
//*************************************************************************************//
//                                  pour envoyer les paquets                           //
//*************************************************************************************//
int CSNBAgent::command(int argc, const char*const* argv)
{
int k;
  if (argc == 2) {
    if (strcmp(argv[1], "send-message") == 0) {
            size_ =2400/nbr_proc;                   // Pour la taille de paquet
            fid_ = 0;                               // Couleur Bleu
            Packet* pkt = allocpkt();               // Création d'un nouveau paquet
            hdr_messageApp* Mhdr = hdr_messageApp::access(pkt);      // L'accès à l'entête des
paquets
            if (cp_state==1)
                Mhdr->own_trigger=own_trigger;
            else
                Mhdr->own_trigger.pid= -1;
            sent=1;
            Mhdr->type="Message";
            Mhdr->id_proc=here_.addr_;
            Mhdr->Mcsn=csn[here_.addr_];
            send(pkt, 0);                           // Envoyer le message
     return (TCL_OK); }

    else if (strcmp(argv[1], "initiator") == 0){

          csn[here_.addr_]++;
          own_trigger.pid=here_.addr_;
          own_trigger.inum= csn[here_.addr_];
          cp_state=1;
          for( k = 0 ; k < nbr_proc ; k++ ) {
                              MR.csn[k]=0;
                              MR.R[k]=0;
                              }
          MR.csn[here_.addr_]=csn[here_.addr_];
          MR.R[here_.addr_]=1;
          nbr_reply=0;
          proc_cp(R,MR,here_.addr_,own_trigger,1);
          old_csn=csn[here_.addr_];
          sent=0;
          Reset(R,here_.addr_);


       return (TCL_OK); }
```

```
   else  if (strcmp(argv[1], "initiator_local") == 0) {
          nbr_reply=0;
          csn[here_.addr_]++;
          old_csn=csn[here_.addr_];
          own_trigger.pid=here_.addr_;
          own_trigger.inum= csn[here_.addr_];
          sent=0;
          date_fin_=Scheduler::instance().clock();
       return (TCL_OK); }
   else  if (strcmp(argv[1], "initial") == 0) {
          for( k = 0 ; k < nbr_proc ; k++ )
            {
             R[k]=0;
             csn[k]=0;
             MR.csn[k]=0;
             MR.R[k]=0;
            }
          R[here_.addr_]=1;
          own_trigger.pid= -1;
          weight=0;
          CP.estmutable="Null";
       return (TCL_OK);}
            }
return (Agent::command(argc, argv));
}
//*************************************************************************************//
//                                  pour traiter les messages reçu                     //
//*************************************************************************************//

void CSNBAgent::recv(Packet* pkt, Handler*)
{
int k,i ;

hdr_messageApp* Mhdr = hdr_messageApp::access(pkt);
hdr_request* RQhdr = hdr_request::access(pkt);
hdr_reply* Rhdr = hdr_reply::access(pkt);
hdr_commit* Chdr = hdr_commit::access(pkt);


if (Mhdr->type=="Message")
 {
 recv_m=1;
 if (csn[Mhdr->id_proc]>= Mhdr->Mcsn)                    //message reçu aprés request
            {
             R[Mhdr->id_proc]=1;                    //mais envoye avant le request
            //traiter le message;
             Packet::free(pkt);
            }
  else if ( csn[Mhdr->own_trigger.pid] >= Mhdr->own_trigger.inum )      // déjà recu request
    {
             csn[Mhdr->id_proc]= Mhdr->Mcsn;
             R[Mhdr->id_proc]=1;
             //traiter le message;
             Packet::free(pkt);
```

```
    }
    else
        {
            csn[Mhdr->id_proc]= Mhdr->Mcsn;
        if ((Mhdr-> own_trigger.pid !=-1)&& ( sent==1) && ( own_trigger.pid != Mhdr->
own_trigger.pid)
              &&( own_trigger.inum != Mhdr-> own_trigger.inum))
           {
            nbr_mutable=nbr_mutable+1;                  // point de reprisr mutable
            CP.estmutable="Mutable";
            CP.CPi_trigger=Mhdr-> own_trigger;
           for( k = 0 ; k < nbr_proc ; k++ )
             {
              CP.R[k]= R[k];
             }
            CP.sent= sent;
            sent= 0;
           Reset( R,here_.addr_);
          }

        if ((Mhdr-> own_trigger.pid !=-1) && ( cp_state==0))
          {  cp_state=1;
            csn[here_.addr_] ++;
            own_trigger = Mhdr-> own_trigger;
           }
            R[Mhdr->id_proc]=1;
           //traiter le message;

          Packet::free(pkt);
        }

 }
//*************************************************************************************//
//                                    pour traiter les requests reçoit                                    //
//*************************************************************************************//
else if (RQhdr->type=="Request")
 {  i=here_.addr_;
  csn[RQhdr->id_proc]=RQhdr->Mcsn;
  if (old_csn > RQhdr->Vcsn)
            {
             size_ =2400/nbr_proc;                          // Pour la taille de paquet
             fid_ = 3;                                      // Couleur  Violet
             dst_.addr_ =RQhdr-> msg_trigger.pid;       // Adresse destination l'initiateur
             Packet* pkt = allocpkt();                       // Création d'un nouveau paquet
             hdr_reply* Rhdr = hdr_reply::access(pkt); // L'accès à l'entête des paquets
             Rhdr->type="Reply";
             Rhdr->id_proc= here_.addr_;
             Rhdr->weight=RQhdr->weight;
             send(pkt, 0);                                  // Envoyer le paquet
            }
   else
            {
             cp_state=1;
```

```
            if ( (own_trigger.pid==RQhdr-> msg_trigger.pid)&&(own_trigger.inum==RQhdr->
msg_trigger.inum))
                {
                 if (( CP.CPi_trigger.pid==RQhdr-> msg_trigger.pid)
                      &&( CP.CPi_trigger.inum==RQhdr-> msg_trigger.inum))
                    {
                    proc_cp( CP.R,RQhdr->temp,here_.addr_,RQhdr->msg_trigger,RQhdr->weight);
                    nbr_mutable_forced ++;                //save mutable on stable storage
                    old_csn =  csn[here_.addr_];
                    CP.estmutable="Null";
                    size_ =2400/nbr_proc;                 // Pour la taille de paquet
                    fid_ = 3;                             // Couleur Violet
                    dst_.addr_ =RQhdr-> msg_trigger.pid;     // Adresse destination est l'initiateur
                    Packet* pkt = allocpkt();             // Création d'un nouveau paquet
                    hdr_reply* Rhdr = hdr_reply::access(pkt);// L'accès à l'entête des paquets
                    Rhdr->type="Reply";
                    Rhdr->id_proc= here_.addr_;
                    Rhdr->weight= weight;
                    send(pkt, 0);                         // Envoyer paquet
                     }
                    else
                       {
                        size_ =2400/nbr_proc;                 // Pour la taille de paquet
                        fid_ = 3;                             // Couleur  Violet
                        dst_.addr_ =RQhdr-> msg_trigger.pid;      // Adresse destination
                        Packet* pkt = allocpkt();             // Création d'un nouveau paquet
                        hdr_reply* Rhdr = hdr_reply::access(pkt); // L'accès à l'entête des paquets
                        Rhdr->type="Reply";
                        Rhdr->id_proc= here_.addr_;
                        Rhdr->weight=RQhdr->weight;
                        send(pkt, 0);                         // Envoyer le paquet
                       }
                }
                 else  {

                    own_trigger=RQhdr-> msg_trigger;
                    csn[i]=csn[i]+1;                      // point tentative
                    proc_cp( R,RQhdr->temp,here_.addr_,RQhdr->msg_trigger,RQhdr->weight);
                    old_csn =   old_csn+1;
                    size_ =2400/nbr_proc;                         // Pour la taille de paquet
                    fid_ = 3;                                     // Couleur  Violet
                    dst_.addr_ =RQhdr-> msg_trigger.pid;     // Adresse destination est l'initiateur
                    Packet* pkt = allocpkt();                     // Création d'un nouveau paquet
                    hdr_reply* Rhdr = hdr_reply::access(pkt);// L'accès à l'entête des paquets
                    Rhdr->type="Reply";
                    Rhdr->id_proc= here_.addr_;
                    Rhdr->weight= weight;
                    send(pkt, 0);                         // Envoyer le paquet
                    sent=0;
                    Reset( R,here_.addr_);
                    }
            }

}
```

```
//*************************************************************************************//
//                                pour traiter les reply reçoit                        //
//*************************************************************************************//
 else if(Rhdr->type=="Reply")
     {
     nbr_reply++;
     weight= weight + Rhdr->weight;                    // faire la somme de weight
     if ( weight==1)                                   // jusqu'a que weight =1
                        {
                         cp_state=0;
                         Broadcast_Commit(own_trigger);          // faire diffusion de commit
                         nbr_request=nbr_reply;
                         date_fin_=Scheduler::instance().clock();
                         recv_c=1;
                        }
    Packet::free(pkt);
     }
//*************************************************************************************//
//                                pour traiter les commit reçoit                       //
//*************************************************************************************//
    else if(Chdr->type=="Commit") {
          recv_c=1;
          csn[Chdr-> msg_trigger.pid]=Chdr-> msg_trigger.inum;        // enregistrer csn de l'initiateur
          cp_state=0;
          if(( CP.CPi_trigger.pid==Chdr-> msg_trigger.pid)&&
            ( CP.CPi_trigger.inum==Chdr-> msg_trigger.inum)&&( CP.estmutable!="Null"))
                        {                                 // return a l'etat enregistrer déjà
                         sent= sentj_U_CPjsent( sent, CP.sent);
                         Rj_U_CPiR( R, CP.R);
                         CP.estmutable="Null";
                        }
                        Packet::free(pkt);
                       recv_m=0; k = 0 ;
                         while( (recv_m!=1) && (k < nbr_proc) )
                           {
                            if( R[k]==1 && k != here_.addr_)
                                recv_m=1;
                            k++;
                           }
                       }
}
```

```
//*************************************************************************************//
//                                Fin_CSNB.cc                                          //
//*************************************************************************************//
```

3. Le fichier scenario.tcl :

```
#*************************************************************************************#
#                                scenario.txt                                         #
#*************************************************************************************#
```

```
set s [open "scenario.txt" w]

puts "Entrez le nombre de processus: "
gets stdin nbr_
puts $s "$nbr_"

puts "Entrez le temps de simulation: "
gets stdin date_
puts $s "$date_"

puts "Entrez le nombre de message : "
gets stdin nbr_msg
puts $s "$nbr_msg"

puts "Entrez le nombre d'initiateur : "
gets stdin nbr_I
puts $s "$nbr_I"
#*************************************************************************************#
#                                Définir la distribtion des dates d'intialisation                                 #
#*************************************************************************************#
set randSeed 0
set interval_ [expr $date_/$nbr_I ]
for {set i 0} {$i <$nbr_I} {incr i} {
            set  nown [expr $interval_*[expr $i + rand()]]
            set  initiator($i,0) $nown
            }
#*************************************************************************************#
#                                        les identifiant des initiateurs                                          #
#*************************************************************************************#
set randSeed 25
set initiator(0,1) [expr int(rand()*$nbr_)]
for {set i 1} {$i <$nbr_I} {incr i} {
                set id_initiator [expr int(rand()*$nbr_)]
                while { $id_initiator == $initiator([expr $i-1],1) } {
                      set id_initiator [expr int(rand()*$nbr_)]
                    }
                set initiator($i,1) $id_initiator
                          }
#*************************************************************************************#
#                          souvgarder les information des initialisation dans fichier                             #
#*************************************************************************************#
for {set j 0} {$j <$nbr_I} {incr j} {
                              puts $s "$initiator($j,0)"
                              puts $s "$initiator($j,1)"          }
#*************************************************************************************#
#                              Prendre les temps pour faire les envoies des messages                              #
#*************************************************************************************#
set randSeed 50
set temps $date_
for { set i 0} {$i < $nbr_msg } { incr i} {

          set nown [expr rand()*$temps]

          set envoi_message($i,0) $i
```

```
            set envoi_message($i,1) $nown

                                }

#*****************************************************************************#
#                                  Tri les messages                           #
#*****************************************************************************#
for {set i 0} {$i <[expr $nbr_msg -1]} {incr i} {

          for {set j [expr $i+1]} {$j <$nbr_msg} {incr j} {

                    if { $envoi_message($i,1) > $envoi_message($j,1)} {

                              set message $envoi_message($i,1)
                    set envoi_message($i,1) $envoi_message($j,1)
                    set envoi_message($j,1) $message
                                                  }
                                        }
                              }
#*****************************************************************************#
#              Prendre les informations d'envoies et de réceptions des messages #
#*****************************************************************************#
set m [open "messages.txt" w]

set randSeed 75
for {set j 0} {$j <$nbr_msg} {incr j} {
          set src_ [expr int(rand()*$nbr_)]
          set tab_src_($j) $src_
                              }

set randSeed 100
for {set j 0} {$j <$nbr_msg} {incr j} {
          set dest_ [expr int(rand()*$nbr_)]
          set src_ $tab_src_($j)
          while { $src_ == $dest_} {
                    set dest_ [expr int(rand()*$nbr_)]
                              }
          for {set k 0} {$k < 3} {incr k} {
                                        switch $k {
                                        0 { set inf_message($j,$k) $envoi_message($j,1) }
                                        1 { set inf_message($j,$k) $src_ }
                                        2 { set inf_message($j,$k) $dest_ }
                                       }
                              puts $m "$inf_message($j,$k)"
                                   }
                                 }
#*****************************************************************************#
  close $m;                                                                    #
  close $s;                                                                    #
#*****************************************************************************#
```

4. Le fichier CSNB.tc :

```
#*************************************************************************************#
#                                        CSNB .tcl                                    #
#*************************************************************************************#
set ns [new Simulator];                                          # Création d'un simulateur #
set bw CSNB.dat;                                                 # Le ficher de trace       #
set nf [open CSNB.nam w];                                        #    Ouverture du NAM      #
set f0 [open $bw w];
set ft [open CSNB.tr w];
$ns trace-all $ft;
$ns namtrace-all $nf;
#*************************************************************************************#
#                          Procedure de Terminaison de la simulation                  #
#*************************************************************************************#
proc finish {} {
global ns f0 n nbr nbr_msg p nbr_mutable nbr_mutable_forced old_csn cp_state sent nbr_I initiator

puts $f0 " ------------------------------------------------------------------------------------------  "
puts $f0 "                                                                                            "
puts $f0 "                          ----------------------------                                      "
puts $f0 "                         | Les résultats de la simulation |                                 "
puts $f0 "                          ----------------------------                                      "
puts $f0 "                                                                                            "
puts $f0 " ------------------------------------------------------------------------------------------  "
puts $f0 "                                                                                            "
puts $f0 " Dans cette simulation, nous utilisons les paramètres suivant:                              "
puts $f0 "                                                                                            "
puts $f0 " - Le nombre des processus est          :   $nbr                                            "
puts $f0 "                                                                                            "
puts $f0 " - Le nombre des initiateurs est        :   $nbr_I                                          "
puts $f0 "                                                                                            "
puts $f0 " - Le nombre des messages est           :   $nbr_msg                                       "
puts $f0 "                                                                                           "
puts $f0 " ------------------------------------------------------------------------------------------  "
puts $f0 " Et alors les résultats  sont comme suit:                                                   "
puts $f0 " ------------------------------------------------------------------------------------------  "
puts $f0 "                                                                                            "

set nbr_req 0
for {set i 0} {$i <$nbr_I} {incr i} {
                        set req $initiator($i,4)
                        set nbr_req [ expr $nbr_req + $req ]
                         }

puts $f0 " -Le nombre de requete est              :   $nbr_req                                        "
puts $f0 "                                                                                            "
puts $f0 " ------------------------------------------------------------------------------------------  "
puts $f0 "                                                                                            "

set nbr_mtb 0
for {set i 0} {$i <$nbr} {incr i} {
                        set nbr_mutable_ [$p($i) set nbr_mutable]
                        set nbr_mtb [ expr $nbr_mtb + $nbr_mutable_ ]
```

```
                        }

puts $f0 " - Le nombre de point mutable est       :   $nbr_mtb                                                    "
puts $f0 "                                                                                                        "
puts $f0 " ------------------------------------------------------------------------------------------------------  "

set nbr_mtb_f 0
for {set i 0} {$i <$nbr} {incr i} {
                        set nbr_mutable_forced_ [$p($i) set nbr_mutable_forced]
                        set nbr_mtb_f [ expr $nbr_mtb_f + $nbr_mutable_forced_ ]
                         }

puts $f0 "                                                                                                        "
puts $f0 " - Le nombre de point mutable forced est :   $nbr_mtb_f                                                  "
puts $f0 "                                                                                                        "
puts $f0 " ------------------------------------------------------------------------------------------------------  "

puts $f0 "                                                                                                       "
puts $f0 " -Les résultats d'intialisation pour chaque initiateur est :                                           "
puts $f0 "                                                                                                       "
set s 0
set nbr_i 0
for {set i 0} {$i <$nbr_I} {incr i} {
                        set fin $initiator($i,2)
                        set debut $initiator($i,0)
                        set initiator($i,3) [expr $fin-$debut ]
                        if { $initiator($i,3)!= 0 } {
                            incr nbr_i
                            set s [expr $s + $initiator($i,3) ]
                                        }
                          }

for {set i 0} {$i <$nbr_I} {incr i} {

puts $f0 " ----------------------------------------                                                          "
puts $f0 "                                                                                                   "
puts $f0 " - Initiateur [expr $i+1] est $initiator($i,1):                                                    "
puts $f0 "                          -La date debut est          : $initiator($i,0)                           "
puts $f0 "                          -La durée d'intialisation est : $initiator($i,3)                         "
puts $f0 "                          -Le nombre de requète est     : $initiator($i,4)                          "



                          }

if { $nbr_i != 0 } {
    set moy [expr $s/$nbr_I]

puts $f0 " ----------------------------------------                                                          "
puts $f0 "                                                                                                   "
puts $f0 " La durée moyen d'intialisation est : $moy                                                         "
puts $f0 "                                                                                                   "
              }
```

```
puts $f0 " ----------------------------------------------------------------------------------------------   "
puts $f0 "                                                                                                  "
puts $f0 " - Le dernier etat de chaque processus est :                                                      "
puts $f0 "                                                                                                  "
puts $f0 "   // Old_csn :dernier numero de sequence                                                         "
puts $f0 "   // cp_state:l'etat si en cours d'intialisation (1) ou non (0)                                   "
puts $f0 "   // sent    :l'etat si envoit (1) un message aprés le dernier checkpoint                       "
puts $f0 " ----------------------------------------                                                        "

for {set i 0} {$i <$nbr} {incr i} {

                        set old [$p($i) set old_csn]
                        set mb [$p($i) set nbr_mutable ]
                        set mbf [$p($i) set nbr_mutable_forced ]

puts $f0 " - Le processus $i :                                                                              "
puts $f0 "                  Old_csn            =  $old                                                      "
puts $f0 "                  nbr_mutable        =  $mb                                                       "
puts $f0 "                  nbr_mutable_forced   =  $mbf                                                    "
puts $f0 " ----------------------------------------                                                         "
puts $f0 "                                                                                                  "

                        }

puts $f0 " ------------------------------------------------------------------------------                 "

$ns flush-trace
close $f0
exec nam CSNB.nam &
exit 0
}
#*************************************************************************************#
#                                  Procedure pour l'affichage finale                                  #
#*************************************************************************************#
proc Afficher_fin { j} {
global ns t n p old_csn nbr date_fin_ initiator nbr_mutable nbr_mutable_forced  nbr_request
set date_fin [$p($initiator($j,1)) set date_fin_]
set nown [$ns now]
if { $date_fin==0 } {
                set nown [expr $nown + 1]
                $ns at $nown "Afficher_fin $j"
                } else {
                for {set i 0} {$i <$nbr} {incr i} {

                        set old [$p($i) set old_csn]
                        set mb [$p($i) set nbr_mutable ]
                        set mbf [$p($i) set nbr_mutable_forced ]

set nown [expr $nown + 1 ]

$ns at $nown"$ns trace-annotate\"le processus($i):Old_csn = $old,nbr_mutable= $mb
,nbr_mutable_forced = $mbf\""

                                        }
```

```
$ns at $nown"$ns trace-annotate \"* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *\""
$ns at $nown"$ns trace-annotate \"* * * * * * * *FIN D'INSTIALISATION* * * * * * * *\""
$ns at $nown"$ns trace-annotate \"* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *\""

set initiator($j,2) $date_fin
set initiator($j,4) [$p($initiator($j,1)) set nbr_request]

}
}
#*************************************************************************************#
#                                  Procedure pour l'affichage                         #
#*************************************************************************************#
proc Afficher { j} {
global ns t n p old_csn cp_state sent nbr nbr_I initiator nbr_mutable nbr_mutable_forced
set ns [Simulator instance]
set nown [$ns now]
                    # Afficher sur le trace #
$ns at $nown "$ns trace-annotate \" Dans $nown en lancer l'initiateur ($initiator($j,1)) \""

for {set i 0} {$i <$nbr} {incr i} {
                     set old [$p($i) set old_csn]
                    set mb [$p($i) set nbr_mutable ]
                   set mbf [$p($i) set nbr_mutable_forced ]

set nown [expr $nown + 1 ]
$ns at $nown"$n strace-annotate\"le processus($i):Old_csn = $old,nbr_mutable= $mb
,nbr_mutable_forced = $mbf\""
                    }

if {$j==[expr $nbr_I-1]} {
                    set nown [$ns now]
                    $ns at $nown "Afficher_fin $j"
                   }
}

#*************************************************************************************#
#                                  Procedure pour l'affichage de old_csn              #
#*************************************************************************************#
proc Afficher_old_csn {} {
global n p old_csn nbr_I nbr initiator date_fin_
set ns [Simulator instance]
set nown [$ns now]

for {set i 0} {$i <$nbr} {incr i} {
                   set old [$p($i) set old_csn]
                   $ns at $nown "$n($i) label \" Old_csn = $old   \"";# Afficher Old_csn_i  dans label_i#
                        }

set date_fin $initiator([expr $nbr_I-1],2)
if { $date_fin== 0 } {
               set nown [expr $nown+0.001]
               $ns at $nown "Afficher_old_csn"
               }
```

```
}

#*****************************************************************************#
#                          Procedure pour lancer l' initialisation            #
#*****************************************************************************#
proc lancer_initiator { j } {

global ns n p recv_m recv_c initiator nbr date_fin_ nbr_request nbr_I
set ns [Simulator instance]
set r [$p($initiator($j,1)) set recv_m]
set nown [$ns now]
set initiator($j,0) $nown
if { $j!=0 } {
   set k 0
   set date_fin [$p($initiator([expr $j-1],1)) set date_fin_]
    set rc [$p($initiator($j,1)) set recv_c]

   if { $initiator([expr $j-1],5)==0 || $date_fin ==0 || $rc==0 } {
        set nown [expr $nown + 0.01 ]
        $ns at $nown "lancer_initiator $j"
        } else {
              set initiator($j,5) 1
              set initiator([expr $j-1],2) [$p($initiator([expr $j-1],1)) set date_fin_]
              set initiator([expr $j-1],4) [$p($initiator([expr $j-1],1)) set nbr_request]
              $p($initiator([expr $j-1],1)) set date_fin_ 0
              set r [$p($initiator($j,1)) set recv_m]
              for {set k 0} {$k <$nbr_I} {incr k} {
                                    $p($initiator($k,1)) set recv_c 0
                                    }
             if { $r == 0 } {
                 $ns at $nown "Afficher $j ";       # Executé la procedure Afficher #
                 set initiator($j,0) $nown
                 $ns at $nown  "$p($initiator($j,1)) initiator_local"
                 $ns at $nown "$n($initiator($j,1)) color red"
                 $ns at [expr $nown + 0.05 ] "$n($initiator($j,1)) add-mark m purple hexagon"
                 $ns at [expr $nown + 0.15 ] "$n($initiator($j,1)) delete-mark m"
                 $ns at [expr $nown + 0.2 ] "$n($initiator($j,1)) color black"
                 for {set k 0} {$k <$nbr_I} {incr k} {
                                    $p($initiator($k,1)) set recv_c 1
                                    }
                } else {
             $ns at $nown "Afficher $j ";          # Executé la procedure Afficher #
             set initiator($j,0) $nown
             $ns at $nown  "$p($initiator($j,1)) initiator";
             $ns at $nown "$n($initiator($j,1)) color red"
             $ns at [expr $nown + 0.2 ] "$n($initiator($j,1)) color black"

                    }
             }

  } else {
   for {set k 0} {$k <$nbr_I} {incr k} {
                          $p($initiator($k,1)) set recv_c 0
                         }
```

```
    set initiator($j,5) 1
    if { $r == 0 } {
    $ns at $nown "Afficher $j ";                    # Executé la procedure Afficher #
    set initiator($j,0) $nown
    $ns at $nown  "$p($initiator($j,1)) initiator_local"
    $ns at $nown "$n($initiator($j,1)) color red"
    $ns at [expr $nown + 0.05 ] "$n($initiator($j,1)) add-mark m purple hexagon"
    $ns at [expr $nown + 0.15 ] "$n($initiator($j,1)) delete-mark m"
    $ns at [expr $nown + 0.2 ] "$n($initiator($j,1)) color black"

    for {set k 0} {$k <$nbr_I} {incr k} {
                                $p($initiator($k,1)) set recv_c 1
                                }
      } else {
      $ns at $nown "Afficher $j ";                    # Executé la procedure Afficher #
      set initiator($j,0) $nown
      $ns at $nown  "$p($initiator($j,1)) initiator";
      $ns at $nown "$n($initiator($j,1)) color red"
      $ns at [expr $nown + 0.2 ] "$n($initiator($j,1)) color black"


        }
       }
}
#*************************************************************************#
#                          L'accés au fichier scenario.txt                #
#*************************************************************************#
set s [open "scenario.txt" "r"]

set nbr [gets $s]
set date_fin [gets $s]
set nbr_msg [gets $s]
set nbr_I [gets $s]

for {set j 0} {$j <$nbr_I} {incr j} {
            set initiator($j,0) [gets $s]
                set initiator($j,1) [gets $s]
            set initiator($j,2) 0
            set initiator($j,3) 0
            set initiator($j,4) 0
            set initiator($j,5) 0
                        }
close $s
#*************************************************************************#
#                          Déclaration des couleurs                       #
#*************************************************************************#
$ns color 0 blue;                                  #  La couleur Bleu est message     #
$ns color 1 red;                                   #  La couleur Rouge est request    #
$ns color 2 limegreen;                             #  La couleur Vert est reply       #
$ns color 3 magenta;                               #  La couleur Violet est commit    #

#*************************************************************************#
#          Création des pocessus et leurs connections aux agents  CSNB    #
#*************************************************************************#
```

```
for {set i 0} {$i < $nbr} {incr i} {          #  Boucle  pour creer les nbr_proc pocessus      #
  set n($i) [$ns node];                       #  Création d'un noeud                           #
  set p($i) [new Agent/CSNB];                 #  Création d'un Agent CSNB                      #
  $p($i) set nbr_proc $nbr;                   #  Initialiser nbr_proc                          #
  $ns at 0.0 "$p($i) initial";                #  Executé la procedure initial                  #
  $ns attach-agent $n($i) $p($i);             #  L'attachement des noeuds aux agents CSNB      #
}
#*************************************************************************************#
#                                faire des liens avec tous les nœuds                  #
#*************************************************************************************#
for {set i 0} {$i <$nbr} {incr i} {
        for {set j [expr $i+1]} {$j <$nbr} {incr j} {
        $ns duplex-link $n($i) $n($j) 1Mb 10ms DropTail
        $ns connect $p($i) $p($j)
                                    }
                        }
#*************************************************************************************#
#                                L'accés au fichier messages.txt                      #
# ************************************************************************************#
set m [open "messages.txt" "r"]
for {set j 0} {$j <$nbr_msg} {incr j} {

            set date_ [gets $m]
          set src [gets $m]
              set dest [gets $m]

              $ns at $date_ "$p($src) set dst_addr_ $dest"
              $ns at $date_ "$p($src) send-message";  # Executé la procedure send-message #
                          }
close $m
#*************************************************************************************#
#                                lancer l' initialization                             #
#*************************************************************************************#
for {set j 0} {$j <$nbr_I} {incr j} {
                          $ns at $initiator($j,0) "lancer_initiator $j"
                        }

#*************************************************************************************#
#                Appeler le procedure Afficher_old_csn si nbr de processus <=16       #
#*************************************************************************************#
if { $nbr<=16 } {
$ns at 0.0 "Afficher_old_csn"
}
#*************************************************************************************#
#                Appel de procédure "finish" aprés un temps de simulation             #
#*************************************************************************************#
$ns at 1000 "finish"
#*************************************************************************************#
#                             lancer la simulation                                    #
#*************************************************************************************#
$ns run
#*************************************************************************************#
#                             Fin_CSNB.tcl                                            #
#*************************************************************************************#
```