الجمـــهوريـــة الجـــزائريـــة الديـمقـراطيـــــــة الشـــعبيـــــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعـلـيـم الـعـالـــي و الـبـحـث الـعـلـمـي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
جـامـعـة عمّـــار ثـلـيـــجـــي بالأغـــواط
UNIVERSITE AMAR TELIDJI LAGHOUAT
كـلـيـــــة الـعـلـــــوم
FACULTE DES SCIENCES
DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE

Mémoire de MASTER

Domaine : Mathématiques et Informatique
Filière : Informatiques
Option : Réseaux, Systèmes et Applications Réparties

Par:
NAIDJATE KHEDIDJA

THEME

Evaluation des performances des protocoles CIC dans un environnement défaillant

---

Soutenu publiquement devant le jury composé de:

Mme TAABA	***M.C.(B)***	***Présidente***
M^{elle} F.BOUSBAA	***M.A.(A)***	***Examinatrice***
M^{elle} A.BELABACI	***M.C.(A)***	***Examinatrice***
Mlle ABDELHAFIDI Zohra	***M.A.(A)***	***Encadreur***

Année Universitaire 2015/2016

Remerciment

je remercie tout d'abord ALLAH , le tout puissant de m'avoir donné la force et la patience
et de m'avoir rapprocher des personnes qui m'ont soutenu et aidé pour accomplir ce
travail.

Je tiens à exprimer ma sincère reconnaissance et remerciements à
Mlle Abdelhafidi Zohra
d'avoir accepté d'encadrer et de diriger mes travaux et de m'avoir fait bénéficier de ses compétence, sa disponibilté, et ses conseils.

Mes remerciements s'adressent également à toutes les personnes qui ont contribué de
près ou de loin avec leurs conseils ou avec leurs encouragements à l'accomplissement de
ce travail .

Enfin, j'exprime mes vifs remerciements à toute ma famille et spécialement à mes deux sœurs Nada et Nesrine.

Merci a tous

dédicace

Je dédie ce mémoire à :

ma mére (que dieu ait son ame) et mon pére qui peut être fier et trouver ici le résultat de longues années de sacrifices et de privations pour m'aider à avancer dans la vie.

Mes soeurs (Nada et Nesrine) qui n'ont cessé d'être pour moi des exemples de persévérance, de courage et de générosité.

Et à toutes les personnes qui m'on soutenu et surtout mes amies

Table des matières

Table des figures :

RÉSUMÉ

Dans les systèmes répartis plusieurs problèmes peuvent se passer et risque de faire perdre des informations, la tolérance aux pannes permet de réduire la perte d'information par plusieurs mécanisme, parmi ces mécanismes c'est l'utilisation des points de reprises.

Assurer le recouvrement sur erreur sans risque de l'effet domino nécessaitera. de revenir en arrière si c'est possible au dernier point de reprise cohérent. A cet égard, trois classe de points de reprise ont été proposé : coordonée, incoordonée et induite par les communications (CIC).

Dans ce mémoire, nous nous s'intéressons à l'étude des algorithmes de point de reprise avec recouvrement pour le cas de plusieurs fautes simultanés. Pour cela nous avons implémenté et simulé deux algorithmes de type CIC MS et HMR avec l'outil NS-2.Des scenarios sont fait par la combinaison de différents paramètres, pour cela nous avons choisit comme métrique de performance le nombre de point de reprise forcé, le nombre de messages enregistrer, la distance de recouvrement et le nombre de points de reprise supprimés.

Mots clé : tolérance aux pannes recouvrement arrière, système réparti, NS-2, simulation.

ABSTRACT

In distributed systems several problems can happen and risk of losing information, fault tolerance can reduce the loss of information by several mechanisms, among these mechanisms is the use of checkpointing.

Ensure recovery of error without risk of domino effect requires to go back if possible to the last consistent global checkpoint. In this regard, three classes checkpointing have been proposed: coordinated,uncoordinatedand communication induced (CIC).

In the present maniscript, we are interested in studying checkpointing algorithms with recovery in the event of several simultaneous faults. For this we have implemented and simulated two CIC algorithms MS and HMR in NS-2 tool. The scenarios are made by combining different parameters, for this we have chosen as a performance metric the number of forced checkpoint, the number of messages recorded, the rollback distance and the number of deleted checkpoints.

Keywords : checkpoint, distributed system, fault tolérance, NS-2 , rollback, simulation

INTRODUCTION GÉNÉRALE

Depuis une dizaine d'années, l'homme a créé et utilisé des outils l'aidant à calculer, communiquer, et exécuter certains programmes, tout cela a connu une évolution passant par la machine la plus simple à la plus développée.

La fiabilité des composants avec lesquels étaient construites les machines informatiques n'était pas assurée, car ces machines n'étaient pas capables d'exécuter un programme de sans être victime d'une panne. Et afin de réduire ces anomalies, des techniques ont été établies notamment nommées, les **points de reprises et le recouvrement** dont le rôle est de faire face à des défaillances de processus dans un système distribué.

Lors d'une panne (ou plusieurs), les protocoles à base de points de reprise restaurent l'état global cohérent du système. Cet état est l'ensemble points de reprise locaux, un par processus. La determination du point de reprise global est faite par l'une des trois methodes : coordonnée, non-coordonnée et induite par les communications.

Nous nous intéressons aux algorithmes de la classe CIC. Pour cela, nous étudions et simulons les deux algorithmes (MS et HMR) en analysant leurs performances via différents scénarios dans un environnement défaillant.

Ce mémoire est composé de trois chapitres et deux annexes :

- ✓ dans le premier chapitre, nous allons présenter des généralités sur les systèmes répartis , la tolérance aux pannes et les classes de points de reprise et le recouvrement arrière.
- ✓ Dans le deuxième chapitre, nous allons donner une description des deux algorithmes simulés (MS, HMR) (l'algorithme, description, exemples, explication des exemples).
- ✓ Le troisième chapitre regroupe les étapes de simulation, l'outil de simulation et l'analyse des résultats.
- ✓ À la fin de ce mémoire, on termine par une conclusion générale.
- ✓ Les annexes sont représentées de la manière suivante :
- ✓ un exemple de code source (exemple du protocole MS cas de plusieurs fautes).
- ✓ Le mode d'utilisation pour voir la réalisation de la simulation.

Chapitre 1

Systèmes répartis et la tolérance aux pannes
Généralités et définitions

1.1 introduction

La communication par message passe généralement par au moins deux primitives : envoi(message, processus) et réception(message). Ce modèle est utilisé par de très nombreuses applications de calcul scientifique et les systèmes distribués. C'est pourquoi, il est particulièrement intéressant de permettre à ces applications d'intégrer de manière transparente un mécanisme de reprise en cas de défaillance, dans ce chapitre on va définir de manière générale les systèmes répartis et la tolérance aux pannes.

1.2 Définition :

Un système réparti est un ensemble des processus (sites) indépendants et interconnectées qui peuvent communiquer entre eux, il apparait à un utilisateur comme un système unique et cohérant, un calcul distribué est constitué de N processus séquentiels désignés par P0, P1, P2…Pn en cours d'exécution en même temps sur un ensemble d'ordinateurs dans le réseau.

1.2.1Les deux principaux modèles d'un système répartis :

- **Un modèle synchrone** est un modèle où les contraintes temporelles sont bornées
  - On sait qu'un processus évoluera dans un temps borné
  - On sait qu'un message arrivera en un certain délai
  - On connaît la limite de dérive des horloges locales
- **Un modèle asynchrone** n'offre aucune borne temporelle

1.3 La tolérance aux pannes

1.3.1 Définition de la tolérance aux pannes

La tolérance aux pannes (on dit également « insensibilité aux pannes ») désigne une méthode de conception permettant à un système de continuer à fonctionner, éventuellement de manière réduite au lieu de tomber complètement en panne, lorsque l'un de ses composants ne fonctionne plus correctement[1].

Il se peut que dans un système qui fonctionne correctement, d'avoir à un moment donné un problème qui va influencer négativement sur son comportement, c'est le problème de panne d'un site dans le système. Il existe plusieurs types de solutions permettant de résoudre ce problème.

Les solutions suivantes sont plus adéquates à un système centralise qu'a un système reparti.

- ✓ La prévention des fautes : qui s'attache aux moyens permettant d'éviter l'occurrence de fautes dans le système.
- ✓ L'élimination des fautes : qui se focalise sur les techniques permettant de réduire la présence de fautes ou leurs impacts.

- ✓ Cependant dans un système reparti, ils existent plusieurs sites qui sont reliées par un réseau de communication, cela va permettre à un site de prendre les tâches d'un autre site qui a subit une faute ou une panne : La tolérance aux pannes.

1.3.2 Modes de pannes :

- **pannes franches (fail stop):** soit le système fonctionne normalement (les résultats sont correctes), soit il ne fait rien. Il s'agit du modèle de panne le plus simple auquel on essaie de se ramener chaque fois que cela est possible.
- **pannes par omission :** des messages sont perdus en entrée et/ou en sortie. Ce type de panne est utilisé pour les réseaux.
- **pannes par valeurs :** les résultats produits par un composant défaillant sont incorrects.
- **pannes byzantines :** le système peut faire n'importe quoi, y compris avoir un comportement malveillant.
- **panne temporelle :**le temps de réponse du système dépasse les exigences des spécifications.

1.3.3 Classes de tolérance aux pannes :

La tolérance aux pannes permet l'assurance de la sûreté de fonctionnement, il existe deux classes de tolérance :

- La tolérance aux pannes par mémoire stable
- La tolérance aux pannes par duplication.

2.3.4 La tolérance aux pannes par duplication :

L'approche de tolérance aux pannes par duplication consiste à la création de plusieurs copies des processus sur différents nœuds de calcul, elle est réalisée par masquage d'erreur, la panne est masquée par le comportement de l'une des copies non défaillantes. Deux techniques différentes sont proposées pour implémenter cette approche la duplication est :

de nature active : elle est définie par symétrie du comportement des copies d'un composant dupliqué où chaque copie joue un rôle identique a celui des autres.

de nature passive :où seulement la copie primaire exécute l'application tandis que les autres copies sont en état passif et elles ne font que se synchroniser avec la copie primaire. Ainsi en cas de panne une copie secondaire prend la place de la copie primaire toutefois la tolérance aux pannes via duplication est une approche relativement couteuse en terme de ressource de calcul.

1. 3.5 Tolérance aux pannes par mémoire stable :

Une mémoire stable est considérée comme étant un support persistant de stockage, dont le rôle principal est d'assurer une accessibilité et une protection des données contre les pannes pouvant affecter le système. Ainsi, suite à une panne, un état correct ayant été stocké antérieurement à cette panne sur la mémoire stable reste accessible, cela permet au système un retour à un état antérieur[10]. Il y ‘a deux techniques

1. Tolérance aux pannes par journalisation
2. Tolérance aux pannes à base de points de reprise

1.4 Tolérance aux pannes par journalisation :

Le principe de base des approches par journalisation est de sauvegarder en mémoire stable les messages qui transitent entre les processus, et occasionnellement et de façon totalement indépendante, prendre un point de reprise et le sauvegarder en mémoire stable.

On peut distinguer trois approches différentes dans la tolérance aux pannes par journalisation: pessimiste, optimiste et causale.

1.4.1 Journalisation pessimiste :

La journalisation pessimiste est une approche qui enregistre sur une mémoire stable de façon synchrone tous les messages en transit dans le système.

Tout message, une fois arrivé dans le contexte du récepteur est forcément enregistré, et pourra donc être rejoué en cas de panne : ainsi, tous les points de reprise du processus sont utilisables pour une reprise, puisque tous les événements non déterministes ont été enregistrés depuis la prise de ce point. L’ordre de ces messages doit lui aussi être conservé, de manière à recréer lors de la reprise la même histoire de message.

L’avantage de cette approche est qu’elle ne crée jamais de processus orphelin (c’est un processus dont le message qui lui est envoyé a été perdu). Mais son inconvénient est qu’elle bloque le processus avant chaque retrait de message[10].

1.4.2 Journalisation optimiste :

Dans le cas de la journalisation optimiste, les messages reçus ne sontpas forcément enregistrés immédiatement sur une mémoire stable afin d’améliorerles performances. L’enregistrement des messages en transit dans le système sefait de façon asynchrone, de manière à grouper les enregistrements sur mémoirestable qui représentent un surcoût considérable. Les messages sont conservés en mémoire volatile avant un enregistrement sur mémoire stable dont le momentdépend du protocole. Ces protocoles font donc l’hypothèse

optimiste que ces enregistrementsseront terminés avant qu'une panne n'arrive, de façon à ce que tousles événements puissent être rejoués en cas de panne d'un processus[9].

La figure 1.1 enregistre le déterminant de chaque message dans une mémoire stable.

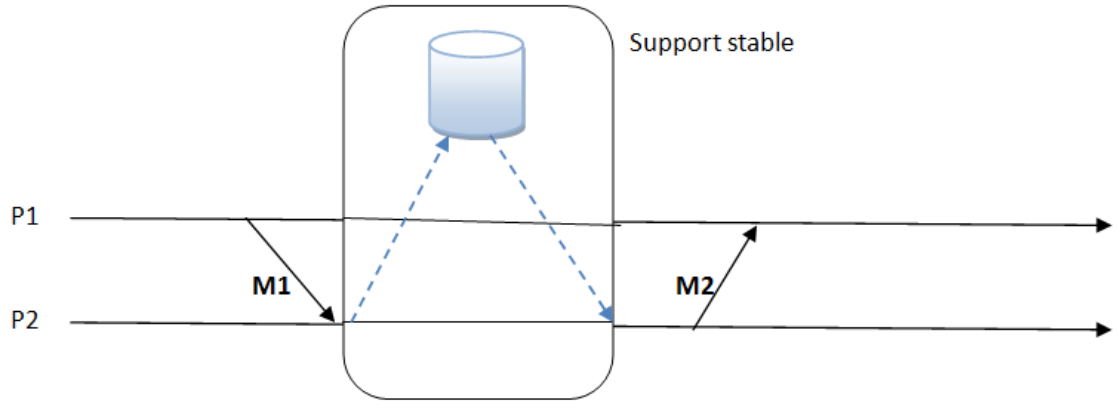


FIGURE 1. 1 Processus de sauvegarde avec la journalisation pessimiste

1.4.3Journalisation causale :

Le principe de la journalisation causale est d'assurer que le déterminant de chaque événement non déterministe qui précède causalement l'état d'un processus se trouve soit en mémoire stable, soit est disponible localement pour ce processus.
La journalisation causale a l'avantage des deux méthodes précédentes en terme de performances à l'exécution et en cas de reprise. Comme la journalisation optimiste, la journalisation causale évite d'une part, la synchronisation avec la mémoire stable à l'exception du moment de la publication de résultats. D'autre part, comme dans la journalisation pessimiste, la journalisation causale ne crée jamais de processus orphelin. Cela permet la reprise de n'importe quel processus défaillant à partir de son dernier état sauvegardé sans risquer l'effet domino. L'inconvénient de cette technique réside en la complexité du protocole de recouvrement arrière suite à une panne[1].

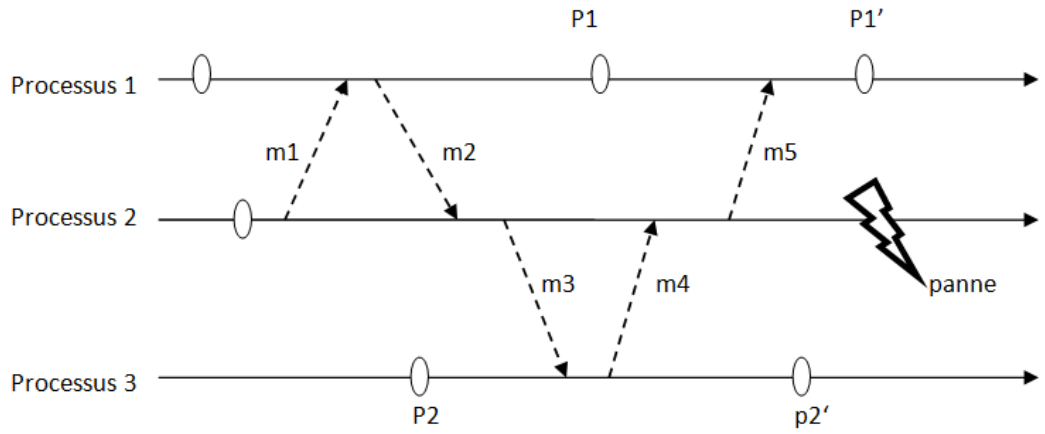


FIGURE 1. 2 journalisation causale

La FIGURE 1. 2 journalisation causale ilustre un recouvrement en utilisant la journalisation causale. Le processus 2 subit une défaillance. Il va redémarrer à partir de son dernier point de reprise et rejouer tous les messages reçus, comme par exemple le message m4 dont le déterminant est contenu dans le message m5 grâce au protocole de journalisation causale qui attache les déterminants des messages sur les autres messages circulant dans le système.

1.5 Recouvrement arrière à base de points de reprise :

1.5.1 Définition d'un Point de reprise :c'est une technique établie pour faire face à des défaillances de processus dans un système distribué. Lorsqu'un processus échoue, leprotocole utilise les points de reprises et la journalisation pour restaurer le système à un uniforme Etat. Il existe plusieurs algorithmes de points de reprise qui sont proposés pour les systèmes distribués.

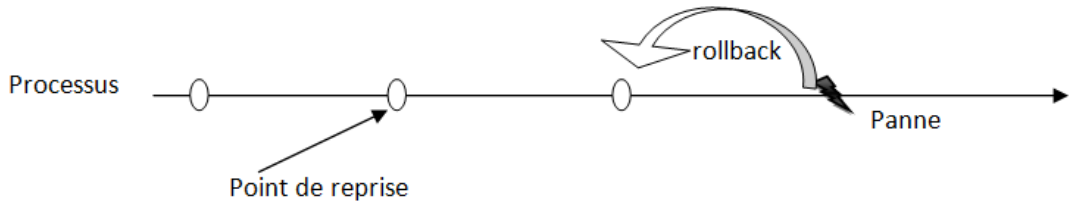


FIGURE 1. 3 Recouvrement arrière après panne

Point de reprise locale :

Un point de reprise locale (ou un état locale) est défini comme un enregistrement complet de l'état de processus. Cet enregistrement sera utilisé en cas de panne pour éviter de revenir au début de l'exécution.

Point de reprise globale :

Un point de reprise globale (ou état globale) d'un système réparti est un ensemble de points de reprises locaux par un processus.

Ce point de reprise peut être cohérant ou incohérent, un point de reprise globale est cohérent si pour chaque message (m) sa réception est enregistrer dans le point de reprise globale, et son émission est aussi enregistrée, ce point de reprise reste cohérent même si un message a été envoyer mais non reçu.

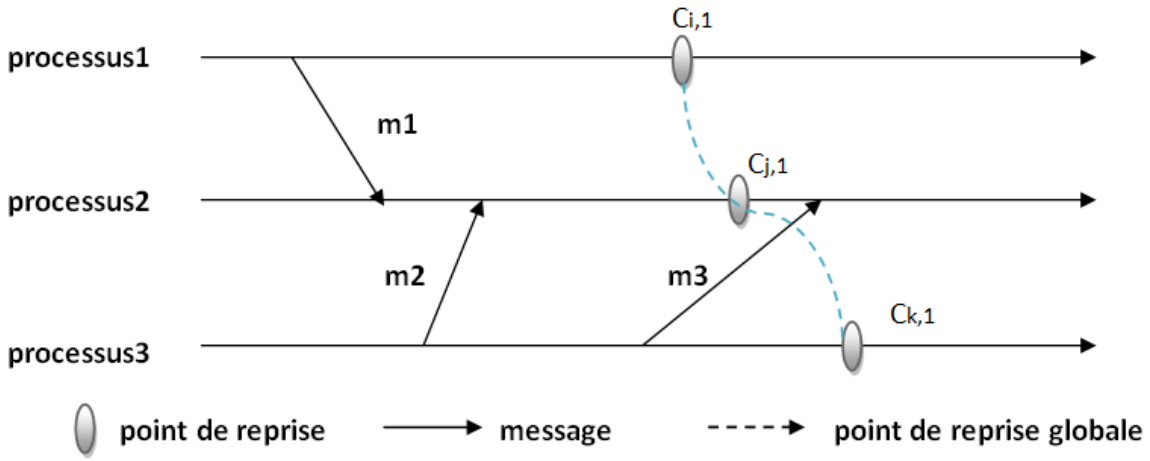


FIGURE 1. 4 un point de reprise globale cohérent

un message orphelin : est un message qui à été envoyé après un point de reprise appartenant à un état globale et reçu avant un point de reprise appartenant à cet état.

La FIGURE1.5 représente un point de reprise global incohérent ($C_{i,1}$,$C_{j,1}$,$C_{k,1}$), le message m_3 est orphelin

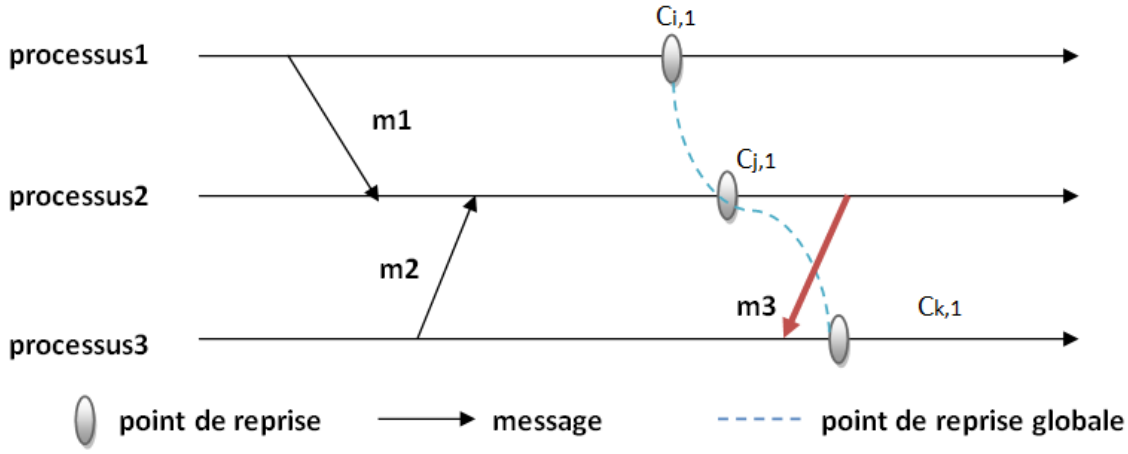


FIGURE 1. 5 un point de reprise globale incohérent

Un message en transit : c'est un message qui a été envoyer avant que le processus établi un point de reprise et sa réception est après avoir établi le point de reprise globale.

1.5.4 Z-chemins :

Définition : un intervalle (noté $I_{i,x}$) est définit par une séquence des évenement produits par un proceesus P_i entre deux points de reprise successifs $C_{i,x-1}$ et $C_{i,x}$ un intervalle $I_{i,x}$ peut etre vu comme un macro-évenement qui fait passer le processus P_i du point de reprise $C_{i,x-1}$ au point de reprise $C_{i,x}$[3].

Un Z-chemin est une séquence de messages particuliére {m1,m2…..} qui connecte deux points de reprises $C_{i,x}$ et $C_{j,y}$, tel que l'évenement émission du message m_i produit par un processus qui appartient au méme intervalle ou à l'intervalle suivant qui contient la reception du message m_{i+1}.

1.5.3 Z-Dépendance :

un point de reprise locale $C_{i,x}$z-dépend d'un autre point de reprise $C_{j,y}$($C_{i,x} \overset{z}{\rightarrow} C_{j,y}$), soit parce que tous les deux sont du même processus (i=j) et $C_{i,x}$ précéde $C_{j,y}$ ou parce qu'il existe un z-chemin de $C_{i,x}$à $C_{j,y}$[3].

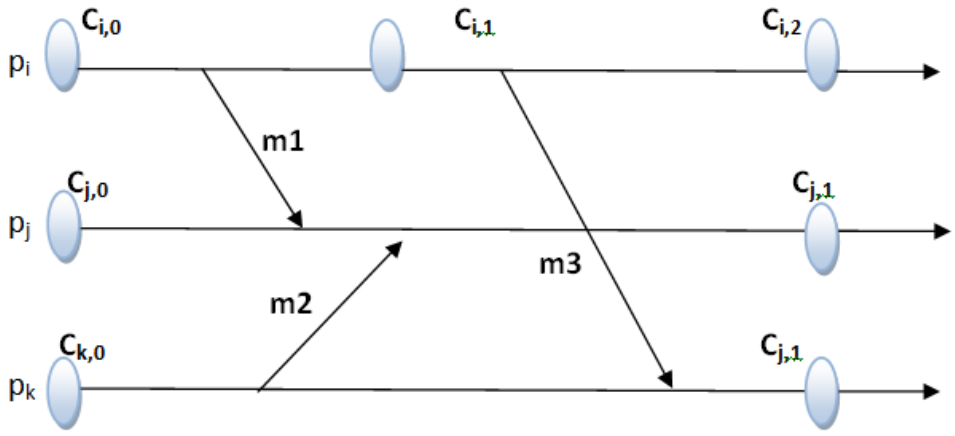


FIGURE 1. 6 Depandance (exécution répartie)

un Z-cycle est un Z-dépendance du point de reprise locale $C_{i,x}$ à lui-même $C_{i,x}$, un Z-cycle est dit inutile, car quelque soit le point de reprise globale contenant $C_{i,x}$, ce point est toujours incohérent[3].

Definition un point de reprise globale G est cohérent si et seulement si :

$$\forall Ci,x\ , Cj,y \in G: \neg(Ci,x \overset{z}{\rightarrow} Cj,y\ )$$

1.6 Protocoles de recouvrement arrière à base de points de reprise :

Lors d'une défaillance, les protocoles à base de points de reprise restaurent l'état global du système à partir de points de reprise locaux de chacun des processus formant l'état global cohérent le plus récent appelé ligne de recouvrement.

Pour former cette ligne de recouvrement, trois stratégies sont possibles : points de reprise non coordonnés, coordonnés, ou induits par les communications.

1.6.1 Points de reprise non coordonnés :

La technique de sauvegarde non coordonnée de points de reprise vise à maximiser les performances en fonctionnement normal. L'objectif est de minimiser le surcoût lié à la sauvegarde des points de reprise lors d'une exécution sans fautes.

Chaque processus sauvegarde de manière indépendante son état local dans un point de reprise, il en conserve plusieurs. Durant l'exécution, les messages sont accompagnés d'une estampille temporelle qui permet de percevoir des dépendances causales entre les états des différents processus[2].

Lors de la reprise, la ligne de recouvrement est calculée à l'aide des estampilles temporelles. Le processus défaillant retourne à l'un de ses points de reprises. Les autres processus peuvent ou non être amenés à faire un recouvrement arrière, en fonction des dépendances causales.

Cette technique a comme principal avantage que chaque processus sauvegarde son point de reprise quand cela est le plus avantageux pour lui-même. Ainsi, il est possible pour un processus de faire cette opération quand la taille des informations sur son état est petite, minimisant ainsi la taille du point de reprise et limitant la dégradation des performances lors d'une exécution sans fautes. Le principal inconvinient et le risque de l'effet domino qui est definit comme suit

Effet domino :

Le problème de l'effet domino apparait 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 globale le plus récent.

A cause d'un message orphelin, chaque point de reprise globale choisi reste incohérent et le système va redémarrer à partir du point de reprise initiale (début d'exécution).

1.6.2 Points de reprise coordonnés :

La technique de sauvegarde coordonnée des points de reprise vise la simplicité de mise en oeuvre et l'assurance d'obtenir un état global cohérent lors de la sauvegarde.

Cette approche repose sur une coordination globale de tous les processus de l'application. Un processus, le coordinateur (désigné dynamiquement ou statiquement) sauvegarde son point de reprise et diffuse un message de recouvrement demandant à tous les autres processus de l'application d'établir leur point de reprise. Quand un processus reçoit le message, il stoppe son exécution, vide ses canaux de communication, prends un point de reprise provisoire, et envoie un message d'acquittement au coordinateur. Une fois que le coordinateur a reçu l'ensemble des acquittements, celui-ci diffuse un message de validation du point de reprise. Chaque processus remplace alors, de manière atomique, son ancien point de reprise permanent par le point de reprise provisoire.[2]

Lors de la reprise, il suffit de restaurer l'ensemble des processus de l'application à partir de leur point de reprise respectif. Cette technique assure que l'ensemble des points de reprise forme un état global cohérent et évite ainsi l'effet domino. De plus, la reprise est très simple. Un autre avantage de cette technique est que l'espace de stockage nécessaire pour conserver les points de reprise est minimisé puisqu'il n'est nécessaire que d'en conserver un seul par processus.

1.6.3 Points de reprise induits par les communications :

La technique des points de reprise induits par les communications à pour idée d'initier les pointsde reprise en fonction des messages émis et reçus entre les processus. Aucun message spécifique decoordination n'est envoyé, mais les points de reprise sont réalisés lors d'évènements particuliers.

Dans chaque message de l'application, des informations du protocole sont encapsulées. À partir deces informations encapsulées qui contiennent des estampilles temporelles sur les messages envoyés et les points de reprises créés, le processus récepteur du message décide s'il doit ou non créer un point de reprise. Il faut noter que certains points de reprise peuvent également être pris de manière indépendante mais ces derniers ne garantissent pas la progression de la ligne de recouvrement.La reprise se passe de la même manière que pour la stratégie des points de reprise non coordonnés.

Les techniques de sauvegarde de points de reprise induits par les communications évitent l'effet domino tout en ne nécessitant pas la coordination de tous les processus de l'application[2].

Conclusion :

L'objectif de la tolérance aux fautes est de permettre au système de continuer à fournir le servicemalgré l'occurrence des fautes.
Dans ce chapitre Nous avons présenté les concepts générales des systèmes répartis et la définition de la tolérance aux pannes, la journalisation …etc.

Chapitre 2

Les algorithmes simulés

2.1 Introduction :

Un point de reprise d'un processus impliqué dans un calcul distribué est dit être utile s'il fait partie d'un point de reprise globale cohérent. Dans ce chapitre, nous présentons des algorithmes quasi-synchrones qui font de chaque point de reprise une utilité précise.

2.2 Algorithmes de points de reprise:

2.2.1 Protocole MS

Le protocole MS est un algotrithme Quasi-Synchrone (CIC) proposé par Manivannan et singhal en 1996, cet algorithme est gerer de la maniére suivante [5]:

- ✓ Chaque processus (Pi) dispose d'une variable $next_i$ etsn_i pour garderla trace du nombre actuel de point de reprise de processus[6].

- ✓ Mais avant de prendre ce point il faut incrémenter $next_i$ d'une unité de temps et associé cette valeur à sn_i, En outre, les processus sont forcés de prendre des points de reprise lors de la réception de certains messages.
- ✓ Quand un processus P_i décide d'envoyer un message M, il superpose le numero de squence sn sur le message noté M.sn.
- ✓ Chaque processus est autorisé à prendre des points de reprise spontanée (basic checkpoint) de manière asynchrone.
  Si (next > sn), il est possible qu'un processus P_i prend un point de reprise spontanée.
  - si la condition next > sn n'est pas vérifiée, l'ajout d'un autre point de reprise spontané est annulé à cause de la présence d'un point de reprise forcé enregistré.
- ✓ Lors de la réception d'un message, chaque processus en état normale vérifie les deux valeurs de M.sn et sn
  - Si M.sn > sn, alors le processus prend un point de reprise supplémentaire (forced checkpoint) et poursuit le message.

2.2.1.1 structures de données

Sn_i := 0 ; // numéro de séquence de point de reprise, initialisé a 0

$Next_i$:= 1 ; // numéro de séquence destiné au prochain point de reprise initialisé a 1

2.2.1.2 L'algorithme :

Quand un processus P_i décide d'incrementer « $next_i$» :

$Next_i := next_i + 1$;

Quand un processus P_i prend un point de reprise spontané :

Si ($next_i > sn_i$) alors
//prendre un point de reprise C
C.sn := $next_i$; //C.sn reçoit $next_i$ comme numéro de séquence
Sn_i :=C.sn ; // le sn_i est mis à jour
Finsi

Quand un processus P_i envoi un message M :

M.sn := sn ; //numéro de séquence du point de reprise courant supérposé avec M
Envoyer (M) ;

Quand un processus P_j reçoit un message M d'un processus P_i :

si (**M.sn > sn_j**) alors
//prendre un point de reprise forcé C
C.sn := M.sn ;
Sn_j := C.sn ;
Sinon
// poursuivre le message
finsi

et pour comprendre mieux l'algorithme MS on va voir un éxemple expliquat toutes les étapes de cet algorithme

2.2.1.3 Exemple :

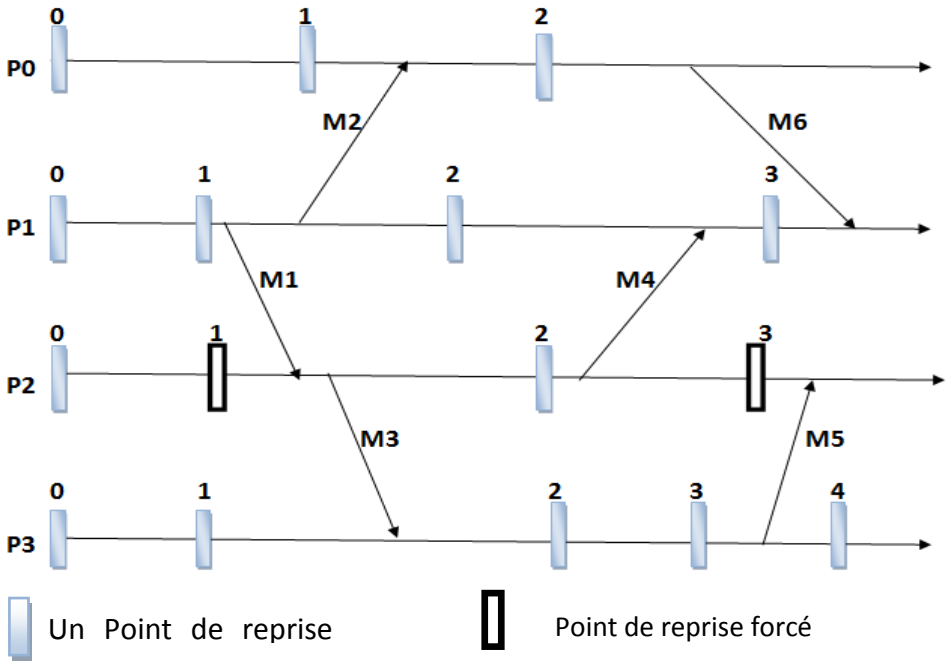


Figure2. 1 exemple du protocole MS

Soit un système composé de quatre processus (Figure2. 1 exemple du protocole MS), chaque processus incrémente la valeur de next périodiquement.

Lors de la reception du message **M1, P2** est forcé de prendre un point de reprise supplémantaire (forced checkpoint) avec le numéro de séquence 1 avant le traitement de **M1** parce que **M1.sn** =**1** et $\mathbf{Sn_2}$(=0) < 1. De même, P2 est forcéde prendre un point de reprise (3) avant de traiter le message**M5**.

2.2.2 Protocole HMR:

2.2.2.2 Description générale du protocole :

Le protocole HMR repose sur un mécanisme d'estampillage. Chaque point de reprise local Ci,x est associé un estampille de Lamport, et chaque Message m porte un nombre entier d'estampille temporelle (m.t)[7].

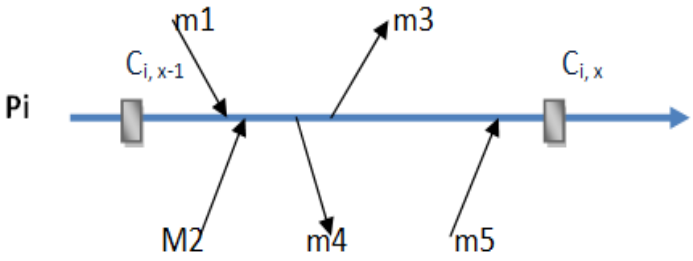


Figure2. 2 deux points de reprises locaux successives

Variables et structures de données

Lc_i est l'estampille de message

Min_com //entier initialisé a +∞ qui va contenir le minimum du numero de sequence des messages

Ce protocole est définit par trois procédures suivantes :

Prendre un point de reprise local :

Dans cette la procédure, la variable lc_i est incrémenté, cette procédure est appelé par un processus $P_i$quand il veut prendrespontané ou quand ilest dirigé par le protocole de prendre un point de reprise forcé.

Envoyer un message :

Cette procédure est exécutée par le protocole à chaque fois qu'un processusPi envoie un message m. Le message m et estampillé par lci noté par m.t

procédure réception d'un message:

Cette procédure est exécutée quand un message m arrive au niveau de processus P_i elle maintient la validité des conditions suivantes :

- ✓ **m.t <lc_i** : si cette condition est vérifié le message m doit être enregistré.
- ✓ **m.t =lc_i** $MIN\_COM_i := \min(MIN\_COM_i, m.t)$.
- ✓ **m.t >lc_i :**si (**m.t> MIN_COM**$_i$) alors Pil va prendre point de reprise forcé

2.2.2.4 l'algorithme:

Quand un processus décide de prendre un point de reprise)

Définir le point de reprise locale (courant) $C_{i,x}$;

$C_{i,t} := lc_i$;

x := x+1 ;//X est considéré comme un compteur initialisé à 0

Min_COM := +∞ ;

$lc_i := lc_i + 1$;

Quand un processus décide ded'envoyer un message :

Soit M le message courant envoyer par P_i ;

m.t :=lc_i ;

$MIN\text{-}COM_i := \min(MIN\text{-}COM_i, m.t)$;

envoyer le message M ;

Quand un processus reçois un message par p_i:

Soit M le message reçu par P_i ;

Cas m.t <lc_i :enregistrer m ;

Cas m.t =lc_i :$MIN\_COM_i := \min(MIN\_COM_i, m.t)$

Cas m.t >lc_i : **si** (m.t> MIN_COM_i) alors

prendre point de reprise

finsi

$MIN\_COM_i := (MIN\_COM_i, m.t)$;

Finde cas ;

Délivrer le message m ;

Exemple

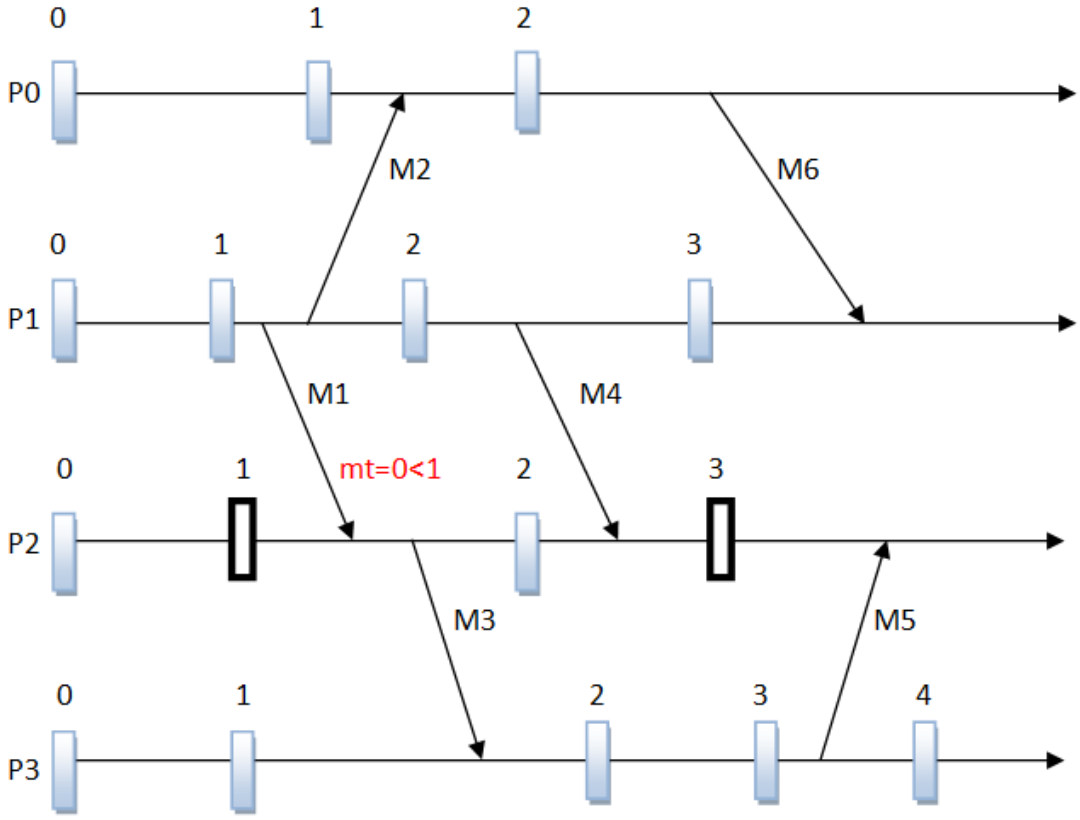


Figure2. 3 exemple du protocole HMR

on reprend le meme exemple du MS où on à les trois cas de réceptions de messages :

Cas1 : le processus P1 décide d'envoyer un message M1 à P2 il va faire :

M1.t =1 et MIN-COM$_2$:= min(MIN-COM$_2$, M1.t$_2$)=1.

Quand p2 reçoit M1, il va trouver que M1.t > lc$_2$ (1>0) et donc il va prendre un point de reprise forcé le même cas pour les messages M1.

Cas2 : le processus P1 décide d'envoyer un message M4 à p2 il va faire :

M4.t=lc$_1$ et MIN-COM$_2$:= min(MIN-COM$_2$, m.t$_2$)=1

Quand p2 M4 il va trouver que M4.t=lc$_2$ (2=2) et donc il va prendre

MIN-COM$_1$=min(MIN_COM$_1$, M4.t) la même chose pour le message M3.

Cas3 :le processus p0 décide d'envoyer un message M6 à p1 il va faire :

M6.t=lc$_0$ dans notre ca M6.t =2

Quand M6 il va trouver que M6.t <lc_0 (2<3) et donc il va enregistrer le message M6

2.3 Algorithmes de recouvrement:

2.3.1 Cas d'une seule faute :

Les hypothèses de cet algorithme sont :

- la panne traitée est de type arrêt définitif (fail-stop fault).
- un seul processus tombe en panne dans le système.
- lors de recouvrement, les processus ne bloquent leurs exécutions.

2.3.1.1 Les états d'un processus :

Un processus peut avoir trois états (normale, en panne, en recouvrement) :

État normale : un processus peut envoyer et recevoir des messages simple aux autres, et recevoir des messages de recouvrement ou un message qui indique qu'un autre processus est en panne. La réception de ces messages met le processus en état de recouvrement.

En cas de panne : le processus devient en état de recouvrement.

Fin de recouvrement : le processus revient a son état initiale donc en état normale et peut continuer son exécution.

2.3.2 Description du mécanisme de recouvrement :

1. chaque processus (P_p) dispose d'une variable inc_i pour garder le nombre de panne pour chaque processus, et d'une variable rec_line_i pour garder le numéro de la ligne de recouvrement.
2. Un processus ayant subit une panne envoie des messages de recouvrement à tous les autres processus et devient en état de recouvrement.
    - ❖ Quand un processus reçoit un message normal, il doit tester la condition (**M.inc <inc_p**) pour verifier si un recouvrement est en cours ou non .
    - ❖ Lors de la réception d'un message de recouvrement le processus vérifie s'il ya une autre panne avec cette condition (**M.inc>inc**), si c'est le cas le processus exécute la procédure de recouvrement, sinon le message est ignorer.

- Pendant le recouvrement, un point de reprise forcé est pris si la condition (**rec_line$_p$>sn$_p$**) est valide sinon il restore un point de reprise déjà sauvegardé.

Remarque : on adapte le même mécanisme de recouvrement pour les algorithmes MS et HMR.

Notation :

C.sn: numéro de séquence du point de reprise C.
C.mesglog: Message log associé avec le point de reprise C.
M.sn: numéro de séquence du dernier point de reprise (piggybacked) avec le message M.
M.inc: numéro d'incarnation du dernier recouvrement avec le message M.
M.rec line: numéro de la dernière ligne de recouvrement.
Cur chkptp: le point de reprise courant d'un processus P_p.
Tous les messages sont des messages d'application, à l'exception du message de restauration qui est unmessage de commande envoyé par un processus qui a échoué.

2.3.2.1 structure de donnée à P_p :

Sn_p : entier (:=0) ; //numéro de séquence du point de reprise courant, initialisé à 0
$Next_p$:entier (:=1) ; // numéro de séquence associé au prochain point de reprise initialisé à 1
Inc_p : entier (:=0) ; // numéro de séquence de la courante incarnation,initialisé à 0
Rec_line_p :entier(:=0) // la dérniére line de recouvrement initialisé à 0.

2.3.2.2 L'algorithme :

Les deux premières procédures de l'algorithme précédant sont les mêmes dans cet algorithme

Début

Quand un processus P_p décide d'envoyer un message M :

M.sn := sn_p ;
M.rec_line := rec_line_p ;
M.inc := inc_p ;
Envoyer (M) ;

Quand un processus P_p reçoit un message M :

Si (M.inc <inc$_p$) alors
Si (**M.sn**<recnum(**M.inc**+1)) alors

Poursuive le message M ;

Sinon

Annuler le message M ;

Sinon Si (M.sn = inc_p) alors

Si(M.sn <sn_p) alors

Sinon si (M.sn **>**sn_p) alors

Prendre un point de reprise C ;

C.sn := M.sn ;

Sn_p := C.sn ;

Poursuivre M ;

Sinon si (M.inc>inc_p) alors

Rec_line_p := M.rec_line ;

Inc_p :=M.inc ;

Recouvrement de **(p) ;**

Replay_logged_message(**p**) ;

Traiter M comme si qu'il vient d'arrivé ;

Reprendre l'exécution;

Finsi

Apres l'echec du processus P_p :

Revenir au dernier point sauvegardé

Inc_p := inc_p +1 ;

Rec_line_p := Sn_p ;

Envoyer un recouvrement (inc_p ,rec_line_p) à tous les autres processus

Reprendre l'exécution;

Quand un processus P_q reçoit un message de recouvrement (inc_p, $rec\ line_p$) d'un processus P_p :

Si (inc_q>inc_p) alors

inc_p := inc_q ;

rec_line_p := rec_line_q ;

Recouvrement de (p) ;

Replay_logged_message(p) ;

//continuer normale

Sinon

Ingnorer le message de rollback

Finsi

la procédure de recouvrement (p :entier):

Si ($rec\_line_p > sn_p$) alors
//prendre un point de reprise
$C.sn := rec\_line_p$;
$Sn_p := C.sn$;
Sinon
// trouver le point de reprise le plus proche de C avec $C.sn \geq rec\_line_p$
$Sn_p := C.sn$;
$Cur\_chkpt_p := C$;
//restorer le point de reprise C
// supprimer tous les points de reprise après C
Finsi

2.3.2.4 Exemple

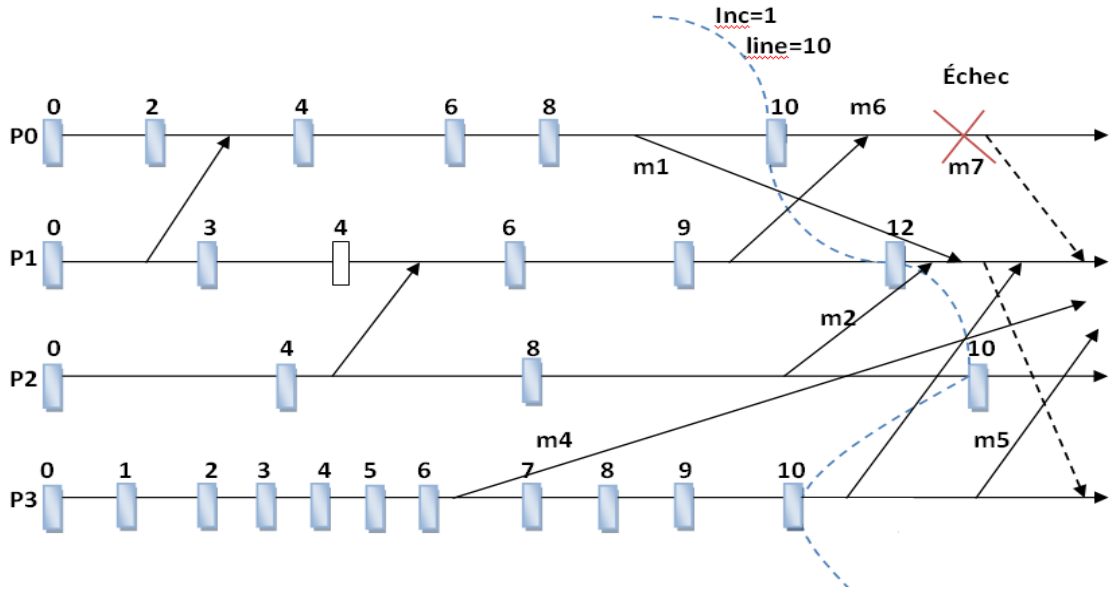


Figure2. 4 traitement des messages pendant le recouvrement arrière

Chaque processeur (P_i) incrémente sa valeur de $next_i$ à chaque unité de temps x :

- Le processus p0 prend un point de reprise pour toutes les 2 unités de temps.
- Le processus p1 prend un point de reprise pour toutes les 3 unités de temps.
- Le processus p2 prend un point de reprise pour toutes les 4 unités de temps.
- Le processus p3 prend un point de reprise chaque une unité de temps.

le processus P0 échoue au point X, puis il revient à son dernier point de reprise $\mathbf{C_{0,10}}$ et il incrémente $\mathbf{inc_0}$ à 1, et **rec_line** à 10 et envoie un message de recouvrement (0,10) à tous

les autres processus. En réponse à ce message, les processus P1, P2, et P3 ils vont mettre ces variables, inc_p =1 (M.inc) et rec_line :=10 et ils vont revenir à leurs derniers points de reprises ($C_{1,12}$), ($C_{2,10}$), ($C_{3,10}$) respectivement dont le numéro de séquence (sn ≥ rec_line). La ligne de recouvrement est représentée par la ligne en zigzag bleu dans la Figure2. 4 traitement des messages pendant le recouvrement arrière.

2.3.2.5 classification des messages

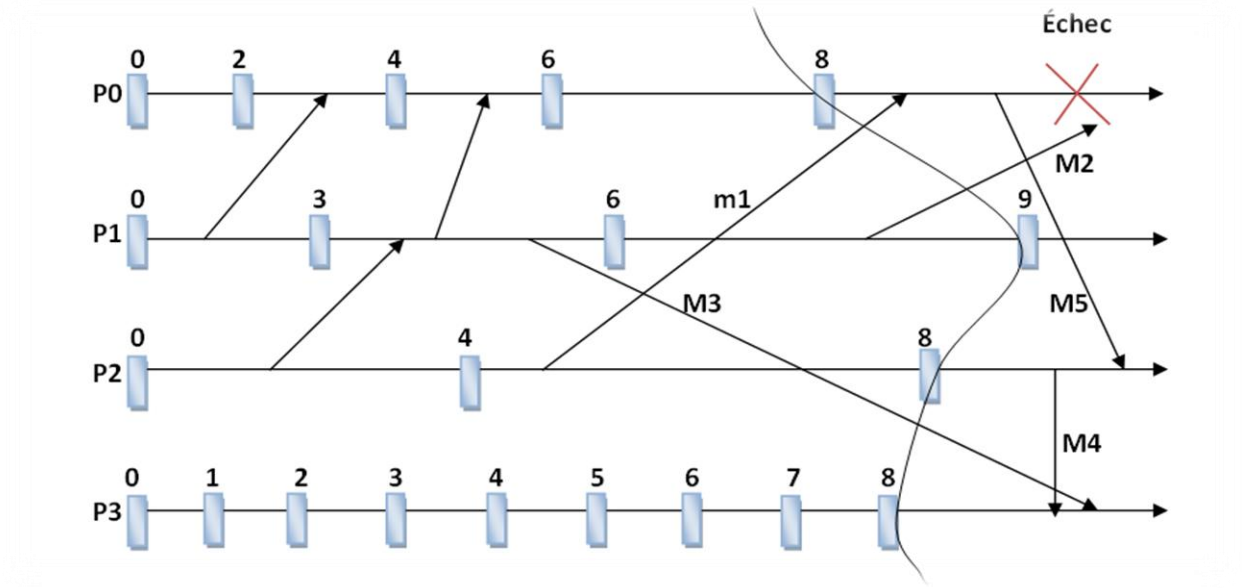


Figure2. 5 différent type de messages

Nous utilisons la Figure2. 5 différent type de messagespour aider à classer les différents types de messages qui doivent être traitées lors du recouvrement.
Soit un système composé de quatre processus P1, P2, P3,P4

Message perdu :

Ce sont des messages dont l'envoi d'un événement n'est pas annulé, mais la réception des événements est annulée en raison de rollback.
Dans la Figure2. 5 différent type de messageson a le message m2 qui est envoyer mais sa réception est annulé a cause de l'échec et donc il fait un recouvrement à son dernier point de reprise.

Message en transit :

Ce type de message se pose quand un processus fait un retour à un point de reprise avant la réception du message alors que l'expéditeur ne doit pas revenir à un point de avant de l'envoyer.
La Figure2. 5 différent type de messagesmontre que les messages m1 est message en transit par rapport au point de reprise globale ($C_{1,8}$, $C_{2,9}$, $C_{3,8}$,$C_{4,8}$)

Les messages omis :

Ce type de message apparait si le processus récepteur retourne à un point de reprise qui précède sa réception.

Message retardé :

Ce type de message apparait quand l'émission est faite avant la ligne de recouvrement mais il n'est pas reçu qu'après la ligne de recouvrement.

Dans la Figure2. 5 différent type de messagesles messages m2 se sont des messages retardés.

Message orphelin :

Un message orphelin est un message dont l'émission a été annulée (non enregistrer) mais dont la réception n'a pas été annulée.

Les messages orphelins ne se posent pas si les processus roulent vers une approche globale cohérente

Message dupliqué :

Ce sont les messages retransmis à cause du recouvrement, Le message m4 était envoyé et reçu avant le recouvrement. En raison de recouvrement, P4 à annulé la réception de m4 et p3 à annulé l'envoie de **m4,** Après P4 fait un retour au point de reprise avec un numéro de séquence (sn=8), il convient de ne pas retransmettre le message m4 depuis p3 à annulé l'envoie de m4, si P4 retransmit m4 sa va etre un message doublé, car il sera de nouveau envoyé a p3.

2.3.2 Cas de plusieurs fautes :

L'algorithme de recouvrement précédant est proposé pour le cas d'une seule faute, Cependant, plusieurs processus pourraient échouer en même temps. Nous allons présenter l'algorithme de recouvrement pour faire face à l'échec simultané de plusieurs processus.

2.3.2.1 Intialisation de la structure de donnée à Pp :

Sn_p : entier (:=0) ; //numéro de séquence du point de reprise courant, initialisé à 0

$Next_p$:entier (:=1) ;// numéro de séquence associé au prochain point de reprise initialisé à 1

Inc_p : entier (:=0) ; // numéro de séquence de la courante incarnation,initialisé à 0

Rec_line_p :entier(:=0) ; // la dernière line de recouvrement initialisé à 0

$Inc\_rec\_set_p$: un ensemble de pairs {(0,0),..} ;

Dans cet algorithme les deux premières procédures de l'algorithme précédant sont les même.

2.3.2.2 L'algorithme :

Quand un processus P_p décide d'envoyer un messageM :

Quand un processus décide d'envoyer un message il va définir son numéro de sequence et son point reprise comme étant le numéro de la ligne de recouvrement aprés il peut envoyer le message.

M.sn := sn_p ;
M.rec_line := rec_line_p ;
M.inc := inc_p ;
Envoyer (M) ;

Quand un processus reçoit un message M :

Le processus ayant reçu un message m composé d'une pair d'entiers (inc,rec) vérifie si (M.sn<rec) si c'est le cas, il va poursuivre et enregistrer le message Sinon il va l'annuler, une suite de conditions et vérifier pour traiter les message comme on le voie dans la procedure de reception de message .

La procedure suivante va décrire le comportement du processus aprés son échec

Si (M.inc_rec_set ⊂M.$inc\_rec\_set_p$) alors
trouver le plus petit élément (inc,rec) ∈ ($inc\_rec\_set_p$-M.inc_rec_set) ;
si (M.sn<rec) alors
poursuivre le message ;
sinon
annulé le message ;
sinon si (M.inc_rec_set =M.$inc\_rec\_set_p$) alors
si (M.sn>sn_p) alors
prendre un point de reprise C ;
C.sn := M.sn ;
Sn_p := C.sn ;
Poursuivre le message M ;
Sinon
trouver le plus petit élément (inc1, rec1) ∈ (M.inc_rec_set - $inc\_rec\_set_p$) ;
si ($inc\_rec\_set_p$ ∉M.inc_rec_set) alors
trouver le plus petit élément (inc2, rec2) ∈ (inc_rec_setp-M.inc_rec_set) ;
si ((rec1<rec2) ∨ ((rec1=rec2) ∧ (inc1<inc2))) alors
suprimmé (inc2,rec2) de $inc\_rec\_set_p$;
sinon si (M.sn<rec2) alors
poursuivre le message ;
partir à last ;
sinon
annulé le message ;
partir à last ;
rec_line_p := rec1 ;
inc_rec_set := $inc\_rec\_set_p$∪ {(inc_p, rec_line_p)} ;
restaurer(p) ;

replay_logged_message(p)
traiter le message comme s'il vient d'arrivé;

Recouvrement après l'échec d'un processus P_p :

Quand un processus tombe en panne il incrémente son numéro d'incarnation et envoie des messages de recouvrement à tous les autres processus.

$inc_p := ((inc_p div N) + 1) * N + p$;
$rec\_line_p := sn_p$;
$inc\_rec\_set_p := inc\_rec\_set_p \cup \{(inc_p, rec\_line_p)\}$;
envoyer recouvrement ($inc\_rec\_set_p$) à tous les autres processus ;
replay_logged_message(p) ;
continuer normale ;

quand un processus reçoit un recouvrement ($inc\_rec\_set_p$) d'un processus P_p :

Quand un processus reçoit un message de recouvrement, il test si son ensemble de (inc,rec) appartient ou pas à l'ensemble de P_p

- Si il n'appartient pas à p_p Dans ce cas, chacun des deux ensembles contieent au moins un élément qui n'appartient pas à l'autre. Donc, il y a des recouvrements simultanées lancées entre les deux.
- Sionon il va rejouer le message par l'appel de la procedure replay

si ($inc\_rec\_set_q \notin inc\_rec\_set_p$) alors
  trouver le plus petit élement (inc1, rec1)$\in$($inc\_rec\_set_q - inc\_rec\_set_p$) ;
    si ($inc\_rec\_set_p \notin inc\_rec\_set_q$) alors
      trouver le plus petit élement (inc2, rec2) $\in$ ($inc\_rec\_set_p - inc\_rec\_set_q$) ;
        si ((rec1 < rec2) $\vee$ ((rec1=rec2) $\wedge$ (inc1 < inc2))) alors
          supprime (inc2,rec2) de $inc\_rec\_set_p$;
        sinon
          partir à last ;
          rec_line_p :=rec1 ;
          inc_p :=inc1 ;
          $inc\_rec\_set_p := inc\_rec\_set_p \cup \{(inc_p, rec\_line_p)\}$;
          replay_logged_message(p) ;
          catch_up_with_recovery($inc\_rec\_set_q$, p) ;
          last : continuer normale ;
        finsi
    finsi
finsi

procedure replay_logged_message(p) :

la procedure replay logged message nous permet de rejouer les messages dans le cas où un processus tombe en panne

si (M.sn < rec_line_p) alors
    rejouer le message M
sinon
    supprimer le message

procedure roll_back(p :entier) :

la procedure rollback permet au processus de revenir au dernier point de reprise lors d'une panne ou prendre un point de reprise forcé dans le cas où le numéro de la ligne de recouvrement est superieure à son numero de sequence ($rec\_line_p > sn_p$)

si ($rec\_line_p > sn_p$) alors
  prendre un point de reprise C ;
  C.sn := rec_line_p ;
  Sn_p := C.sn ;
Sinon
  Trouver le dernier point de reprise C tel que (C.sn ≥ rec_line_p) ;
  Sn_p := C.sn ;
  Cur_chkpt_p := C ;

procedure catch_up_with_recovery(i : pairs d'entier, p :entier) :

p_p à besoin de trouver le minimum des elements de $inc\_rec\_set_p$ et $inc\_rec\_set_q$ et le considére comme une nouvelle valeur pour l'ensemble $inc\_rec\_set_p$, et met à jours inc_p et rec_line_p et verifie la condition ($sn_p < rec\_line_p$) pour prendre un point de reprise.

$inc\_rec\_set_p$:= Min (i , $inc\_rec\_set_p$) ;
soit (inc, rec) le dernier élement dans $inc\_rec\_set_p$;
rec_line_p := rec ;
si ($sn_p < rec\_line_p$) alors
  prendre un point de reprise ;
  C.sn := rec_line_p ;
  sn_p := C.sn
finsi

fonction Min(i,j :des pairs d'entier) :

La fonction min à comme paramétres deux ensembles, elle va trouvé le plus petit élments de deux ensembles et elle va etre applé dans la fonction catch pour définir un point de reprise .

```
var : temp : pairs d'entier ;
début
        si (i=j ) alors
         min := i ;
        sinon
         temp :=0 ;
        tant que ((i≠ 0) et (j≠ 0)) faire
debut
        soit (inc1, rec1) ∈ j et (inc2, rec2) ∈ j le plus petit élement de i et j ;
        si ((rec1 < rec2) ∨ ((rec1=rec2) ∧ (inc1 < inc2))) alors
         temp := temp ∪ {(inc1, rec1)}
        sinon
         temp := temp ∪ {(inc2, rec2)}
         i := i-{(inc1, rec1)} ;
         j := j-{(inc2, rec2)} ;
fin
        temp :=(temp ∪i ∪j ) ;
        min :=temp ;
fin ;
fin de Min;
```

2.3.2.3 Exemple :

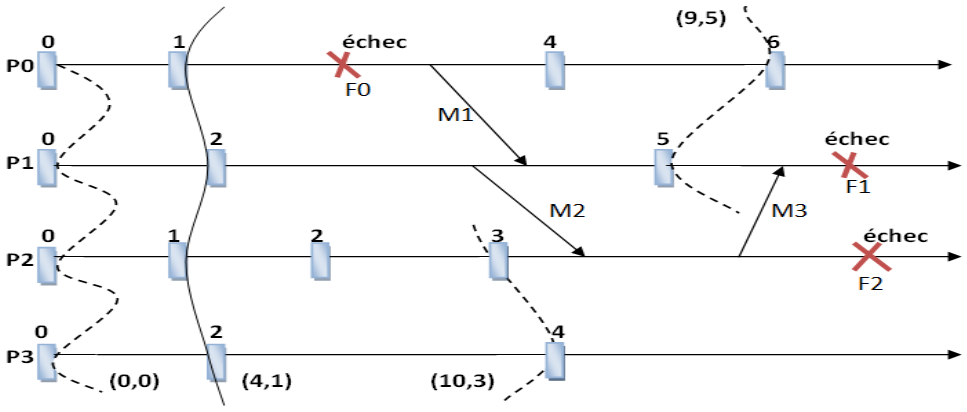


Figure2. 6 exemple de cas de plusieurs pannes.

La Figure2. 6 exemple de cas de plusieurs pannes.represente un exemple sur le cas de plusieurs pannes

si un processus tombe en pannes il incrémente son numéro d'incarnation, de telle manière que d'autres processus peuvent reconnaître qu'il ya plusieurs échecs en comparant le numéro d'incarnation reçu dans le message avec son propre numéro d'incarnation. Pour être précis, quand un processus P_p échoue, il définit le numéro l'incarnation inc_p, inc_p :=(((inc_p div N) +1)*N+p).
Considérons le calcul distribué composé de quatre processus illustré dans la Figure 2. 1

- ✓ P0 échoue au point F0 et revient à son dernier point de reprise $C_{0,1}$.
- ✓ Il définit son numéro d'incarnation inc_0 := ((inc_0 div 4)+1)*4+0 (ici, inc_0=4).
- ✓ rec_line_0 :=1 et envoie le message rollback(1,4) à tous les autres processus.
- ✓ À la réception de ce message, tous les processus reviennent aux points de reprise sur la ligne de recouvrement marquée (4, 1).

Après le recouvrement on suppose que p1 et p2 échouent simultanément au point indiqué par F1 et F2, respectivement.

- ✓ après la panne de F1, P1 revient à son dernier point de reprise $C_{1,5}$.
- ✓ Il définit son numéro d'incarnation inc_1 := ((inc_1 div 4)+1)*4+0 (ici, inc_1=9).
- ✓ P1 envoie le message de recouvrement(9,5) à tous les autres processus.
- ✓ Quand P0 reçoit le message de recouvrement envoyé par P1, il revient a son dernier point de reprise $C_{0,6}$ (c'est le plus récent point de reprise de P0 avec le numéro de séquence ≥ 5).

Au même temps P3 reçoit le message de recouvrement envoyé par P2 et revient à son dernier point de reprise $C_{3,4}$.

Les lignes de recouvrement partielles établies à la suite de ces reculs sont présentés dans la Figure2. 6 exemple de cas de plusieurs pannes.avec (10,3) et (9,5), Ensuite, lorsque P1 reçoit le recouvrement (10,3) envoyer par P2, P1 va reconnaitre qu'il ya eu des échecs concurrents.

2.3.2.4 Exemple :

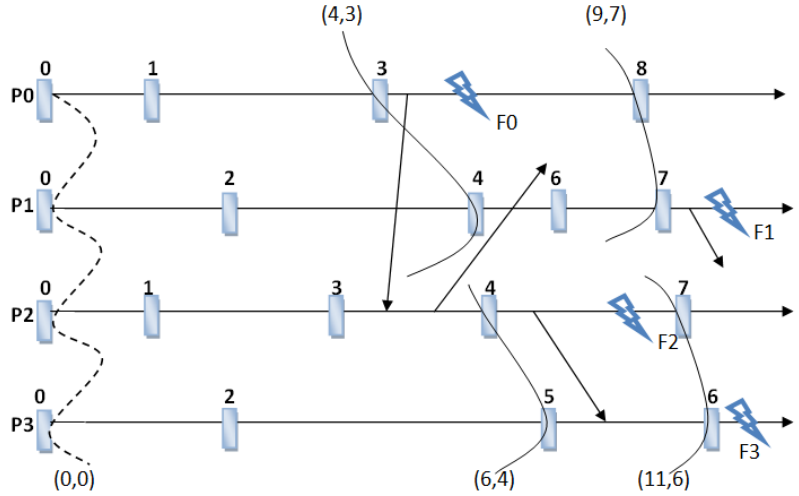


Figure2. 7 recouvrement partiel

La Figure2. 7 recouvrement partiel montre un calcul distribué composé de quatre processusp0 et p2 tombent en panne aux points indiqués par F 0 et F2, respectivement.

Comme resultat p0 et p2 vont envoyer un message de recouvrement {(0,0), (4,3)} et {(0,0), (6,4)} à tous les autres processus, supposant que p1 reçoit un message de recouvrement {(0,0), (4,3)} en premier et donc il revient à son point de reprise avec le numéro de séquence est égale à 4, la ligne de recouvrement partielle établie par P0 et P1 est représenté avec une étiquette (4,3).

On suppose maintenant que p3 reçoit message de recouvrement {(0,0), (6,4)} de la part de p2 et donc il revient à son point de reprise avec le numéro de séquence est égale à 5, la ligne de recouvrement partielle établie par P2 et P3 est représenté avec une étiquette (6,4).

Ensuite, supposons que P1 échoue à la le point indiqué par F1 avant qu'il connait l'échec de P2 à F2, et p3 échoue au point indiqué par F3 avant qu'il arrive à connaître les échecs F0 et F1, Comme résultat de l'échec à F1, p1 revient à son point de reprise avec le numéro de séquence $C_{1,7}$ et envoie le message de recouvrement {(0,0), (6,4), (9,7)} à tous les autres processus.

De même, en raison de l'échec à F3, p3 revient a son point de reprise $C_{3,6}$ et envoie le message de recouvrement {(0,0), (6,4), (11,6)} à tous les autres processus, après supposant que p2 reçoit un recouvrement {(0,0), (6,4), (11,6)} de la part de p3, et il revient à son point de reprise $C_{2,7}$..

En même temps, on suppose que p0 reçoit {(0,0), (4,3), (9,7)} de la part de P1 et il revient à son point de reprise $C_{0,8}$,À ce stade, ni P0 et ni P1 savent qu'il ya eu les échecs F2 et F3 De même, P2 et P3 ne connaissent pas les échecs F0 et F1

Toutes les lignes de recouvrement partielles établies à ce stade sont présentées dans laFigure2. 7 recouvrement partiel.

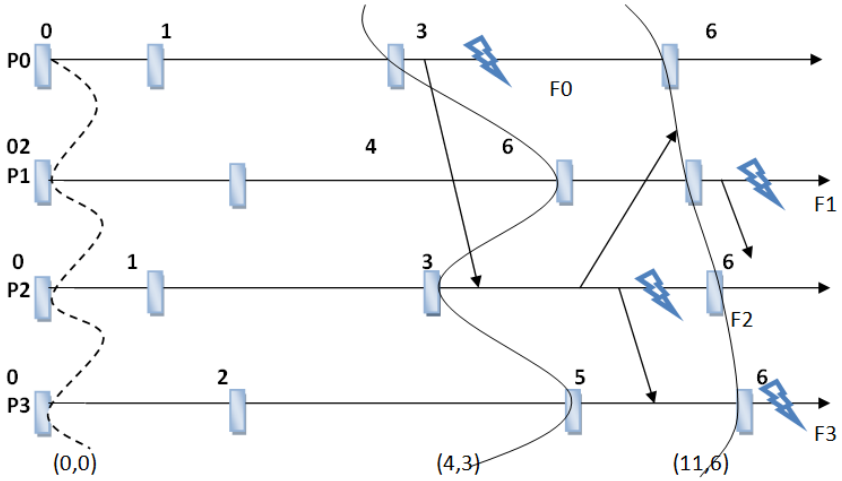


Figure2. 8 recouvrement complet établi après plusieurs pannes

Quand P0 et P1 viennent de connaître les échecs à F2 et F3, ils vont ignorer l'échec à F2 ou le numéro de la ligne de recouvrement correspondant à l'échec F2 est 4, qui est plus grand par rapport à la ligne de recouvrement correspondant à l'échec F0 à savoir, 3. Cependant, P0 et P1 vont revenir en réponse à l'échec de F3, de même, lorsque P2 et P3 viennent de connaître les échecs à F0 et F1, ils vont revenir en réponse à l'échec F0 et seulement prendre un point de reprise avec le numéro de séquence 6 en réponse à l'échec F1.

Après que tous les processus sont au courant à propos des quatre échecs, la ligne de recouvrement complète est établie et présentées dans la Figure2. 8 recouvrement complet établi après plusieurs pannes

Conclusion

Les algorithmes de recouvrements permettent de restaurer et récupérer les messages en cas de panne grâce au point de reprise. Ce chapitre résume les principales descriptions du protocole MS et le Protocole HMR et aussi nous avons expliqué leur mécanisme de recouvrement et détaillé le traitement de chaque type de message et de comportement pour chaque cas.

Chapitre 3

Simulation et analyse de performance

3.1 Introduction :

Pour résoudre les problèmes liés à un système donné, on fait appel à des algorithmes ou des protocoles, ces algorithmes doivent être testés et évalués pour les valider, et pour satisfaire leurs insuffisances, et cela à l'aide d'un outil de simulation [8], le but de ce chapitre est de simuler les deux algorithmes (protocole MS et le protocole HMR) avec le simulateur NS-2, afin de faire une étude comparative pour évaluer la performance des algorithmes.

3.2 Les techniques d'évaluation des performances

Il existe différentes techniques pour l'évaluation des performances d'un système :

3.2.1 émulation

Le principe de cette technique est de faire des mesures et de les analyser directement sur un système réel. Cette technique permet de comprendrele vrai comportement du système. Mais elle n'est pas toujours réalisable car le fonctionnement de système réel peut être perturbé, en plus les résultats issus de cette mesure ne reflètent qu'une seule trajectoire du système [6].

3.2.2 Modélisation

Le principe de cette technique est de réduire le système en un modèle mathématique (les automates, les réseaux de pétri, les approches de probabilités,…) et de l'analyser numériquement. Généralement, certaines hypothèses sont posées pour simplifier l'étape de modélisation du système et rendent l'évaluation numérique faisable. Ces hypothèses simplificatrices peuvent toucher la fidélité de la représentation du système [6].

3.2.3 Simulation

Cette technique est basée sur l'implémentation d'un modèle simplifié du système à l'aide d'un programme de simulation adéquat. Cette méthode traduit le comportement du système à évaluer d'une manière réaliste. La simulation permet en plus de visualiser les résultats sous forme de graphes faciles à analyser et à interpréter.

Afin d'évaluer la performance des algorithmes étudiés nous avons besoin de les simuler pour présenter les résultats qui permettent de réalisé une étude comparative entre les algorithmes et de tester leurs fonctionnement.

3.3 L'outil de simulation

NS-2 est un simulateur à événements discrets, écrit en C++. C'est le simulateur le plus célèbre dans le domaine de la simulation des réseaux, Il est essentiellement élaboré avec les idées de la conception par objets, de la réutilisation du code et de modularité. Il est aujourd'hui un standard de référence en ce domaine, plusieurs laboratoires de recherche recommandent son utilisation pour tester les nouveaux protocoles.

3.4 L'environnement du travail

3.4.1 Enivrement logiciel

Nous avons réalisé la simulation sous linux ubuntu, en utilisant la version NS-allinone 2.35 car la manipulation et la configuration de NS est plus facile sous linux.

3.4.2 Enivrement matérielle

Nous avons utilisé un ordinateur dont la configuration suivante :

Figure 3. 1 envirenment matérielle

3.5 Réalisation de la simulation

La simulation nécessite des données qui caractérisent l'environnement, tels que la surface du réseau, la topologie utilisée, le protocole à simuler ... etc. Généralement, ces données ne sont pas définies en NS. Pour cela, l'utilisateur doit définir les informations (données) en utilisant le langage C++[8].

- ✓ Afin de réaliser un algorithme, nous devons créer deux fichiers écrits en langage C++ (**.h, *.cc*).
- ✓ Le premier (**.h*) est un fichier d'en-tête, qui contient la structure des messages échangés entre les sites, le deuxième (**.cc*) contient les fonctions nécessaires de l'algorithme (envoi et réception des messages, ...).
- ✓ La compilation de du fichier (*.cc) nous permet d'avoir un fichier de type objet (**.o*), ce dernier doit être intégré dans le simulateur NS-2, en lui ajoutant à la variable *OBJ_CC* du fichier (*makefile*), enfin, on recompile le noyau de NS par la commande *make* écrite dans un terminal.

3.6 Etapes de la simulation :

Après l'ouverture du terminal et le répertoire ou se trouve les fichiers contenant le programme à exécuter :

- ✓ Apres avoir vérifié que le fichier source (*.cc) fonctionne correctement (compilation avec la commande Make et ouverture du fichier (*.o)).
- ✓ En passe au fichier (scenario.tcl) qui permet a l'utilisateur de saisir les information nécessaire pour la simulation (temps de simulation, nombre de pannes, nombre de processus qui tombent en panne…) ces fichiers seront ensuite enregistrer dans un fichier rollback.txt.
- ✓ Le fichier scenario.txt est le résultat du premier fichier qui contient les informations d'envoi des messages.

- ✓ Le fichier basic.txtcontient l'identifiant du processus et le moment de prise d'un point de reprise.

- ✓ Le fichier panne.txt contient le moment et l'identification du processus qui tombe en panne.

- ✓ Le fichier final.txt est un fichier global sous forme de tableau trié qui regroupe tous les temps nécessaires pour la simulation (temps d'envoi, réception, temps de panne, temps pour prendre un point de reprise).

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

Lancer le nam où on peut voir l'animation de notre simulation afin de voir les nœuds et leurs liens, l'envoi et la réception d'un message…etc.

Obtenir les résultats de la simulation pour tracer les courbes afin d'analyser la performance de l'algorithme simulé.

On peut résumer les étapes de réalisation d'un algorithme dans la figure suivante :

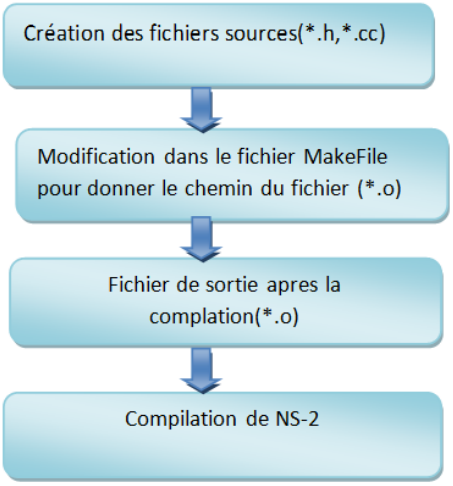


Figure 3. 2étapes de simulation

3.7 Paramètres et métrique de simulation

Afin de comparer les deux protocole (MS,HMR) dans le cas de plusieurs fautes nous avons besoin des paramètres des métriques :

Tableau3. 1 paramétres de simulation(cas de N pannes)

Les paramètres	**Descriptions**
Nb_processus	Nombre de processus
L	Longueur d'intervalle qui sépare deux points de reprises
Nb_pannes	Nombre de pannes
Nombre de fautes	Nombre de fautes suit la loi exponontielle avec λ =0.2 est fixe pour toute la simulation

Tableau3. 2 les métriques de simulation (cas de n pannes)

Les métriques (avec fautes)	Descriptions
Nb_forced	Nombre de points de reprises forcés
Deleted	Nombre de points de reprises supprimés
Nb_msg	Nombre de messages sauvegardés
Dist	La distance de recouvrement

scenarion 1 : Variation du nombre de processus (Nb_processus)

Dans ce scenario nous étudions l'influence du nombre de processus sur le nombre des points de reprise forcés (F), le nombre de messages sauvegardés, nombre de point de reprise supprimés et la distance de recouvrement.

Scenario2 : Variation de lalongueur d'intervalle (L) :

Dans ce scenario, nous étudions l'influence de la longueurd'intervalle (L) qui sépare deux points de reprises spontanés sur les métriques de simulation.

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

Tableau3. 3 Variation des paramètres de simulation

	Nombre de processus	**Temps de simulation**	**Nombre de messages**	**Longueur d'intervalle**
Scenarion1	**10<Nb_processus<60**	**500s**	**300**	**40**
Scenario 2	**40**	**500s**	**300**	**10<L<60**

Nombre dee points de reprise forcés

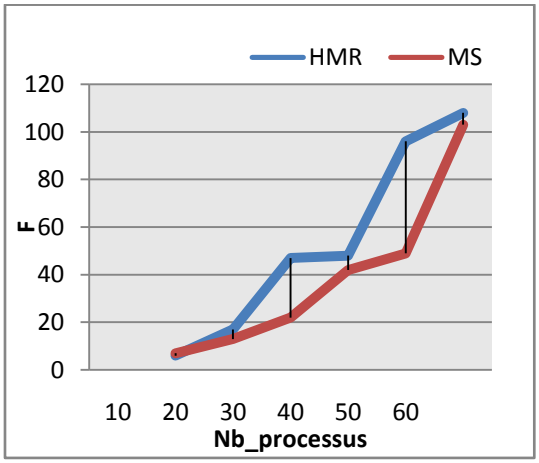


Figure 3. 3 Nb_processus verssus F

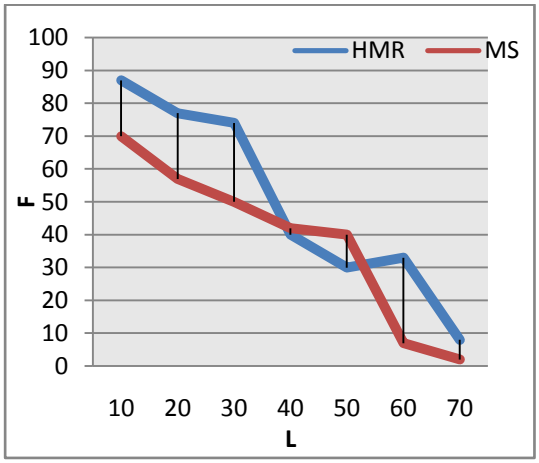


Figure 3. 4 L verssus F

LesFigure 3. 3 Nb_processus verssus F Figure 3. 4 L verssus F, representent le nombre de points de reprise forcés en fonction de du nombre de processus et la longueur d'intervalle respectivement

Dans la Figure 3. 3 nous remarquons l'augmentation du nombre de point de reprise forcés (F) avec l'augmentation de nombre processus .Lorsque le nombre de processus(NB_Processus) varie entre 10 et 30 le nombre de point de reprises forcés pris par les deux algorithmes est à peu près le même (il ya pas une grande différence), Dans le cas 30<Nb_processus<60 la diférence et un peux plus grande on remarque que l'algorithme (HMR) prend plus de point de reprises que l'algorithme (MS).car chaque processus va recevoir un grande nombre de messages et si la condition d'induction est verifié (pour les deux algorithmes MS et HMR) du messages n'est pas vérifier, le processus va être obligé de prendre un point de reprise forcé avant de traité le message.

Dans la Figure 3. 4 Le nombre de point de reprise forcés (F) diminue avec l'augmentation de l'intervalle. Dans le cas 10<L<30 le nombre de points de reprise pris par l'algorithme HMR est plus grand que celui pris par MS.

Pour le reste de l'intervalle la différence entre le nombre de point de reprises pris par les algorithme HMR et MS n'est pas grande.

Nombre de points de reprise supprimé à cause de recouvrement

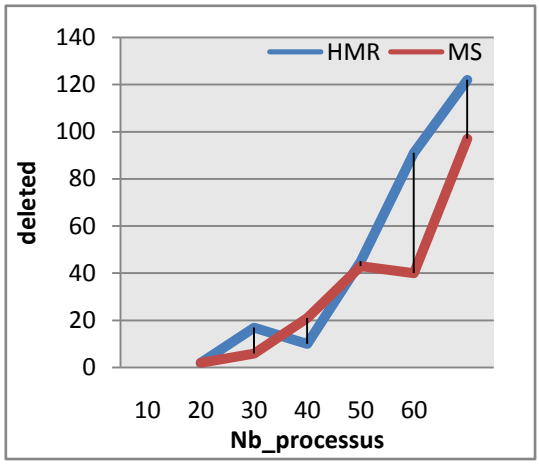


Figure 3. 5 Nb_processus verssus deleted

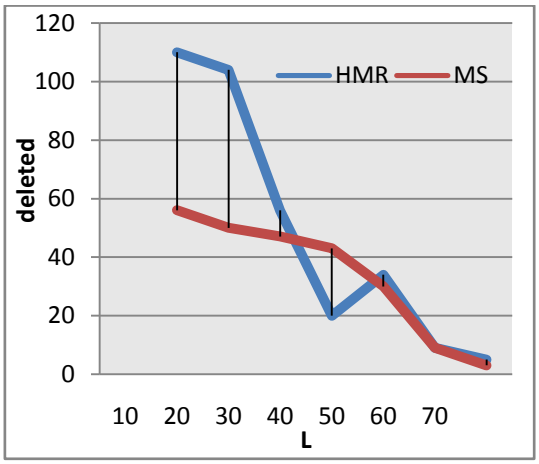


Figure 3. 6 L verssus deleted

LesFigure 3. 5 Nb_processus verssus deleted et Figure 3. 6 L verssus deleted representent le nombre de points de reprise supprimés en fonction de du nombre de processus et la longueur d'intervalle respectivement .

La Figure 3. 5 Nb_processus verssus deleted nous montre que Le nombre de points de reprise supprimés pour les deux algorithmes (MS, HMR) augmente avec l'augmentation du nombre de processus.

- Pour l'intervalle 10<NB_processus<30 il ya pas une grande diférence entre MS et HMR.
- Pour l'intervalle 40<NB_processus<60 le nombre de points de reprise forcé supprimés par l'algorithme HMR et un peu plus grand que celui de l'algorithme MS.

Dans la Figure 3. 6 L verssus deleted Le nombre de points de reprise supprimés diminue pour les deux algorithmes (MS et HMR) avec l'augmentation de la longueur d'intervalle.

Pour l'intervalle (10<L<30) l'algorithme HMR supprimes un nombre plus grand de points de reprise par rapport a l'algorithme MS, pour le reste de l'intervalle la différence du nombre de points de reprise supprimés entre l'algorithme MS et HMR à diminué.

La diminution de nombre des points de reprise supprimé est du à la diminution du nombre de points de reprise quand la longueur d'intervalle augmente.

Distance de recouvrement

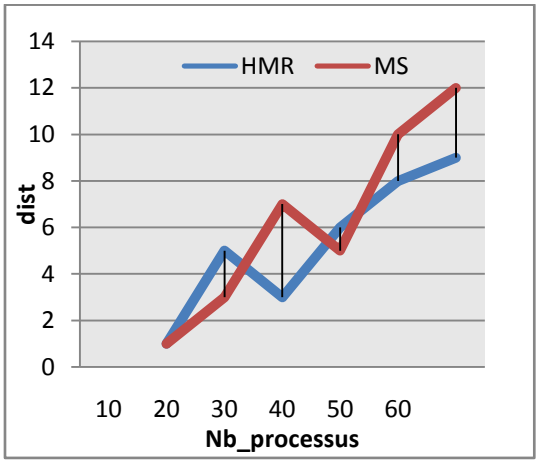


Figure 3. 8 Nb_processus verssus dist

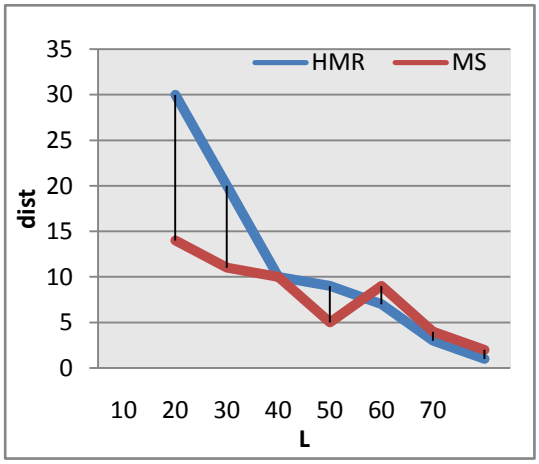


Figure 3. 7 L verssus dist

Les Figure 3. 8 Nb_processus verssus distFigure 3. 7 L verssus dist representent la distance de recouvrement forcés en fonction de du nombre de processus et la longueur d'intervalle respectivement

Dans la Figure 3. 8on remarque que la distance de recouvrement augmente avec l'augmentation du nombre du nombre de processus, avec un grand nombre de processus les les messages de recouvrement augmente est implique l'augmentation du nombre de sauts pour trouvé un ligne de recouvrement.

Dans figure Figure 3. 7 L verssus distla distance de recouvrement diminu avec l'augmentation de la longeur d'intervalle.

Nombre de messages sauvgardés

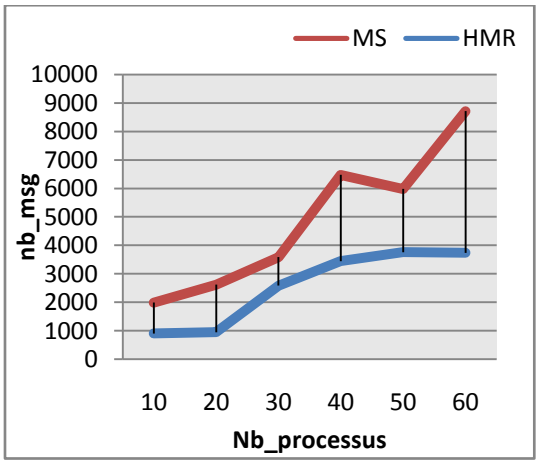


Figure 3. 9 Nb_processus verssus nb-msg

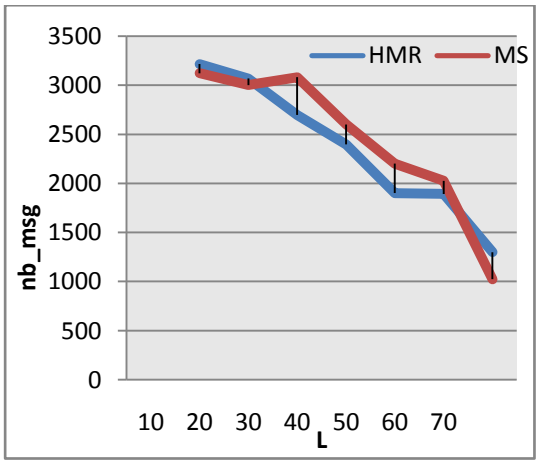


Figure 3. 10 L verssus nb_msg

Les(Figure 3. 9 Nb_processus verssus nb-msg Figure 3. 10 L verssus nb_msg) representent le nombre de message sauvegarder en fonction de du nombre de processus et la longueur d'intervalle respectivement.

La Figure 3. 10représente l'augmentation du nombre de message par rapport au nombre de processus. Nous remarquons le nombre de messages sauvegardés augmente i pour les deux algorithmes, celaestjustifiée par un échange important de messages entre les processus.

On remarque dans Figure3.9 que Le nombre de messages enregistrés diminue avec l'augmentation de l'intervalle pour les deux protocoles (MS et HMR).

Les deux mécanismes de recouvrement présentent des avantages et des inconvénients on peux résumé leur avantages et inconvéniants dans le tableau suivant :

Tableau3. 4 avantages et incovéniants des deux protocoles

	Avantages	**Inconvéniants**
Protocole MS	✓ Traite le cas de plusieurs pannes ✓ Nous permet de restaurer les information en cas de panne ✓ Ne prend pas un grand nombre de point de reprise	✓ Prend un grand nombre de messages ✓ Supprime un grand nombre de proints de reprises ✓ La distance de recouvrement a augmenter avec l'augmentation du nombre de prcocessus
Protocole HMR	✓ Traite le cas d plusieurs pannes ✓ Nous permet de restaurer les information en cas de panne ✓ Ne prend pas un grand nombre de messages	✓ Prend un grand nombre de point de reprise ✓ Supprime un grand nombre de proints de reprises ✓ La distance de recouvrement a augmenter avec l'augmentation du nombre de prcocessus

Conclusion :

Le comportement des deux protocoles simulés a été soigneusement analysée à travers une étude de simulation, qui nous à permit de tirer les remarques suivantes :

- Pour les systèmes qui sont constitué d'un grand nombre de processus la performance des deux algorithmes est dégradée.
- Malgré que les deux protocoles traitent le cas de plusieurs fautes il ya toujours un nombre de point de reprise supprimé et il augmente avec l'augmentation du nombre de processus.

- Le nombre de message sauvegardé par le protocole MS est plus grand que celui du HMRcar le protocole MS induiit un nombre de points de reprise forces mois que le protocole HMR.
- La manière dont les deux protocoles suivent pour prendre un point de reprise spontané a un impacte sur la performance des deux algorithmes.
- La reprise, même si elle est la plus utilisée, peut être couteuse en temps et en espace de stockage.
- Avec l'augmentation du nombre de processus le nombre de pannes augmente aussi et donc la tolérance aux panne va devinir un élément indisponsable pour les processus.
- Les deux protocoles ont des avantages et des iconvéniants pour l'éspace de stockage le protocole HMR est meilleur car il enregistre moins de messages et pour le nombre de point de reprises forcés le protocole MS est meilleur.

CONCLUSION GÉNÉRALE

Le recouvrement arrière est une solution pour assurer la tolérance aux fautes dans les processus, parmi les techniques existante celle qui sont basées sur les points de reprise.

Dans ce mémoire, nous avons étudié les performances des deux protocoles (MS et HMR) de recouvrement arrière avec les points de reprise. Nous avons présenté des notions de bases sur les systèmes repartis et la tolérance aux pannes, ainsi que la description détaillée de ces deux algorithmes, Nous avons choisi de simuler les deux protocoles dans le cas de plusieurs panne, Ces résultats ont été bénéfiques pour analyser le comportement des protocoles. Ce mémoire nous a permis de :

- ✓ Comprendre le fonctionnement des algorithmes de points de reprise .
- ✓ Améliorer nos connaissances sur tolérance aux pannes.
- ✓ Comparer par la simulation les performances de l'algorithme étudié
- ✓ Maitriser l'outil de simulation NS-2.
- ✓ Conclure que les deux protocoles ont des avantages et des iconvéniants pour l'éspace de stockage le protocole HMR est meilleur car il enregistre moins de messages et pour le nombre de point de reprises forcés le protocole MS est meilleur.

Comme perspectives et amélioration de ce travail au futur, nous avons pensé à :

- améliorer les protocoles afin de réduire le nombre de messages et le nombre de points de reprise supprimés
- Utiliser le principe des deux protocoles pour résoudre ce problème dans d'autres systèmes tels que : les réseaux mobiles, les grilles…etc.
- Améliorer les algorithmes pour traiter les autres types de pannes.
- Minimisé le surcout des algorithmes.

BIBLIOGRAPHIE

[2] Matthieu Fertré ,Intégration d'un mécanisme de reprised'applications parallèles dans un systèmed'exploitation pour grappe. Rapport de stage Master 2 de Recherche en Informatique Université de Rennes 1,juin 2015.

[3]Abdelhafidi.Z, points de reprises dans les systemes répartis étude bassée sur la simulation des protocoles CIC assurant la propriété RTD. Thése Mgistere à l'université Amar Thelidji-Laghouat specialité informatique, 2008.

[4] Massata.N. Techniques de gestion des défaillances dans les grilles informatiques tolérantes aux fautes. Thése de doctorat à L'Université Pierre et Marie Curie - Paris, 2013.

[5] Manivannan, D., & Singhal, M. (1996, May). A low-overhead recovery technique using quasi-synchronous checkpointing. In *Distributed Computing Systems, 1996., Proceedings of the 16th International Conference on* (pp. 100-107).

[6] Barkat.F et Oouled kouider.L, Analyse de performance des algorithmes de point de reprise avec recouvrement. Thése d'ingenieur à L'Université Amar Thelidji-Laghouat specialité informatique,PFE 2009

[7] Hélary, J. M., Mostefaoui, A., & Raynal, M. (1999). Communication-induced determination of consistent snapshots. Parallel and Distributed Systems, IEEE Transactions on, 10(9), 865-877.

[8]Oubbati.O,proposition et simulation d'un algorithme de partage de ressource dans les MANETs basé sur l'algorithme de Naimi et Trehel , thése de master à l'université d'Amar thelidji Laghouat specialité informatique, PFE 2011.

WEBOGRAPHIE

[1] [En ligne]https://fr.wikipedia.org/wiki/Tol%C3%A9rance_aux_pannes

[10] [En ligne] http://www.fredptitgars.net/informatique/Les-systemes-a-tolerance-de-pannes

[9] [En ligne]http://www.fredptitgars.net/informatique/Les-systemes-a-tolerance-de-pannes

Annexe 1

Exemple de code source

Dans notre nous présentant un exemple de code source de l'algorithme MS dans le cas de plusieurs fautes (ms.h , ms.cc, withfault.tcl,ms.tcl)

MS.H :

```
#ifndef ns_ms_h
#define ns_ms_h
#include <utility>
#include <map>
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include <vector>
#include <utility>
#include <iostream>
#include <algorithm>
#include <set>


struct hdr_message {

        char type;
int Msn;            //numéro de séquence superposé avec le message M
int Minc;           //numéro d'incarnation (compteur d'erreur) superposé avec le message M
        int Mrec_line;      //numéro de la ligne de recouverement superposé avec le message M
        int Mid_proc;       //l'identifiant de processus superposé avec le message M
   vector <pair<int, int>> mirset;//ensemble de pairs qui contient les (inc,rec)

// La méthode pour accéder à l'entête du paquet
        static int offset_;

        inline static int &offset() { return offset_; }
        inline static hdr_message* access(const Packet* p) {
                return (hdr_message*) p->access(offset_);
        }
};
struct buf{
// utilisé pour sauvegarder le sn de point de reprise c, le sn superposé avec le message M
// et l'identifiant de processus
        int csn;
        int msn;
        int pid;
}
```

```
////////////////////////////////////////////////////////////////////////////////
//                    Création de la classe msAgent héritée de la classe Agent
////////////////////////////////////////////////////////////////////////////////
class msAgent : public Agent {

public:
        msAgent();
        int dist;
        int nb_fr_;       //Nombre de points forcés ajoutés lors de recouvrement
        int nb_force_;    //Nombre de points forcés
int nb_basic_;   //Nombre de points basics
        int deleted;     //Nombre de points supprimer lors de recouvrements
        int nb_msg_;      //Nombre de messages
        int nb_node_;    //Nombre de processus
int nb_sauv_;     //Nombre de messages sauvegardés
        int nb_annul_;   //Nombre de messages annulés
        int nb_retard_;   //Nombre de messages retardés
        int recieved;        //Nombre de messages reçus
      vector<pair<int, int>> irset;//ensemble de pairs qui contient les (inc,rec)
vector<buf>logmsg;
        virtual int command(int argc, const char*const* argv);
        virtual void recv(Packet*, Handler*);
        void Rollback(int);
        void Min(vector<pair<int, int>> ,vector<pair<int, int>>,vector<pair<int, int>> );
        void Catch(vector<pair<int, int>>, vector<pair<int, int>>,int,int,int);
        void replay(int,vector <buf>);


protected:
        vector<pair<int, int>> soust;
        int id_proc;       //l'identifiant de processus
        int sn;          // numéro de séquence de processus
        int inc;    // incrémenté à chaque fois qu'un processus échoue
        int rec_line;    //le numéro de la ligne de recouvrement.
        int rec;
        vector<int> checkpoint;
        int s;
        int t;
};
#endif  // ns_ms_h
```

ms.cc :

```
#include "ms.h"
#include "agent.h"
#include <utility>
#include <map>
#include <iomanip>
#include <iostream>
#include <vector>
#include <utility>
#include <iostream>
#include <algorithm>
using namespace std;
int hdr_message::offset_;

//**************************************************************************************
//                              pour calculer offset de l'en tête hdr_ms
//**************************************************************************************

static class msHeaderClass : public PacketHeaderClass {

public:
        msHeaderClass() : PacketHeaderClass("PacketHeader/ms",
                                        sizeof(hdr_message)) {
                bind_offset(&hdr_message::offset_);
        }
} class_mshdr;

//**************************************************************************************
*******
// procedure replay
void msAgent::replay(int n,vector <buf>logm )
{
for(int i = 0; i < logm.size();i++){
if (logm[i].msn<n)
 {
   nb_msg_ = nb_msg_ +1;
 }
else {
   logm.erase (logm.begin()+i);}
}
}
```

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

public:
        msClass() : TclClass("Agent/ms") {}
        TclObject* create(int, const char*const*) {
                return (new msAgent());
        }
} class_ms;

//**************************************************************************************
**********************
//                              pour faire la lien entre les variables en C++ et en TCL (bind)
//**************************************************************************************
**********************
msAgent::msAgent() : Agent(PT_MS), nb_node_(8), nb_msg_(0), nb_basic_(0), nb_force_(0),
id_proc(0),          sn(0),    rec_line(0),inc(0),nb_retard_(0),nb_annul_(0),nb_sauv_(0),s(0),deleted(0),
recieved(0),t(0),nb_fr_(0),dist(0)
                                                // PT_ms identifier un protocole dans les traces
{

bind("packetSize_", &size_);
bind("nb_node_", &nb_node_);
bind("nb_msg_", &nb_msg_);
bind("nb_basic_", &nb_basic_);
        bind("nb_retard_", &nb_retard_);
        bind("nb_annul_", &nb_annul_);
        bind("nb_sauv_", &nb_sauv_);
bind("nb_force_", &nb_force_);
        bind("id_proc",&id_proc);
        bind("recieved",&recieved);
        bind("inc",&inc);
        bind("rec_line",&rec_line);
        bind("sn",&sn);
        bind("deleted",&deleted);
        bind("indice",&s);
        bind("nb_fr_",&nb_fr_);
        bind("dist",&dist);
}
```

```
//*************************************************************************************
**********************
//                                          pour envoyer les paquets
//*************************************************************************************
**********************
int msAgent::command(int argc,const char*const*argv)
{
char find;
int i=0;
find='f';//ind=0;
int n,p;

if (argc == 2) {
if (strcmp(argv[1], "send-message") == 0) {
                    //Pour la taille de paquet

size_ =600;
                    //Pour la couleur de paquet
fid_ = 0;
                     // Création d'un nouveau paquet
          Packet* pkt = allocpkt();
             // L'accès à l'entête ms pour un nouveau paquet
hdr_message* hdr = hdr_message::access(pkt);
                    hdr->type = 'M';
                    hdr->Mid_proc = here_.addr_;
                    hdr->Msn = sn;
                     //irset.push_back(make_pair(inc,rec_line));
                    hdr->mirset = irset;
                        nb_msg_ = nb_msg_ +1;
          //Envoyer le paquet
send(pkt, 0);
return (TCL_OK); }
else if (strcmp(argv[1], "take-basic") == 0) {

                                    nb_basic_=nb_basic_+1;
                                        sn++;
                                           checkpoint.push_back(sn);

return (TCL_OK);}
else if (strcmp(argv[1], "take-initial") == 0) {

                    sn=0;inc=0;rec_line=0;
                                    nb_basic_=nb_basic_+1;
                                                checkpoint.push_back(sn);        //ce tableau permet de
sauvegarder les points de reprise spontanés

return (TCL_OK);}
else if (strcmp(argv[1], "send-rollback") == 0) {
size_ =600;
```

```
                    //Pour la couleur de paquet
fid_ = 1;
                     // Création d'un nouveau paquet
            Packet* pkt = allocpkt();
            // L'accès à l'entête des paquets ms
                    inc=((inc/nb_node_)+1)*nb_node_+here_.addr_;

                     hdr_message* hdr = hdr_message::access(pkt);
                    hdr->type = 'R';
                    hdr->Mid_proc = here_.addr_;
                    hdr->Msn = sn;
                    hdr->Minc = inc;
                              rec_line=sn;

                    hdr->Mrec_line = rec_line;
                     irset.push_back(make_pair(inc,rec_line));
                    hdr->mirset=irset;
send(pkt, 0);     //envoyer le packet

return (TCL_OK); }

 }
          return (Agent::command(argc, argv));

}


//*********************************************************
//****************msAgent::recv*****************************************
//*********************************************************
void msAgent::recv(Packet* pkt, Handler*)
{
  hdr_message* hdr = hdr_message::access(pkt);//
vector<pair<int, int>> temp1,temp;
int inc1,inc2 ,rec1, rec,in,rec2;
buf mes;
vector<pair<int, int>> soust1,soust2,soust3,soust4;
if(hdr->type=='M'){


sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);
sort (irset.begin(),irset.end(),myfunction1);
   if ( includes(irset.begin(),irset.end(),hdr->mirset.begin(),hdr->mirset.end(), myfunction2) && !hdr-
>mirset.empty() ){
sort (irset.begin(),irset.end(),myfunction1);
sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);
      std::set_difference (irset.begin(), irset.end(), hdr->mirset.begin(), hdr->mirset.end(),
inserter(soust,soust.begin()),myfunction);
      //minset(soust,inc,rec);
```

```
if (!soust.empty()){
sort (soust.begin(),soust.end(),myfunction1);
in=soust[0].first;
rec=soust[0].second;}

          if (hdr->Msn >rec){
                    mes.csn= sn;
                    mes.msn=hdr->Msn;
                    mes.pid=hdr->Mid_proc;
logmsg.push_back(mes);
nb_sauv_=nb_sauv_+1;// enregistrer message
nb_retard_=nb_retard_+1;
                    t=t+1;
                    }else{
nb_annul_=nb_annul_+1; //message dupliqué
                              }       //message de recouvrement
          } else if (hdr->mirset==irset){
                    if (hdr->Msn < sn){
                  mes.csn= sn;
                              mes.msn=hdr->Msn;
                              mes.pid=hdr->Mid_proc;
                              logmsg.push_back(mes);
                    nb_sauv_=nb_sauv_+1;} //message enregistrer
          else if (hdr->Msn > sn){

                    nb_force_=nb_force_+1; //prendre un point de reprise forcé
                              //nb_fr_=nb_fr_+1;
                              sn = hdr->Msn;
                              checkpoint.push_back(sn);

      }
          } else {
sort (irset.begin(),irset.end(),myfunction1);
sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);

          std::set_difference     (hdr->mirset.begin(),     hdr->mirset.end(),     irset.begin(),     irset.end(),
inserter(soust1,soust1.begin()),myfunction);
                    //minset(soust1,inc1,rec1);
if (!soust1.empty()){
sort (soust1.begin(),soust1.end(),myfunction1);
            inc1=soust1[0].first;
            rec1=soust1[0].second;}

          //if (!inclu(irset,hdr->mirset)){
sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);
sort (irset.begin(),irset.end(),myfunction1);
          if ( includes(hdr->mirset.begin(),hdr->mirset.end(),irset.begin(),irset.end(), myfunction2) ){
//soustraction irset-hdr->mirset
```

```
            //  sous(irset,hdr->mirset,soust2);
sort (irset.begin(),irset.end(),myfunction1);
sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);

        std::set_difference (irset.begin(), irset.end(), hdr->mirset.begin(), hdr->mirset.end(),
inserter(soust2,soust2.begin()),myfunction);

        //  minset(soust2,inc2,rec2);
if (!soust2.empty()){
sort (soust2.begin(),soust2.end(),myfunction1);
        inc2=soust2[0].first;
        rec2=soust2[0].second;}
                        if ((rec1 < rec2) || ((rec1=rec2) & (inc1 < inc2))){
                                for (int i=0; i<irset.size();i++){
                                        if  ((irset[i].first==inc2)&(irset[i].second==rec2)){  //supprimer
inc2 rec2 de irset "inc_rec_set"
                                        irset.erase(irset.begin()+i);}
                                        }
                                }
else if(hdr->Msn<rec2){
                mes.csn= sn;
                mes.msn=hdr->Msn;
                mes.pid=hdr->Mid_proc;
logmsg.push_back(mes);
nb_sauv_=nb_sauv_+1; //enregistrer le message
nb_retard_=nb_retard_+1;
                t=t+1;
goto last;// message en transit
            } else {
                        nb_annul_=nb_annul_+1;
                        goto last;//message dupliqué
            }
                rec_line=rec1;
                inc=inc1;
                irset.push_back(make_pair(inc,rec_line));
                //Rollback(rec_line);
                int i;
                i=0;
                char find;
                find='f';
                if (rec_line > sn){                     //prendre un point de reprise forcé
                        nb_fr_=nb_fr_+1;
                        //nb_force_=nb_force_+1;
                        sn = rec_line;
                        checkpoint.push_back(sn);
                }else{  i=0;                    // trouver un point de reprise avec sn >= rec_line
                        while ((i<checkpoint.size())&&(find=='f'))
                        {
```

```
                        if(checkpoint[i]>=rec_line)
                                {
                                find='t';
                                sn=checkpoint[i];
                                }
                        else i=i+1;
                        }
                if (find=='t'){+-

                if(dist<checkpoint.size()-1-i){dist=checkpoint.size()-1-i;}
                deleted=deleted+ checkpoint.size()-1-i;
if (i<checkpoint.size()){ checkpoint.erase (checkpoint.begin()+i,checkpoint.end());}
        }
                }
                //repaly

                for(int i = 0; i < logmsg.size();i++){
                        if (logmsg[i].msn<rec_line)
                        {
                                nb_msg_ = nb_msg_ +1;
                        }
                        else {
                        logmsg.erase (logmsg.begin()+i);}
                        }
                //Catch(hdr->mirset,irset,sn,rec_line,inc);
                        //Min(hdr->mirset,irset,irset);

int x1;
int y1;
int x2;
int y2;
                //vector<pair<int, int>> temp;
if (hdr->mirset==irset){
irset=hdr->mirset;
                }
else {
for(int i = 0; i < hdr->mirset.size(); i++){
if  ((!hdr->mirset.empty())&&(!irset.empty()))
                {
                //minset(s,x1,y1);
sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);
x1=hdr->mirset[0].first;
                y1=hdr->mirset[0].second;
                //minset(s,x2,y2);}}
sort (irset.begin(),irset.end(),myfunction1);
x2=irset[0].first;
                y2=irset[0].second;}}

if ((y1 < y2) || ((y1==y2) & (x1 < x2))){
```

```
                temp.push_back(make_pair(x1, y1));
                }
else {
                temp.push_back(make_pair(x2, y2));}
                //supprimer de s et n
for (int i=0; i<hdr->mirset.size();i++)
                {if ((hdr->mirset[i].first==x1)&(hdr->mirset[i].second==y1)){
hdr->mirset.erase(hdr->mirset.begin()+i);}
                }
                }

for (int i=0; i<irset.size();i++)
                {if ((irset[i].first==x2)&(irset[i].second==y2)){
irset.erase(irset.begin()+i);}
                }
                //unnion temps s et n
temp.insert(temp.end(),hdr->mirset.begin(),hdr->mirset.end());
temp.insert(temp.end(),irset.begin(),irset.end());
sort (temp.begin(), temp.end(), myfunction1);
irset=temp;


//fin de min
                        rec_line=irset[irset.size()-1].second;
                        inc=irset[irset.size()-1].first;
                        if (sn<rec_line){
                                nb_force_=nb_force_+1;
                                nb_fr_=nb_fr_+1;
                                sn=rec_line;
                        }
last: ;
      }
}
}
//*************************************************************************
//          reception d'un message de recouvrement
//*************************************************************************

else if(hdr->type=='R'){

          sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);
sort (irset.begin(),irset.end(),myfunction1);
   if ( !includes(irset.begin(),irset.end(),hdr->mirset.begin(),hdr->mirset.end(), myfunction2) && !hdr-
>mirset.empty() ){

sort (irset.begin(),irset.end(),myfunction1);
sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);

      std::set_difference (hdr->mirset.begin(), hdr->mirset.end(), irset.begin(), irset.end(),
inserter(soust3,soust3.begin()),myfunction);
if (!soust3.empty()){
sort (soust3.begin(),soust3.end(),myfunction1);
```

```
      inc1=soust3[0].first;
      rec1=soust3[0].second;}
sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);
                                    sort (irset.begin(),irset.end(),myfunction1);
                                    if                    (                    !includes(hdr->mirset.begin(),hdr-
>mirset.end(),irset.begin(),irset.end(), myfunction2) && !hdr->mirset.empty() ){
sort (irset.begin(),irset.end(),myfunction1);
sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);

            std::set_difference (irset.begin(), irset.end(), hdr->mirset.begin(), hdr->mirset.end(),
inserter(soust4,soust4.begin()),myfunction);
if (!soust4.empty()){
sort (soust4.begin(),soust4.end(),myfunction1);
            inc2=soust4[0].first;
            rec2=soust4[0].second;}
if ((rec1 < rec2) || ((rec1=rec2) & (inc1 < inc2))){
                                    for (int i=0; i<irset.size();i++){
                                          if  ((irset[i].first==inc2)&(irset[i].second==rec2)){  //supprimer
inc2 rec2 de irset "inc_rec_set"
                                          irset.erase(irset.begin()+i);}
                                          }
}else{
goto last1;
                }
                  rec_line=rec1;
                  inc=inc1;
                  irset.push_back(make_pair(inc,rec_line));
                  int i;
                  i=0;
                  char find;
                  find='f';
                  if (rec_line > sn){          //prendre un point de reprise forcé
                        nb_fr_=nb_fr_+1;
                        //nb_force_=nb_force_+1;
                        sn = rec_line;
                        checkpoint.push_back(sn);
                  }else{   i=0;                // trouver un point de reprise avec sn >= rec_line
                        while ((i<checkpoint.size())&&(find=='f'))
                        {
                        if(checkpoint[i]>=rec_line)
                              {
                              find='t';
                              sn=checkpoint[i];
                              }
                        else i=i+1;
                        }
```

```
                    if (find=='t'){

                    if(dist<checkpoint.size()-1-i){
dist=checkpoint.size()-1-i;}
                    deleted=deleted+ checkpoint.size()-1-i;
if (i<checkpoint.size()){ checkpoint.erase (checkpoint.begin()+i,checkpoint.end());}
          }

                    }
                    //repaly
                    for(int i = 0; i < logmsg.size();i++){
                              if (logmsg[i].msn<rec_line)
                              {
                                        nb_msg_ = nb_msg_ +1;
                              }
                              else {
                              logmsg.erase (logmsg.begin()+i);}
                              }
                    int x1;
int y1;
int x2;
int y2;
                  //vector<pair<int, int>> temp1;
if (hdr->mirset==irset){
irset=hdr->mirset;
                    }
else {
for(int i = 0; i < hdr->mirset.size(); i++){
if  ((!hdr->mirset.empty())&&(!irset.empty()))
                    {
                    //minset(s,x1,y1);
sort (hdr->mirset.begin(),hdr->mirset.end(),myfunction1);
x1=hdr->mirset[0].first;
                    y1=hdr->mirset[0].second;
                    //minset(s,x2,y2);}}
sort (irset.begin(),irset.end(),myfunction1);
x2=irset[0].first;
                    y2=irset[0].second;}}

if ((y1 < y2) || ((y1==y2) & (x1 < x2))){
                    temp1.push_back(make_pair(x1, y1));
                    }
else {
                    temp1.push_back(make_pair(x2, y2));}
                    //supprimer de s et n
for (int i=0; i<hdr->mirset.size();i++)
                    {if ((hdr->mirset[i].first==x1)&(hdr->mirset[i].second==y1)){
hdr->mirset.erase(hdr->mirset.begin()+i);}
```

```
                }
                }

for (int i=0; i<irset.size();i++)
                {if ((irset[i].first==x2)&(irset[i].second==y2)){
irset.erase(irset.begin()+i);}
                }
                //unnion temps s et n
temp1.insert(temp1.end(),hdr->mirset.begin(),hdr->mirset.end());
temp1.insert(temp1.end(),irset.begin(),irset.end());
sort (temp1.begin(), temp1.end(), myfunction1);
irset=temp1;

//fin de min
                        rec_line=irset[irset.size()-1].second;
                        inc=irset[irset.size()-1].first;
                        if (sn<rec_line){
                                nb_force_=nb_force_+1;
                                nb_fr_=nb_fr_+1;
                                sn=rec_line;
                        }
                  }
//fin
```

Withfault.tcl :

```
#############################################################################################
##########
#    L'ouverture de fichier rollback.txt qui contient les informations de simulation          #
#############################################################################################
##########
set r [open "rollback.txt" w]

#******************Demande de l'utilisateur pour entrer le nbre de processus***************#
puts "Entrez le nombre de processus de votre simulation : "
gets stdin nb_node_
puts $r "$nb_node_"

#*****************Demande de l'utilisateur pour entrer la durée de simulation**************#
puts "Entrez le temps de simulation : "
gets stdin tps
puts $r "$tps"

#****************Demande de l'utilisateur pour entrer le nombre de messages**********#
puts "Entrez le nombre de message : "
gets stdin nb_msg_
set nb_msg_ [expr $nb_msg_*$nb_node_]
```

```
puts $r "$nb_msg_"

#*****************Demande de l'utilisateur pour entrer le nombre du fauts**************#
puts "Identifiez le nombre du fautes : "
gets stdin nb_faut
puts $r "$nb_faut"

#****Demande de l'utilisateur pour entrer le nombre d'événements pour faire point basic****#
puts "Aprés combien d'évènements faire point basic ? "
gets stdin nb_eve_
puts $r "$nb_eve_"

#************Donner chaque processus un identifiant et numéro de séquence initial***********#
for {set j 0} {$j < $nb_node_} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
                                        switch $k {
                                 # identifiant de processus
                                0 { set table($j,$k) $j }
                                 # initialisé sn de chaque site  à '0'
1 { set table($j,$k) 0 }
                           }
puts $r "$table($j,$k)"
        }
}
close $r
##############################################################################################
##########
##############################################################################################
##########
#             Création de scénario d'envoie de message                          #
##############################################################################################
##########
##############################################################################################
##########

##############################################################################################
##########
#                    L'accés au fichier rollback.txt                                        #
##############################################################################################
##########
set r [open "rollback.txt" "r"]
set nb_node_ [gets $r]
set tps [gets $r]
set nb_msg_ [gets $r]
set nb_faut [gets $r]
set nb_eve_ [gets $r]
```

```
close $r
#############################################################################################
##########
#                   Prendre les temps pour faire les envoies des messages                   #
#############################################################################################
##########
set tcom  $tps

for {set i 0} {$i < $nb_msg_} {incr i} {
set exp [expr rand()*$tcom]
        for {set j 0} {$j < 2} {incr j} {
                                        switch $j {
                                    0 { set envoi_table_($i,$j) $i }
1 { set envoi_table_($i,$j) $exp }
                                }
    }
}
#############################################################################################
##########
#                   Tri de tableau des envois des messages par décalage                     #
#############################################################################################
##########
for {set i 0} {$i <[expr $nb_msg_-1]} {incr i} {
        for {set j [expr $i+1]} {$j <$nb_msg_} {incr j} {
                if { $envoi_table_($i,1) > $envoi_table_($j,1)} {
                        set interr $envoi_table_($i,1)
            set envoi_table_($i,1) $envoi_table_($j,1)
            set envoi_table_($j,1) $interr
         }
    }
}
#############################################################################################
##########
#                   L'ouverture de fichier scenario.txt qui contient                        #
#         les informations d'envoies et de réceptions des messages                          #
#############################################################################################
##########
set s [open "scenario.txt" w]

for {set j 0} {$j <$nb_msg_} {incr j} {
        set src_ 0
        set dest_ 0

        while { $src_ == $dest_} {
                set src_ [expr int(rand()*$nb_node_)]
                set dest_ [expr int(rand()*$nb_node_)]
        }
        for {set k 0} {$k < 3} {incr k} {
                                        switch $k {
```

```
                                0 { set scenario($j,$k) $envoi_table_($j,1) }
                                1 { set scenario($j,$k) $src_ }
                                2 { set scenario($j,$k) $dest_ }
                        }
        puts $s "$scenario($j,$k)"
        }
}
close $s
#############################################################################################
##########
#               L'ouverture de fichier panne.txt qui contient                   #
#       les processus qui tombent en panne et le moment de panne                        #
#############################################################################################
##########
set pa [open "panne.txt" w]
set id_panne 0
set taille 0
set inter [expr $tps/$nb_faut]

set j 0

set arrivalRNG [new RNG]
set arrival_ [new RandomVariable/Exponential]
$arrival_ set avg_ $inter
$arrival_ use-rng $arrivalRNG
set time [$arrival_ value]

set idRNG [new RNG]
set id [new RandomVariable/Uniform]
$id set min_ 0
$id set max_ [expr $nb_node_ -1]
$id use-rng $idRNG

        while { $time < $tps} {

                for {set k 0} {$k < 2} {incr k} {
                                        switch $k {
                                        0 { set panne($j,$k) $time }
                                        1 { set panne($j,$k) [expr round([$id value])]}
                                }
                puts $pa "$panne($j,$k)"
                }

        set time [expr $time+[$arrival_ value]]
        set taille [expr $taille+1]
        incr j
        }
close $pa
```

```
#############################################################################################
##########
#                          L'ouverture de fichier basic.txt                          #
#############################################################################################
##########
set ba [open "basic.txt" w]
set time 0.02
set long1 0
set long2 0
for {set j 0} {$j < $nb_node_} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
                                switch $k {
        # identifiant de processus
                        0 { set even_table($j,$k) $j }
                        # even de caque processus initialisé à '0'
1 { set even_table($j,$k) 0 }
                                }
        }
}

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

        set src $scenario($i,1)
        incr even_table($src,1)

        set des $scenario($i,2)
        incr even_table($des,1)

        set heurs $scenario($i,0)

        if {$even_table($src,1)==$nb_eve_} {

                set even_table($src,1) 0
                for {set k 0} {$k < 2} {incr k} {
                                        switch $k {
                                        0 { set basic($i,$k) $heurs }
                                        1 { set basic($i,$k) $src }
                                }
                puts $ba "$basic($i,$k)"
                }
        set long1 [expr $long1 +1]
        }
        if {$even_table($des,1)==$nb_eve_} {
                set even_table($des,1) 0
                for {set k 0} {$k < 2} {incr k} {
                                        switch $k {
                                        0 { set basic($i,$k) $heurs }
                                              1 { set basic($i,$k) $des }
```

```
                                        }
                puts $ba "$basic($i,$k)"
                }
        set long2 [expr $long2 +1]
        }
}
close $ba
#############################################################################################
##########
#                                L'accés au fichier basic.txt                               #
#############################################################################################
##########
set ba [open "basic.txt" "r"]
set long [expr $long1 +$long2]
for {set j 0} {$j < $long} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
        set basic($j,$k) [gets $ba]
        }
}
close $ba
#############################################################################################
##########
#                                L'accés au fichier panne.txt                               #
#############################################################################################
##########
set pa [open "panne.txt" "r"]
for {set j 0} {$j < $taille} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
        set panne($j,$k) [gets $pa]
        }
}
close $pa
#############################################################################################
##########
#                        L'ouverture de fichier global.txt qui regroupe                     #
#                        tous les moments dans un tableau glob_tp                           #
#############################################################################################
##########
set global [open "global.txt" w]
for {set i 0} {$i < $nb_msg_} {incr i} {
                        set envoi $scenario($i,0)
for {set j 0} {$j < 2} {incr j} {
                        switch $j {
        0 { set glob_tp($i,$j) $envoi}
        1 { set glob_tp($i,$j) 1 }
                        }
puts $global "$glob_tp($i,$j)"
}
}
```

```
for {set i 0} {$i < $taille} {incr i} {
                                        set pan $panne($i,0)
for {set j 0} {$j < 2} {incr j} {
                                        switch $j {
          0 { set glob_tp($i,$j) $pan}
                        1 { set glob_tp($i,$j) 3 }
          }
puts $global "$glob_tp($i,$j)"
}
}

for {set i 0} {$i < $long} {incr i} {
                                        set bas $basic($i,0)
for {set j 0} {$j < 2} {incr j} {
                                        switch $j {
                        0 { set glob_tp($i,$j) $bas}
                        1 { set glob_tp($i,$j) 2 }
          }
puts $global "$glob_tp($i,$j)"
}
}
close $global
#############################################################################
#                                        L'ouverture de fichier final.txt                                #
#############################################################################
set fi [open "final.txt" w]
#*********************************L'accés au fichier global.txt*********************#
set global [open "global.txt" "r"]
set tot [expr $nb_msg_ + $long]
set total [expr $taille +$tot]
puts $fi "$total"
puts $fi "$long"
puts $fi "$taille"
for {set j 0} {$j <= $total} {incr j} {
          for {set k 0} {$k < 2} {incr k} {
          set glob_tp($j,$k) [gets $global]

          }
}
#*********************************Tri de tableau glob_tp**************************#
for {set i 0} {$i <=$total} {incr i} {
          for {set j [expr $i+1]} {$j <=$total} {incr j} {
                    set x $glob_tp($i,0)
                    set y $glob_tp($j,0)
```

```
        if { $x > $y } {
                        set var $glob_tp($i,0)
set glob_tp($i,0) $glob_tp($j,0)
set glob_tp($j,0) $var

                        set val $glob_tp($i,1)
set glob_tp($i,1) $glob_tp($j,1)
set glob_tp($j,1) $val
                }
        }
puts $fi "$glob_tp($i,0)"
puts $fi "$glob_tp($i,1)"
}
close $global
close $fi
```

ms.tcl

```
#########################################################################################
########
#                                   Création d'un simulateur                            #
#########################################################################################
########
set ns [new Simulator]
set bw rolms.dat
set f0 [open $bw w]

#########################################################################################
########
#                   L'ouverture de fichier trace                                        #
#########################################################################################
########
set nf [open l.nam w]
$ns namtrace-all $nf
#########################################################################################
########
#                                   Define a 'finish' procedure                         #
#########################################################################################
########
proc finish {} {
global ns nf f0 nb_node_ p  nmsg  del delr ackr c basi forced ck sauv annul retard ack deleted  sk rec
forcplus dis

        for {set i 0} {$i < $nb_node_} {incr i} {
        set n [$p($i) set agent_addr_]
        set l [$p($i) set sn]
        set a [$p($i) set nb_basic_]
        set b [$p($i) set nb_force_]
        set c [$p($i) set deleted]
        set d [$p($i) set nb_sauv_]
```

```
        #set j [$p($i) set nb_ack_]
        set k [$p($i) set recieved]
        #set md [$p($i) set skipped]
        set ind [$p($i) set indice]
        set nbf [$p($i) set nb_fr_]
        set dist_ [$p($i) set dist]
        set del [expr [$p($i) set deleted]+$del]
        set basi [expr [$p($i) set nb_basic_]+$basi]
        set forced [expr [$p($i) set nb_force_]+$forced]
        set nmsg [expr [$p($i) set nb_msg_]+$nmsg]
        set sauv [expr [$p($i) set nb_sauv_]+$sauv]
        #set ack [expr [$p($i) set nb_ack_]+$ack]
        set forcplus [expr [$p($i) set nb_fr_]+$forcplus]
        if {$dis < $dist_} {
                set dis $dist_}
        }
        set delr [expr [expr double($del)]/ [expr [expr $basi+$forced]+$forcplus]]
        set ackr [expr [expr double($ack)]/$sauv]
        puts $f0 "#####################################"
        puts $f0 "$delr est del sur basic et forced "
        puts $f0 "$ackr est ack sur sauv "
        puts $f0 "$dis est le rollback distance"
        puts $f0 "##############################"
puts $f0 "$del est le nombre de points supprimés"
        puts $f0 "$basi est le nombre de points spontanés"
        puts $f0 "$forced est le nombre de points forcés"
        puts $f0 "$sauv est le nombre de messages sauvegardés "
        #puts $f0 "$ack est le nombre d'ack "
        puts $f0 "$forcplus est le nombre de points forcés à cause de recouvrement "
$ns flush-trace
        #Close the trace file
close $nf
        #Execute nam on the trace file
    #exec nam ck.nam &
exit 0
}

################################################################################################
########
#                   Procédure permet d'afficher le processus qui prend un point forcé        #
################################################################################################
########
proc forced {p q n} {
set ns [Simulator instance]
set nowe [$ns now]
set time 0.1
set d [$p set sn]
set b [$q set sn]
```

```
if { $d > $b} {
$ns at $nowe "$n color red"
		$ns at $nowe "$n label \" point force ($d) \""
		$ns at [expr $nowe+0.01] "$n color black"
		$ns at [expr $nowe+0.01] "$n label \" \""
}
}
##############################################################################################
########
#	Procédure permet d'afficher le processus qui prend un point spontané		#
##############################################################################################
########
proc basicinitial {p n} {
set ns [Simulator instance]
set nowe [$ns now]
$ns at $nowe "$p take-initial"
set b [$p set sn]
$ns at $nowe "$n color blue"
$ns at $nowe "$n label \" point basic ($b) \""
$ns at [expr $nowe+0.01] "$n color black"
$ns at [expr $nowe+0.01] "$n label \" \""
}

proc basic {p n} {
set ns [Simulator instance]
set nowe [$ns now]
$ns at $nowe "$p take-basic"
set b [$p set sn]
set b [expr $b+1]
$ns at $nowe "$n color blue"
$ns at $nowe "$n label \" point basic ($b) \""
$ns at [expr $nowe+0.01] "$n color black"
$ns at [expr $nowe+0.01] "$n label \" \""
}
##############################################################################################
########
#			Procédure permet d'afficher le processus qui tombe en panne		#
##############################################################################################
########
proc panne {p n } {
global inter tps nb_faut
set ns [Simulator instance]
set nowe [$ns now]
set time 0.01
set b [$p set agent_addr_]
			$ns at $nowe "$n color brown"
		$ns at $nowe "$n label \" p($b) en panne  \""
			$ns at [expr $nowe+$time] "$n color black"
		$ns at [expr $nowe+$time] "$n label \" \""
}
```

```
#########################################################################################
########
#                              procédure pour diffuser le message de Rollback                                  #
#########################################################################################
########
proc diffusion {p n} {
global nb_node_
set ns [Simulator instance]
set nowe [$ns now]
set b [$p set agent_addr_]

for {set i 0} {$i < $nb_node_} {incr i} {
        if { $b != $i} {
        $ns at $nowe "$p set dst_addr_ $i"
        $ns at $nowe "$p send-rollback"
        }
}
}
#*************************************************************************
#########################################################################################
########
#                                      L'accés au fichier rollback.txt                          #
#########################################################################################
########
set r [open "rollback.txt" "r"]
set nb_node_ [gets $r]
set tps [gets $r]
set nb_msg_ [gets $r]
set nb_faut [gets $r]
set nb_eve_ [gets $r]

for {set j 0} {$j < $nb_node_} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
        set table($j,$k) [gets $r]
        }
}
close $r
#########################################################################################
########
#                              L'accés au fichier scenario.txt                                  #
#########################################################################################
########
set s [open "scenario.txt" "r"]
for {set j 0} {$j < $nb_msg_} {incr j} {
        for {set k 0} {$k < 3} {incr k} {
        set scenario($j,$k) [gets $s]
}
```

```
}
close $s
#########################################################################################
########
#                                L'accés au fichier final.txt                                  #
#########################################################################################
########
set fi [open "final.txt" "r"]
set total [gets $fi]
set long [gets $fi]
set taille [gets $fi]
for {set j 0} {$j <= $total} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
        set glob_tp($j,$k) [gets $fi]

}
  }
close $fi
#########################################################################################
########
#                                L'accés au fichier basic.txt                                  #
#########################################################################################
########
set ba [open "basic.txt" "r"]
for {set j 0} {$j < $long} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
        set basic($j,$k) [gets $ba]
        }
}
close $ba
#########################################################################################
########
#                                L'accés au fichier panne.txt                                  #
#########################################################################################
########
set pa [open "panne.txt" "r"]
for {set j 0} {$j < $taille} {incr j} {
        for {set k 0} {$k < 2} {incr k} {
        set panne($j,$k) [gets $pa]
        }
}
close $pa
```

```
#############################################################################################
*****************************Identification du couleur de paquet*********************#
$ns color 0 Green
$ns color 1 Blue
$ns color 2 Red

#*******************************Création des noeuds*******************************#
for {set i 0} {$i <$nb_node_} {incr i} {
set n($i) [$ns node]
}

#*********Création d'une ligne de communication "full duplex" entre les noeuds***********#

#****Création d'un agent MS devenue comme un générateur de paquet pour chaque noeud ****#
for {set i 0} {$i <$nb_node_} {incr i} {
                                        set p($i) [new Agent/ms]
                                        $ns attach-agent $n($i) $p($i)
}

#***********************Connection entre deux agents*****************************#
for {set i 0} {$i <$nb_node_} {incr i} {
        for {set j 0} {$j < $nb_node_} {incr j} {
                if  {$i != $j} {
                        $ns duplex-link $n($i) $n($j) 1Mb 10ms DropTail
                        $ns connect $p($i) $p($j)
                }
        }
$p($i) set inc 0
$p($i) set nb_node_ $nb_node_
$p($i) set agent_addr_ $i
}

#*************************Initialisation des variables utilisés************************#
set nmsg 0
set del 0
set basi 0
set forced 0
set sauv  0
set annul 0
set retard 0
set ack 0
set now 0
set time 0.02
set j 0
set k 0
set ind 0
```

```
set sk 0
set rec 0
set forcplus 0
set dis 0
##########################################################################################
########

##########################################################################################
########
#          Chaque processus peut prendre son point de reprise spontané initial          #
##########################################################################################
########
for {set i 0} {$i < $nb_node_} {incr i} {
        $ns at 0.0 "basicinitial $p($i) $n($i)"
}
##########################################################################################
########
#                Parcours de tableau "glob_tp" trié qui contient les moments           #
#                et l'accés aux fichiers utilisés selon leurs types "1,2 ou 3"         #
##########################################################################################
########
for {set i 0} {$i <= $total} {incr i} {
        set heur $glob_tp($i,0)
        set var $glob_tp($i,1)

        if {$var == 1} {

                set src $scenario($j,1)
                set des $scenario($j,2)
                $ns at $heur "$p($src) set dst_addr_ $des"
                $ns at $heur "$p($src) send-message"
                $ns at $heur "forced $p($src) $p($des) $n($des)"
                incr j
        }

        if {$var == 2} {
                set pre $basic($k,1)
                $ns at [expr $heur+$time] "basic $p($pre) $n($pre)"
                incr k
        }

        if {$var == 3} {
                set id_panne $panne($ind,1)
                $ns at $heur "panne $p($id_panne) $n($id_panne) "
                $ns at $heur "diffusion $p($id_panne) $n($id_panne) "
                        incr ind
        }

}
```

```
#############################################################################################
#######
#                    Appel de procédure "finish" aprés un temps "tps" de simulation                    #
#############################################################################################
########
$ns at $tps "finish"
#############################################################################################
########
#                                                lancer la simulation                                    #
#############################################################################################
########
$ns run
```

Annexe 2

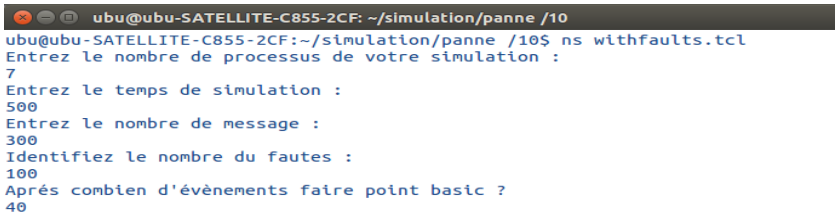
Mode d’utilisation

Introduction :

Pour mieux comprendre et faciliter à l'utilisateur de bien simulé les algorithmes réalisés, nous présentons cette annexe comme mode d'utilisation avec les étapes nécessaires pour lancer la simulation.

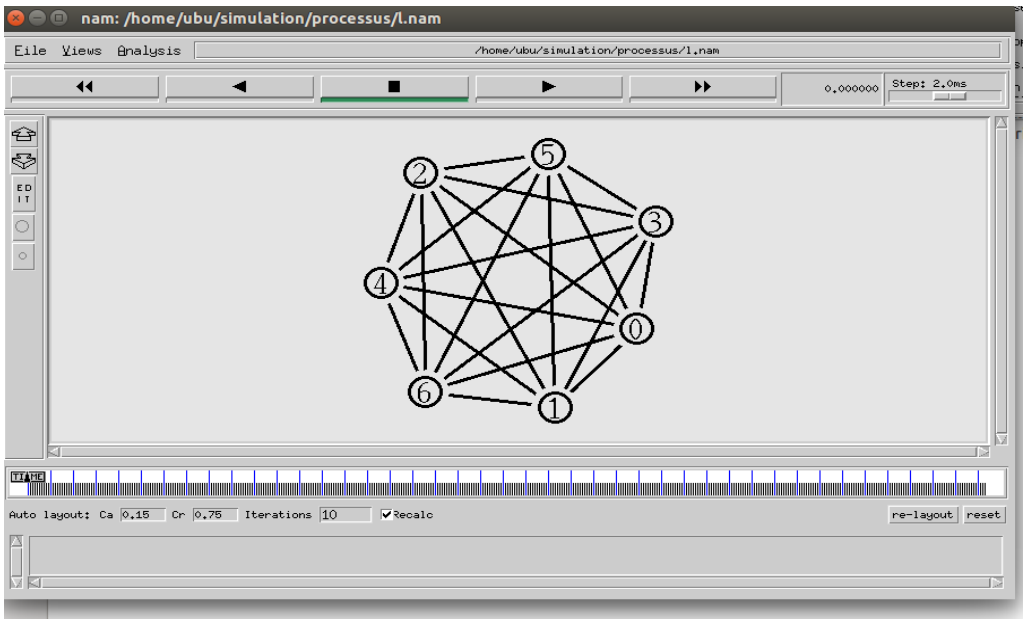
Pour tester notre agent et avant de lancer la simulation on doit d'abord executer le programme «withfault.tcl».

La figure présente une capture écran de la fenêtre qui se lance après avoir lancé le scenario «withfault.tcl» qui va nous permettre de saisir : le nombre de processus, le temps de simulation, le nombre de message,le nombre de panne, la longueur d'intervalle entre les points de reprise.



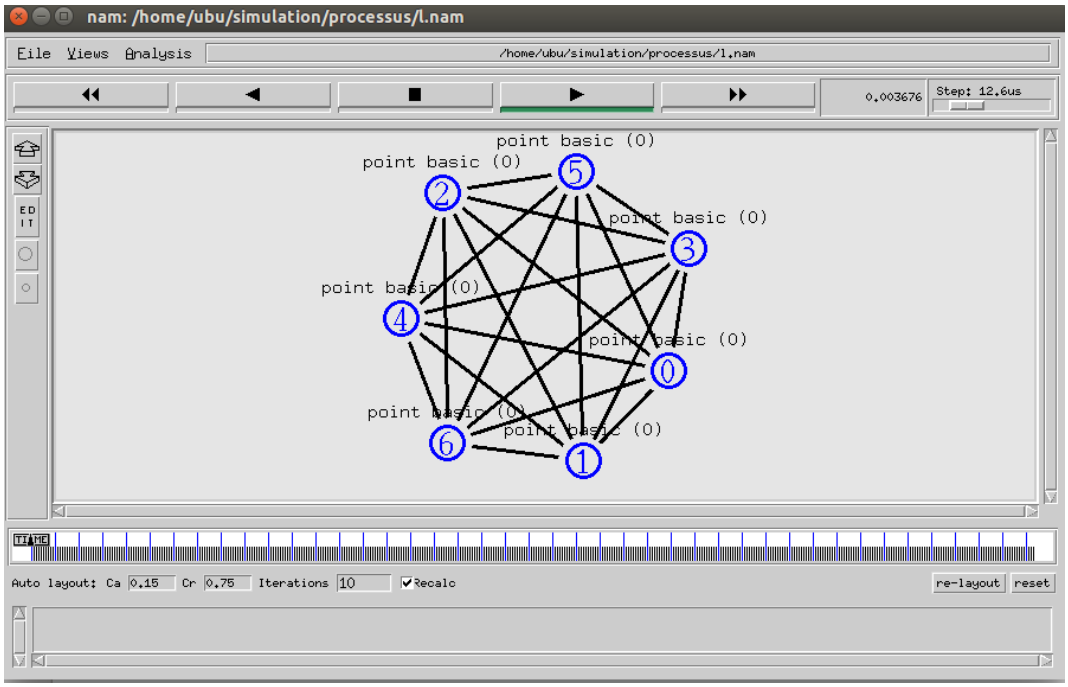
FigureA2. 1 saisie des paramètres de simulation (avec plusieurs fautes)

Apres la saisie des paramètres de simulation on tape «ms.tcl»

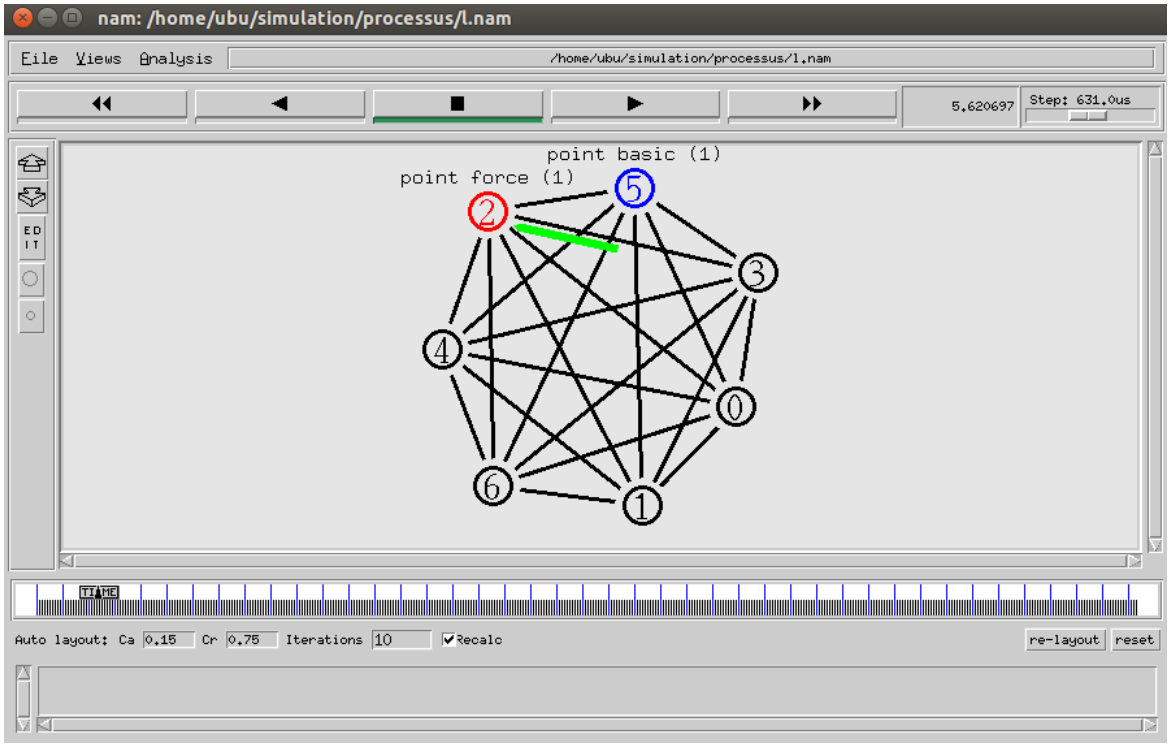


FigureA2. 2 visualisation de l'algorithme MS

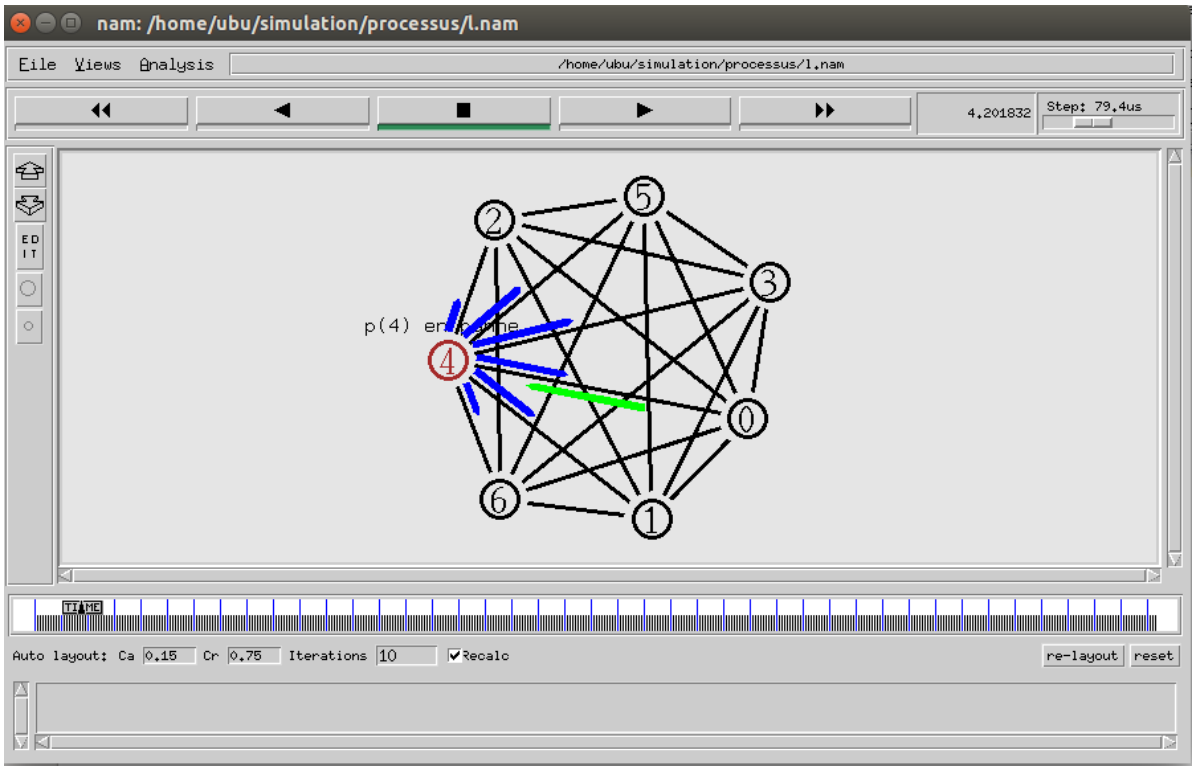
On lance la simulation en cliquant sur le bouton de début de simulation



FigureA2. 3 prendre un point de reprise spontané inital



FigureA2. 4 P5 prend un point de reprise spontané et P2 prend un point de reprise forcé



FigureA2. 5 P4 tombe en panne et diffuse un message de recouvrement

Remarque : On adapte les même étapes pour l'algorithme HMR.