Table des matières

Liste des tableaux

Liste des figures

Introduction Générale

De nos jours, les systèmes informatiques sont en passe de s'intégrer à notre environnement quotidien à un degré jamais égalé jusqu'alors et sont devenus par nature distribués ; l'usage d'Internet se démocratise, le commerce électronique est en plein essor, les bourses européennes et mondiales se regroupent en réseaux, etc. Les applications développées sur ces systèmes distribués, si elles ne sont pas critiques en termes de vies humaines, le sont souvent d'un point de vue économique. La disponibilité de ces applications est très importante car leurs usagers sont répartis sur toute la surface du globe, et par conséquent, les services qu'elles assurent doivent souvent être rendus sans discontinuer, 24 heures sur 24. Néanmoins, la disponibilité de tels systèmes ne peut être assurée qu'en utilisant des techniques de sûreté de fonctionnement, et plus particulièrement de tolérance aux fautes.

De ce fait, l'exigence de la tolérance aux fautes est devenue incontournable dans la conception des supports d'exécution répartie. De part l'évolution technologique, des études ont montré que le coût des défaillances dans l'industrie informatique croit sans cesse. En effet les environnements distribués sont particulièrement *sensibles aux fautes* (dysfonctionnement). La défaillance, même transitoire, d'un des sites entraîne souvent l'échec de l'application ou une incohérence dans les résultats produits.

Il est ainsi essentiel d'évaluer les risques liés à un dysfonctionnement d'une des composantes du système d'information et de prévoir des moyens et mesures permettant d'éviter ou de rétablir dans des temps acceptables tout incident. De nombreux travaux sur les systèmes repartis ont pour but de garantir que la sureté de fonctionnement de ces systèmes n'est pas dégradée par la répartition.

Ainsi avec la prolifération des architectures réparties, l'intégration efficace et transparente des techniques de tolérance aux fautes est devenue *cruciale*. Leur objectif est de permettre à un système de continuer à fournir le service souhaité malgré l'occurrence de fautes. En d'autres termes, la tolérance aux fautes permet à un système d'être fiable, disponible et sûr.

Historiquement, le développement des systèmes répartis a été principalement lié à plusieurs besoins, comme la communication entre entités géographiquement distantes, via des réseaux filaires, l'accélération des calculs suite à l'augmentation des ressources. Sans oublier bien sûr les facteurs économiques comme le fait que plusieurs machines simples sont moins coûteuses qu'une seule machine complexe, à nombre d'utilisateurs égal. Ensuite, il y a eu l'émergence et l'évolution des réseaux sans fil et des équipements qui a abouti au paradigme connu sous le nom « *informatique mobile ou nomade* ». D'une

manière générale, les systèmes mobiles incluent des unités mobiles (assistants personnels, ordinateurs portables…), qui se déplacent tout en étant connectés aux réseaux au moyen de liens sans fil (IEEE 802.11, Bluetooth…) et offrent aux utilisateurs la capacité de pouvoir se déplacer tout en restant connecté aux applications réparties.

De plus, avec les avancées récentes, en termes de performances et de miniaturisation, réalisées en microélectronique, les applications liées au traitement mobile sans fil devraient être de plus en plus répandues et s'étendent à tous les secteurs d'activités. En effet, les réseaux sans fil offrent une grande flexibilité d'emploi. En particulier, ils permettent la mise en réseaux de sites dont le câblage serait trop onéreux à réaliser dans leur totalité, voir même impossible dans certains cas.

Parallèlement aux réseaux mobiles cellulaires, qui s'appuient sur l'existence d'une infrastructure fixe, il y a eu le déploiement des infrastructures pervasives, à base de réseaux mobiles Ad Hoc. Ceci confère une réalité « palpable » à l'informatique ubiquitaire: *le calcul partout et à tout moment*.

Les réseaux mobiles ad hoc sont, eux aussi, des réseaux sans fil. Cependant, à l'inverse des réseaux avec infrastructure fixe, comprenant des stations de base ou points d'accès, les réseaux ad hoc en sont dépourvus. De même, ils ne comportent ni nœud privilégié, ni infrastructure de contrôle centralisée. Ils sont déployés en cas de sinistre (tremblement de terre, incendie, sauvetage..) et peuvent être utiles pour des applications spécifiques aussi bien dans le domaine militaire que civil.

Par conséquent, les réseaux mobiles se distinguent de leur homologues, les réseaux fixes par : la mobilité et la vulnérabilité des stations de travail, les fréquentes déconnexions, les contraintes imposées par les ressources mobiles telles que la bande passante limitée des réseaux sans fils, la durée de vie de la batterie du terminal portable, la capacité mémoire, et la vitesse d'exécution. Ces caractéristiques posent de nouveaux défis en termes de sûreté de fonctionnement, ce qui oblige à changer la vision classique des problèmes liés aux systèmes distribués. En effet, les processus distribués doivent tenir compte de l'apparition de la nouvelle entité: "*unité mobile*" avec toutes ses propriétés. En plus des contraintes liées aux réseaux mobiles avec infrastructure, s'ajoute celles des réseaux mobiles ad hoc qui sont liés à l'absence d'infrastructure fixe de coordination.

En dépit de tous les problèmes mentionnés ci-dessus, un utilisateur mobile souhaite se déplacer librement et continuer à travailler le plus normalement possible avec son terminal mobile. Il est donc souhaitable de fournir une continuité de service malgré les défaillances qui peuvent surgir. Pour cela, on a recours aux techniques de tolérance aux fautes. Plus précisément : le recouvrement arrière (ou la reprise).

Le recouvrement arrière constitue un mécanisme très prometteur pour tolérer les défaillances au cours de l'exécution de l'application. Il est fondé sur la sauvegarde régulière de l'état d'un système appelée point de reprise à partir duquel les tâches peuvent être redémarrées lors d'un échec ; évitant ainsi de recommencer l'exécution depuis le début, surtout s'il s'agit d'un processus long qui a entamé des heures et des heures d'exécution. Un problème central dans les techniques de sauvegarde et de restauration de points de reprise dans un environnement distribué, est la capacité à détecter les dépendances entre processus communiquant pour construire un point de reprise global cohérent.

Si la littérature abonde sur les protocoles de reprise dans les réseaux fixes et les réseaux mobiles avec infrastructure, les réseaux mobiles ad hoc n'ont pas bénéficié de cette attention. Il n'existe pas à notre connaissance, de proposition satisfaisante prenant objectivement en compte les caractéristiques des réseaux ad hoc. C'est ce qui a motivé notre travail, qui consiste à explorer cet environnement peu trivial. La faible largeur de la bande passante du réseau sans fil ne nous permet pas de l'encombrer avec un trafic intensif. De même l'inexistence de station de base fixe (qui résout beaucoup de problèmes dans les réseaux avec infrastructure) et la limitation de la batterie des sites mobiles, ne nous permet pas de faire trop de calcul. Sans oublier la faible capacité des sites mobiles et l'instabilité de leurs supports de stockage par rapport aux stations de base. Face à ces problèmes, nous présentons nos premières investigations pour la conception d'un protocole de checkpointing qui soit adapté aux contraintes des réseaux mobiles en général et des MANET en particulier.

La suite de ce document est constituée de sept chapitres :

Le chapitre 1 : Définit dans leurs grandes lignes les notions de sûreté de fonctionnement, de *faute*, d'*erreur* et de *défaillance*, qui nous permettront de poser la problématique du travail consigné dans ce mémoire. Il introduit la tolérance aux fautes comme un moyen de sûreté de fonctionnement dans les systèmes répartis.

Le chapitre 2 : Présente la problématique de la tolérance aux fautes par recouvrement arrière dans les systèmes répartis, en mettant l'accent sur les aspects essentiels : la cohérence d'état global, le déterminisme d'exécution et les mécanismes de mise en œuvre de la stratégie de reprise.

Le chapitre 3 : Décrit les deux grandes familles de protocoles de tolérance aux pannes par recouvrement arrière, à savoir, les protocoles basés sur les points de reprise et ceux basés sur la journalisation des messages. Deux analyses comparatives ont été présentées: une comparaison d'adaptabilité en fonction de l'application et une autre selon les caractéristiques propres des différentes stratégies.

Le chapitre 4 : présente l'environnement mobile en général, en décrivant les caractéristiques des terminaux portables et la différence entre ces derniers et les stations fixes. Une comparaison a été faite avec les réseaux fixes.

Le chapitre 5 : présente les caractéristiques des réseaux mobile ad hoc en particulier.

Le chapitre 6 : présente les algorithmes de checkpointing dans les réseaux mobiles. Une étude de quelques propositions d'algorithmes a été faite, et une critique a été élaborée pour chaque proposition afin de montrer les points positifs et négatifs de chaque algorithme.

Le chapitre 7 : présente notre contribution, qui consiste en une proposition d'un algorithme de checkpointing adapté aux réseaux mobiles ad hoc. Une preuve analytique a été donnée pour valider notre protocole.

Et en fin de document, une conclusion est donnée pour résumer les apports essentiels de notre travail et ouvrir quelques nouveaux éléments de réflexion et perspectives de recherche.

chapitre 1

1. Introduction

Ce chapitre a pour objectif de présenter le cadre de départ de notre travail. Dans un premier temps, nous définirons dans leurs grandes lignes les notions de sûreté de fonctionnement, de faute, d'erreur et de défaillance, qui nous permettront de poser la problématique de cette thèse. Nous dirons ensuite quelques mots sur les moyens proposés pour réaliser des systèmes sûrs de fonctionnement, puis nous nous attachons à l'un des moyens pour l'obtenir, la tolérance aux fautes, qui reste indispensable pour que le système continue à fonctionner malgré la présence de fautes n'ayant pu être ni éliminées ni prévenues. Nous donnons alors un panorama des techniques pouvant être employées en mettant l'accent sur les particularités des systèmes répartis montrant la complexité du développement d'applications distribuées tolérantes aux fautes.

L'histoire de la sûreté des systèmes fabriqués par l'homme reste à faire, mais elle n'est bien évidemment pas née avec l'informatique. En ne remontant qu'à la révolution industrielle, on trouvait déjà dans les premiers systèmes de signalisation ferroviaire ou les constructions maritimes des considérations élaborées de sûreté de fonctionnement.

Ce mémoire s'intéresse plus particulièrement aux systèmes informatiques. Même s'ils partagent des points communs avec les réalisations industrielles du début du siècle, ceux-ci nécessitent de développer des approches qui leur soient spécifiques. Une tâche essentielle a en particulier consisté à proposer un ensemble de notions claires pour appréhender la sûreté de fonctionnement indépendamment de la nature du système à laquelle elle s'applique.

2. Concepts de base de la sûreté de fonctionnement

La sûreté de fonctionnement d'un système est définie comme :
« La propriété qui permet à ses utilisateurs de placer une confiance justifiée dans la qualité du service qu'il leur délivre. »[1]. Le but de cette discipline est de ″ concevoir, réaliser et utiliser des systèmes informatiques ou la faute est naturelle, prévenue et tolérable″.

On notera dans cette définition l'importance de l'utilisateur, placé immédiatement comme étalon de la sûreté de fonctionnement (« utilisateur » est à prendre ici au sens large: tous ceux à qui le système procure de la valeur ajoutée ; une banque étant autant utilisatrice d'un distributeur automatique que son client).

Selon les applications cibles du système informatique, les différentes facettes de la sûreté de fonctionnement se voient accorder une importance plus ou moins grande. La sûreté de fonctionnement peut être considérée selon des points de vue différents mais complémentaires comme le montre Laprie dans [1]. Dans cette étude, pour la sûreté de fonctionnement d'un système informatique, Laprie propose trois points de vue présentés dans les sous-sections suivantes : les entraves à la sûreté de fonctionnement, les attributs de la sûreté de fonctionnement et les moyens permettant de l'assurer.

2.1 Entraves à la sûreté de fonctionnement

Les fautes, les erreurs et les défaillances sont les causes et les conséquences de la non sûreté de fonctionnement [2].

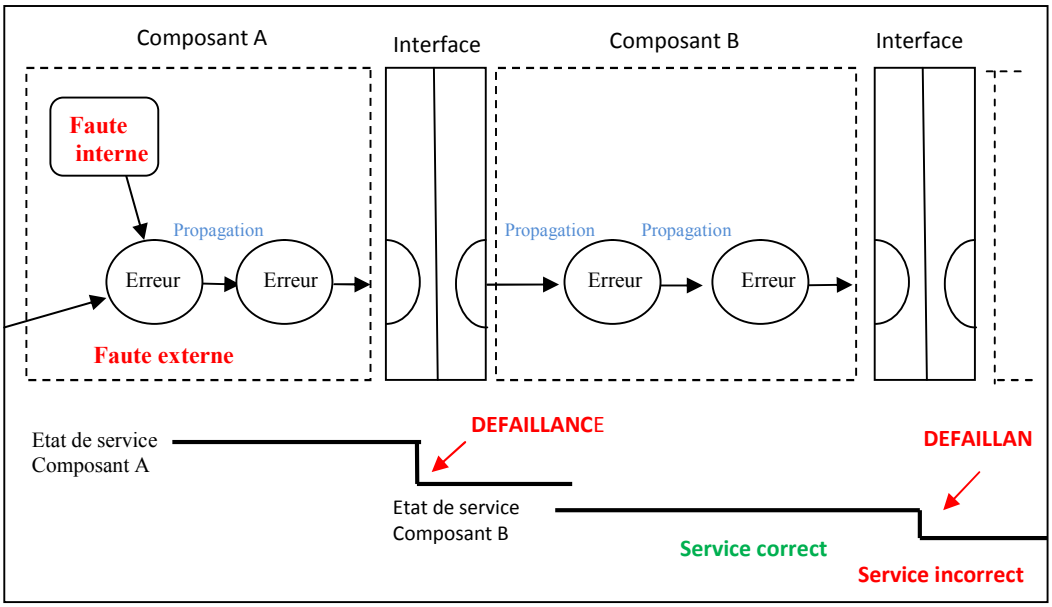


Figure 1. 1 : propagation d'une erreur [2].

Un système défaille (défaillance du système) lorsque le service qu'il délivre diverge du service attendu. La sûreté de fonctionnement cherche donc à éviter les défaillances, ou au moins à prévenir les plus catastrophiques. C'est ainsi qu'on préférera faire arriver les voyageurs d'un train en retard (ce qui est en soi une «défaillance » du système), plutôt que de leur faire risquer une collision mortelle (qui est aussi une défaillance, catastrophique cette fois).

Prévenir les défaillances requiert d'introduire deux nouvelles notions : faute et erreur.

Comme la figure 1.1 l'illustre, la défaillance survient parce que le système a un comportement erroné : une erreur est la partie de l'état du système qui est susceptible d'entraîner une défaillance. La cause adjugée ou supposée de l'erreur est une faute. Une erreur est donc la manifestation d'une faute dans le système, et une défaillance est l'effet d'une erreur sur le service. Ceci conduit à la chaîne fondamentale présentée dans la figure1.2

Fautes : Une faute ou une panne[1] est la cause d'une défaillance, elle est caractérisée par sa nature, son origine et son étendue temporelle [1]. La nature d'une faute précise la manière dont elle a été provoquée: intentionnellement ou accidentellement. La cause de l'apparition d'une faute est révélée par son origine. L'étendue temporelle caractérise la persistance ou la durée d'une faute (temporaire ou permanente).

Erreurs : une erreur est la conséquence d'une faute. Une défaillance survient dès que le système utilise un état erroné [3]. Dans [5], une classification des erreurs selon leurs types a été proposée. Premièrement, le service ne correspond pas en valeur à celui spécifié. Deuxièmement, le service n'est pas délivré dans l'intervalle de temps spécifié. Plusieurs catégories ont été définies dans [5], le service rendu est toujours en avance ou toujours en retard ou encore arbitrairement en avance ou en retard. Le fait que le système omette de rendre le service est considéré également comme un type d'erreur. Un arrêt (crash) est défini par le fait que le système omet définitivement de délivrer des services.

Défaillances : une défaillance dénote l'incapacité d'un élément du système à assurer le service spécifié par l'utilisateur. La défaillance est caractérisée par son domaine, sa perception par les utilisateurs et ses conséquences sur l'environnement [1].

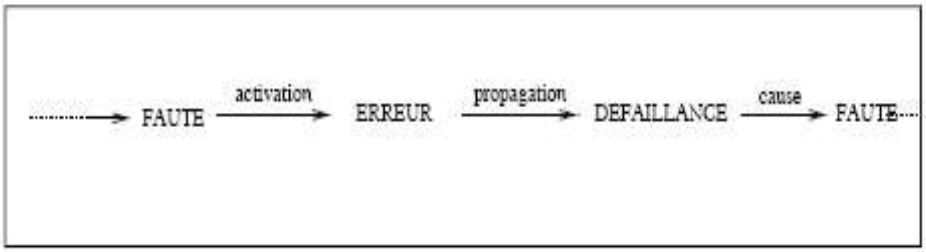


Figure 1. 2 : Chaîne des entraves.

[1] Nous utilisons le plus souvent le mot faute qui veut dire panne.

Une erreur est dite **latente** tant qu'elle n'a pas provoqué de défaillance. Le temps entre l'apparition de l'état d'erreur et la défaillance est le délai de latence. Ainsi, plus le délai de latence est long, plus la recherche des causes d'une défaillance est difficile.

La notion de causalité étant récursive, l'identification d'une faute impose toujours une part de subjectivité. Par exemple, si un ion lourd traverse le circuit d'un satellite, entraînant sa perte, doit-on incriminer l'ion lourd (faute physique, externe au système, opérationnelle), ou doit-on chercher du côté du blindage du circuit qui serait mal conçu (faute humaine, «interne » au système, de conception) ? Une défaillance peut aussi résulter d'une combinaison de fautes multiples, mais nous ne rentrerons pas dans ces subtilités.

2.2 Attributs de la sûreté de fonctionnement

Les attributs de la sûreté de fonctionnement d'un système mettent plus ou moins l'accent sur les propriétés que doivent vérifier la sûreté de fonctionnement du système. Ces attributs permettent d'évaluer la qualité du service fourni par un système. Dans [1], six attributs de la sûreté de fonctionnement sont définis :

Disponibilité : c'est la propriété requise par la plupart des systèmes sûrs de fonctionnement. Il s'agit de la fraction de temps durant laquelle le système est disponible pour des fins utiles (en excluant les temps de défaillance et de réparation);

Fiabilité : cet attribut évalue la continuité du service i.e. le taux en temps de fonctionnement pendant lequel le système ne subit aucune faute;

Sécurité-innocuité : similaire à la fiabilité mais par rapport aux conséquences catastrophiques causées par les fautes ;

Confidentialité : cet attribut évalue la capacité du système à fonctionner en dépit de fautes intentionnelles et d'intrusions illégales ;

Intégrité : l'intégrité d'un système définit son aptitude à assurer des altérations approuvées des données ;

Maintenabilité : cette propriété décrit la souplesse du système vis-à-vis des modifications apportées en vue de sa maintenance.

L'importance des attributs de la sûreté de fonctionnement présentés ci-dessus est principalement liée aux applications et à leurs besoins. Par exemple, pour les applications critiques comme le pilotage de fusées, une grande importance doit être donnée à tous les

attributs de la sûreté de fonctionnement. Les applications parallèles à longue durée d'exécution font prévaloir la maintenabilité et la fiabilité.

Donc, il n'y a pas de critère absolu, l'importance relative des critères dépend :

- ✓ De la nature de l'application
- ✓ Des exigences des utilisateurs
- ✓ Des conditions d'utilisation (environnement, etc.)

Exemples :

1 - Système embarqué : fiabilité, disponibilité.

2 - Système de communication (ex : commutateur téléphonique) : disponibilité.

3 - Service de fichiers, base de données : disponibilité, sécurité, (security)

4 -Système de transport (exemple : navigation, guidage, freinage) : sécurité (safety), disponibilité.

2.3 Moyens d'assurer la sûreté de fonctionnement

Les moyens utilisés pour assurer la sûreté de fonctionnement sont définis par les méthodes et les approches utilisées pour assurer cette propriété [1]. Les approches les plus connues sont:

A. *La prévention des fautes*

La prévision des fautes s'attache aux moyens permettant d'éviter l'occurrence de fautes dans le système. Ce sont généralement les approches de vérification des modèles conceptuels.

B. *L'élimination des fautes*

L'élimination des fautes se focalise sur les techniques permettant de réduire la présence de fautes ou leurs impacts. Cela est réalisé par des méthodes statiques de preuve de la validité du système (simulation, preuves analytiques, tests, ...) Ces deux moyens ne sont cependant pas suffisants du fait de l'usure inévitable des systèmes (pour leur partie matérielle), des environnements souvent agressifs dans lesquels ils évoluent, et, particulièrement pour les logiciels, de l'impossibilité de concevoir des systèmes totalement exempts de fautes.

C. *La prévision des fautes*

La prévision des fautes prédit l'occurrence des fautes (temps, nombre, impact) et leurs conséquences. Ceci est réalisé généralement par des méthodes d'injection de fautes afin de valider le système relativement à ces fautes.

D. *La tolérance aux fautes*

La tolérance aux fautes essaye de fonctionner en dépit des fautes. Le degré de tolérance aux fautes se mesure par la capacité du système à continuer à délivrer son service en présence de fautes.

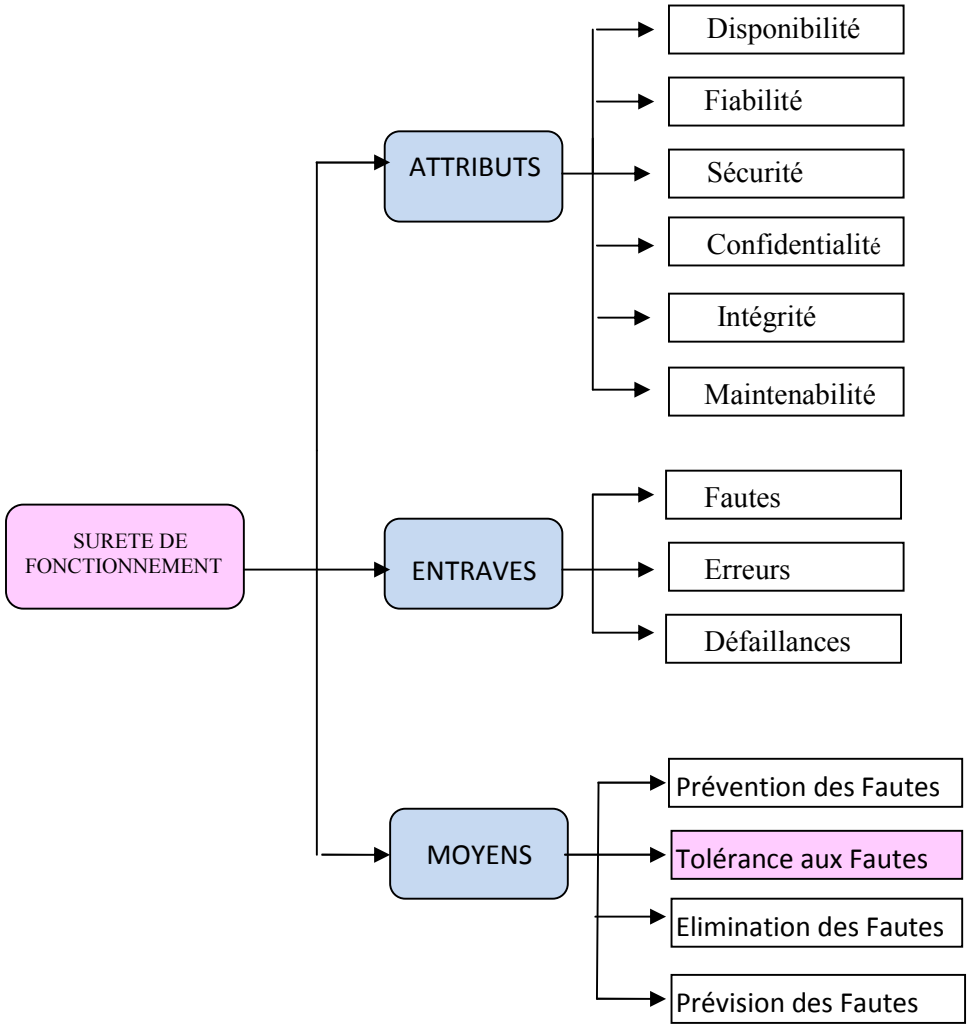


Figure 1. 3 : l'arbre de la sûreté de fonctionnement.[1]

3. Techniques permettant d'atteindre la tolérance aux fautes

Un système est usuellement composé d'autres systèmes, plus petits, qui lui délivrent leurs services. La défaillance d'un de ses composants devient alors une faute pour le système global (figure1.4). La tolérance aux fautes consiste à protéger l'utilisateur externe des conséquences des défaillances des composants internes du système.

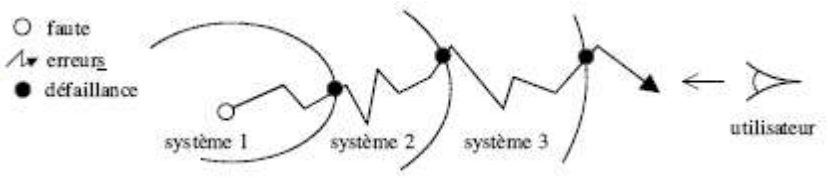


Figure 1. 4 : Récursivité des fautes, erreurs et défaillances [15]

En cassant cette chaîne de causalité, c'est-à-dire en empêchant une erreur de se propager jusqu'à l'utilisateur, on améliore la sûreté de fonctionnement d'un système tout en tolérant ses fautes : c'est le quatrième moyen de la sûreté de fonctionnement, la tolérance aux fautes, que nous détaillons plus dans la section qui suit.

La tolérance aux fautes est mise en œuvre par :

- ✓ Traitement de la faute qui vise à éviter qu'une faute survenue ne se reproduise.
- ✓ Traitement d'erreurs qui vise à éliminer une erreur avant qu'elle ne produise une défaillance.

3.1 Traitement de faute

Cette technique est basée sur:

- **Cas matériels** : Prévoir des composants multiples pour réduire la probabilité qu'une faute puisse conduire à une erreur [8].
- **Cas logiciels** : Réaliser des traitements multiples, comme par exemple la réalisation de la même opération par des algorithmes différents pour tenter d'éviter les fautes de conception.

3.2 Traitement d'erreur

Afin de traiter les erreurs présentes dans un système informatique, deux taches essentielles sont distinguées :

3.2.1 La détection d'erreur permet au système d'identifier un état erroné au cours de son fonctionnement. La détection s'accompagne éventuellement d'un diagnostic pour évaluer le degré de propagation de l'erreur, et l'importance des dommages causés.

3.2.2 Le recouvrement d'erreur consiste à ramener le système vers un fonctionnement sûr (mais éventuellement dégradé) en remplaçant l'état erroné et donc potentiellement dangereux, par un état sans erreur.

On distingue trois formes de recouvrement :

a- Recouvrement par reprise

Le système est ramené dans un état antérieur à partir duquel il reprend son fonctionnement (un exemple de la vie quotidienne consiste à rallumer son ordinateur de bureau après qu'il s'est bloqué, et à recommencer à travailler sur la dernière version sauvegardée de son travail). Cette technique est générale (on peut l'implémenter sur tout type de système) et nécessite la sauvegarde de l'état. Elle est adaptée aux fautes logicielles tout comme aux fautes matérielles transitoires. Cependant, dans notre travail, nous nous situons dans cette classe [6].

b- Recouvrement par poursuite

Le système est amené dans un nouvel état, connu à priori, à partir duquel il peut continuer son fonctionnement (souvent de manière dégradée). Ce mode de recouvrement est souvent très dépendant de l'application. Il n'est pas possible de généraliser son implémentation dans tous les systèmes. Le programmeur doit développer ses applications avec prise en charge de ce mécanisme pour pouvoir traiter les exceptions. Elle est plus adaptée aux fautes logicielles.

c- Recouvrement par compensation

La redondance contenue dans l'état erroné suffit à éliminer l'erreur et à poursuivre l'exécution.

Si l'on estime que la faute est encore présente après le recouvrement (par exemple dans le cas d'une faute persistante), et que la probabilité d'une nouvelle activation n'est pas suffisamment faible, un traitement de faute, constitué d'un diagnostic suivi d'une passivation s'impose en complément. Nous ne les aborderons pas plus en détail dans ce mémoire.

La redondance est souvent utilisée pour réaliser détection d'erreur et recouvrement.

3.3 Redondance et diversité

En dépit de cette grande hétérogénéité de situations, deux grandes idées traversent la tolérance aux fautes: celles de redondance et de diversité. La redondance consiste à construire dans un système une sorte de « **surcapacité** » de service ou d'information, qui peut être partielle (un code correcteur d'erreurs, la vérification en ligne d'une condition de validité), ou totale (embarquer deux calculateurs là où un seul suffit en fonctionnement normal). La diversité consiste à s'assurer que la « surcapacité» ainsi ajoutée défaille indépendamment du service non redondant, ce que Laprie dénomme comme «l'indépendance des redondances par rapport aux processus de création et d'activation des fautes » [1].

Le concept de diversité est essentiel, et est étroitement lié aux fautes que l'on cherche à tolérer, ceci en vertu d'une constatation très simple : aucun élément redondant ne peut protéger des fautes qui l'amènent à défaillir de la même manière et au même moment que le système qu'il double. C'est ainsi que le vol 501 d'**Ariane 5** a échoué en 1996 suite à une défaillance de son système de référence inertielle, en dépit de ses deux calculateurs embarqués. La cause de la défaillance était une faute du logiciel et impactait les deux calculateurs de la même manière, ceux-ci étant rigoureusement identiques, et faisant tourner le même programme. S'il s'était agi d'une avarie matérielle, les conséquences auraient été toutes autres, le second calculateur prenant la main, et masquant ainsi la faute.

On voit ici qu'à chaque type de redondance (matérielle dans le cas d'Ariane 5, de conception dans l'avionique civile, d'information avec les codes correcteurs d'erreur), correspond une classe de fautes qui peuvent être tolérées.

3.4 Modèles de fautes

Nous l'avons vu avec l'exemple d'Ariane 5, il est essentiel de bien cerner la nature des fautes que l'on cherche à tolérer pour construire des mécanismes de tolérance qui contiennent la diversité appropriée pour être efficaces. De nombreuses classifications des fautes existent :

3.4.1 Classification des fautes selon le degré de gravité des défaillances

- ✓ **Fautes franches (crash fault)** : soit le système fonctionne normalement (les résultats sont corrects), 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 ;
- ✓ **Fautes par omission** : des messages sont perdus en entrée et/ou en sortie. Ce type de fautes est utilisé pour les réseaux ;
- ✓ **Fautes de temporisation** : les déviations par rapport aux spécifications concernent uniquement le temps (typiquement le temps de réaction à un événement) ;
- ✓ **Fautes par valeurs** : les résultats produits par un composant défaillant sont incorrects;
- ✓ **Fautes byzantines** : le système peut se comporter de manière totalement arbitraire (peut faire n'importe quoi, y compris avoir un comportement malveillant).

3.4.2 Classification des fautes selon le degré de permanence

- ✓ **Faute transitoire** : elle se produit de manière isolée ;
- ✓ **Faute intermittente** : elle se produit aléatoirement plusieurs fois ;
- ✓ **Faute permanente** : elle persiste dès qu'elle apparaît jusqu'à réparation.

3.4.3 Classification des fautes selon leur nature :

On utilise également un troisième classement selon la nature de la faute :

- ✓ F**autes accidentelles** : elles se produisent de manière accidentelle.
- ✓ F**autes intentionnelles** : elles sont créées avec une intention qui peut être malicieuse.

On peut aussi classer les fautes En fonction de la phase du cycle de vie du système au cours duquel elles apparaissent (fautes de conception dans le cas du vol 501 d'Ariane 5, faute opérationnelle),ou encore en fonction de leur origine par rapport aux frontières du système (internes, externes), voir tableau 1-1 suivant :

Critères	Classes

Gravité	Fautes franches
	Fautes par omission
	Fautes par valeur
	Fautes de temporisation
	Fautes byzantine
Origine	Fautes matérielles
	Fautes logicielles
Nature	Fautes accidentelles
	Fautes malicieuses
Phase d'occurrence	Fautes de développement
	Fautes opérationnelles
Etendue	Fautes internes
	Fautes externes
Persistance	Fautes permanentes
	Fautes transitoires

Tableau 1. 1 : Classification des fautes.

Chacune de ces classes de fautes appelle un traitement spécifique de tolérance aux fautes. Ainsi les fautes de conception ne peuvent être tolérées par une simple réplication matérielle à l'identique, c'est pourquoi en avionique civile des programmes répondant aux mêmes spécifications sont développés en plusieurs versions par des équipes différentes, ce qu'on appelle en anglais le N-version programming [7].

3.5 Exemples de défaillances de systèmes

Nous présentons dans le tableau 1.2 quelques exemples de défaillances dans des systèmes dans le monde :

	Fautes			Défail.				
	Physiques	Développement	Interaction	Localisées	Distribuées	Disponibilité/Fiabilité	Sécurité-innocuité	Confidentialité
Juin 1980 : Fausses alertes à des attaques massives de fusées soviétiques au NORAD	✔			✔		✔		
Juin 1985 - Janvier 1987 : Doses excessives de radiothérapie (Therac-25)		✔		✔			✔	
Août 1986 - 1987 : Le "Wily hacker" pénètre des dizaines de centres informatiques sensibles		✔	✔	✔				✔
15 Janvier 1990 : Indisponibilité totale du téléphone interurbain aux Etats-Unis, pendant 9h		✔			✔	✔		
Février 1991 : Scud raté par un Patriot à Dhahran		✔	✔	✔		✔	✔	
Novembre 1992 : Ecroulement des communications du service d'ambulances de Londres		✔	✔		✔	✔	✔	
26 et 27 Juin 1993 : Refus des cartes de crédit dans les distributeurs de monnaie en France	✔	✔			✔	✔		
4 Juin 1996 : Défaillance du vol 501 d'Ariane 5		✔		✔		✔		
17 Juillet 1997: Mélange des adresses du domaine internet .com			✔		✔	✔		
13 Avril 1998 : Ecroulement réseau de données d'AT&T		✔	✔		✔	✔		
Février 2000 : Engorgement de grands portails Web		✔	✔		✔	✔		
Mai 2000 : Virus "I love you"		✔	✔		✔	✔		
Juillet 2001 : Ver "Code Red"		✔	✔		✔	✔		
Août 2003 : Propagation de panne électrique dans le Nord-Est des USA et le Canada		✔	✔		✔	✔		

Tableau 1. 2 : Quelques défaillances dans le monde.

Dans le cadre de cette thèse, on s'intéresse prioritairement aux traitements associés aux pannes franches (crash fault) des processus. Plus précisément, on ne traite que les fautes qui peuvent être modélisées et considérées comme des fautes franches.

4. L'exigence de la tolérance aux fautes dans les systèmes répartis

La tolérance aux fautes dans les systèmes répartis est un domaine très vaste qui a fait l'objet d'une littérature abondante. Comme nous l'avons déjà évoqué, la tolérance aux fautes est la capacité du système à suivre le comportement défini, et cela même en présence de fautes [9].

L'exigence de tolérance aux fautes est incontournable avec les systèmes répartis. D'une part, la tolérance aux fautes est un **besoin** induit par la multiplicité des ressources et d'autre part de nombreux travaux sur les systèmes répartis ont pour but de garantir que la sûreté de fonctionnement de ces systèmes n'est pas dégradée par la répartition. Par ailleurs, la tolérance aux fautes peut être en elle-même un facteur motivant la répartition. En effet, la tolérance aux fautes ne peut être assurée sans redondance et la répartition des traitements et des données sur des processeurs différents permet de structurer et gérer cette redondance.

La tolérance aux fautes dans de tels systèmes est mise en œuvre principalement par **logiciel**, ce qui permet d'espérer une certaine pérennité des moyens vis-à-vis des évolutions du matériel.

Afin de présenter les techniques de tolérance aux fautes dans les systèmes répartis, nous commençons cette section par une définition simple d'un système réparti.

4.1 Systèmes répartis

De manière générale [10], un système réparti peut être défini comme un ensemble de nœuds de calcul reliés par un réseau de communication et communiquant entre eux par messages.

Dans ce contexte, la tolérance aux pannes est un besoin imposé par la répartition et c'est pourquoi plusieurs travaux de recherches sur les systèmes répartis ont été réalisés afin de la prendre en compte.

4.2 Techniques réparties de tolérance aux fautes

La tolérance aux fautes dans les systèmes répartis et parallèles est toujours réalisée par l'emploi d'un mécanisme de redondance [4]. Cette dernière peut être **spatiale** (duplication de composants), temporelle (traitements multiples) ou bien **informationnelle** (redondance de données, codes, signatures). Dans un tel système à base de processus communicants, les techniques de tolérance aux fautes peuvent être séparées en deux classes : les techniques basées sur la duplication et les techniques basées sur une mémoire stable [11].

4.2.1 Tolérance aux fautes par duplication

La tolérance aux pannes par duplication consiste en la création de copies multiples des composants sur des processeurs différents. Cette approche par duplication rend possible le **traitement des fautes** en les masquant. Trois stratégies principales pour réaliser la duplication sont proposées dans la littérature : les duplications actives, passives et semi-actives. Ces différentes stratégies visent à garantir une cohérence forte entre les copies d'un composant dupliqué.

a. **Duplication active** (N-modules replication)

Elle est définie par la symétrie du comportement des copies d'un composant dupliqué où chaque copie joue un rôle identique à celui des autres.

b. **Duplication passive** (primary-backup replication)

Contrairement à la duplication active, la duplication passive (passive replication ou primary-backups réplication) [12] est asymétrique. Elle distingue deux comportements pour les copies d'un composant dupliqué : la copie primaire (primary copy) et les copies secondaires (backups). La copie primaire est la seule à effectuer tous les traitements. Les copies secondaires, oisives, surveillent la copie primaire. En cas de défaillance de la copie primaire, une des copies secondaires devient la nouvelle copie primaire.

c. **Duplication semi-active** (leader-follower replication)

La duplication semi-active (semi-active replication ou leaderfollowers replication) [12] se situe à mi-chemin entre la duplication active et la duplication passive. Comme cette dernière, la duplication semi-active est une stratégie asymétrique. Contrairement à la duplication passive, les copies secondaires ne sont pas oisives.

La tolérance aux pannes par duplication est alors réalisée par **masquage d'erreur**. La défaillance d'une copie est masquée par le comportement des copies non défaillantes. Cette technique permet de tolérer un nombre **limité** de fautes (au maximum #duplicata−1 dans le cas de pannes franches) et n'est pas adaptée aux calculs parallèles massifs en terme de performances et de ressources de calculs nécessaires à sa mise en œuvre.

4.2.2 Tolérance aux fautes par mémoire stable

L'existence d'une mémoire stable permet la réalisation de la tolérance aux fautes par une détection de défaillance suivie d'un recouvrement d'erreur. La mémoire stable n'est qu'une abstraction [13]. Elle peut être définie comme un support persistant de stockage, dont le rôle principal est d'assurer une accessibilité et une protection aux données contre les fautes pouvant affecter le système. Ainsi, suite à une faute, un état correct ayant été stocké antérieurement à cette faute sur la mémoire stable reste accessible ; cela permet au système un retour à un état antérieur.

Une fois qu'une panne est détectée, un mécanisme de recouvrement doit être mis en place pour traiter la faute identifiée. Le recouvrement peut être par reprise ou bien par poursuite.

Comme nous l'avons déjà remarqué, pour pouvoir réaliser une reprise après une faute, l'état du système ou de l'application doit être sauvegardé périodiquement ou lors de l'occurrence d'événements spécifiques. L'état sauvegardé doit constituer un point de reprise à partir duquel le système peut fonctionner correctement.

Le point critique pour réaliser la tolérance aux pannes par mémoire stable dans les systèmes répartis est la constitution d'un état de reprise cohérent qui réponde aux objectifs attendus de la tolérance aux pannes : QoS, surcoût introduit, type de pannes à traiter, en cas de défaillance reprise globale du système ou bien uniquement reprise des processus défaillants, nombre de composants fiables dans le système durant toute son exécution,etc... . Plus de détails seront présentés dans le chapitre qui suit.

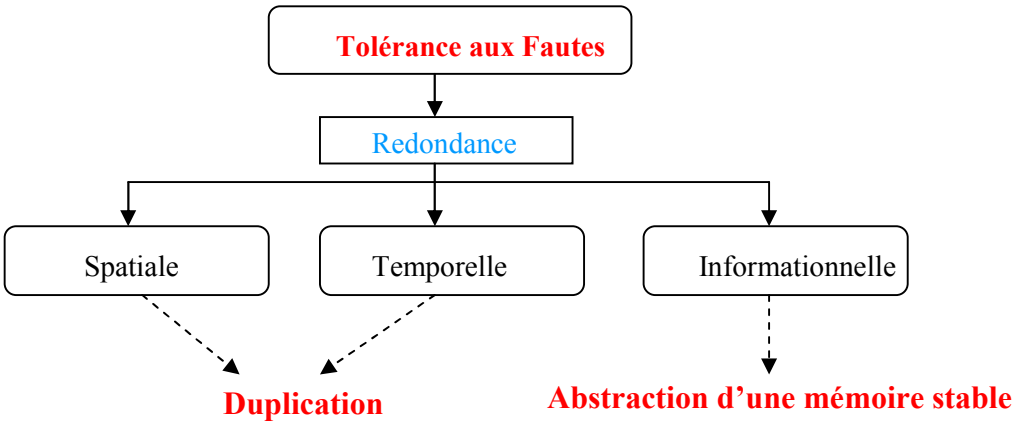


Figure 1. 5 : Les techniques de tolerance aux fautes dans les systems répartis.

5. Bilan et Conclusion

Dans ce chapitre, nous avons présenté la notion de sûreté de fonctionnement d'un système informatique. Nous avons vu que la tolérance aux fautes n'est qu'un moyen pour assurer la sûreté de fonctionnement d'un système et qu'on utilise toujours la redondance pour l'obtenir. Deux méthodes principales permettent de réaliser la tolérance aux fautes dans les systèmes répartis : la duplication et l'abstraction d'une mémoire stable.

Quelles que soient les précautions prises, **l'occurrence de fautes est inévitable** (erreur humaine, malveillance, vieillissement du matériel, catastrophe naturelle, etc.). Cela ne veut pas dire qu'il ne faut pas essayer de prévenir ou d'éliminer les fautes, mais les mesures prises peuvent seulement réduire la probabilité de leur occurrence. Il faut donc concevoir les systèmes de manière à ce qu'ils continuent à rendre le service attendu (éventuellement un service dégradé) même en présence de fautes

Pour conclure ce chapitre, il faut noter qu'il n'existe pas de méthode de tolérance aux fautes qui soit valable dans l'absolu [4]. Seules existent des méthodes adaptées à des hypothèses particulières d'occurrence de fautes. Ces hypothèses doivent donc être explicitement formulées, après analyse soigneuse

Dans notre travail, nous avons adopté le principe d'une mémoire stable pour réaliser la tolérance aux pannes franches pour des applications parallèles s'exécutant sur un réseau mobile ad hoc. Avec cette orientation, nous présentons dans le chapitre suivant, les différents problèmes liés à la tolérance aux fautes par reprise et les diverses solutions proposées.

Chapitre 2

1- Introduction

Comme nous l'avons vu dans le chapitre précédent, la tolérance aux fautes par reprise (recouvrement arrière ou encore rollback) est basée sur la redondance des informations, autrement dit, sur la sauvegarde de données à partir desquelles une reprise du calcul est possible en cas de panne. Ceci nécessite d'une part, l'existence d'une mémoire stable de sorte que les données soient toujours accessibles et, d'autre part, que l'état du système ou du calcul soit cohérent après une reprise.

La condition générale pour qu'une reprise après panne soit correcte peut être définie comme suit : un système (application) peut réaliser une reprise correcte si son état interne est cohérent avec un comportement du système observé avant la panne [16]. Par conséquent, le protocole de reprise doit essentiellement maintenir les informations concernant les interactions non seulement entre les processus participants, mais également les interactions, s'il y en a, avec le monde extérieur.

Ce chapitre présente quelques concepts de base nécessaires afin de poser la problématique de la reprise sur erreur, ainsi que les détails pour la mise en œuvre d'une stratégie de recouvrement arrière.

2- Concepts Généraux

Avant d'entamer la problématique de la reprise sur erreur, nous allons présenter quelques définitions et notions de base nécessaires par la suite dans les sections suivantes.

2.1 Application parallèle

Afin de définir une application parallèle, nous introduisons les définitions suivantes :

Définition 1 : **Une tâche** est une séquence d'instructions exécutées localement sur une ressource unique, éventuellement de manière non préemptive.

Définition 2 : Une application parallèle est un ensemble de tâches éventuellement soumises à des conditions de synchronisation pouvant être potentiellement exécutées en même temps et sur des sites géographiques différents. L'objectif principal est de limiter le temps d'exécution de l'application.

Une application parallèle est donc souvent composée de n processus distincts communiquant via un réseau d'interconnexion, chacun des processus exécutant des tâches et les communications qui leurs sont associées. Par conséquent, l'état de l'application parallèle à un instant t donné, est l'ensemble des états locaux de tous les processus

participants et de tous les canaux de communications ; autrement dit, de tous les messages en transit à l'instant t. Cependant, un processus ne peut stocker que son état local et ses messages entrants et sortants.

Aussi, les messages en transit à un instant donné, ne peuvent être contrôlés par aucun des processus de l'application parallèle.

2.2 Modèle de communication par message

Une application parallèle ou distribuée est composée de plusieurs processus coopérants s'exécutant de façon parallèle et communiquant par échange de messages, chacun peut s'exécuter sur un ordinateur différent (site). Différents processus faisant partie d'une même application s'échangent des informations ; Plusieurs techniques existent pour ce faire : la mémoire ou les variables partagées, et la communication par messages [15].

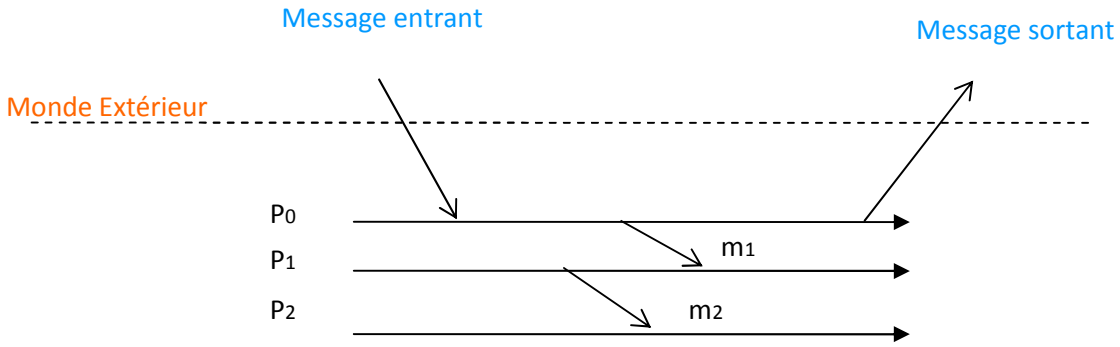


Figure 2. 1 : Applications Distribuées Communiquant Par Message.

La communication par messages passe généralement par au moins deux primitives : Envoi (message, processus) et Réception (message, processus). Ce modèle de programmation est utilisé par de très nombreuses applications de calcul scientifique dans le cadre de leur exécution sur grappe. C'est pourquoi, il est particulièrement intéressant de permettre à ces applications d'intégrer de manière transparente un mécanisme de reprise pour se prémunir contre les cas de défaillance.

2.3 Problèmes liés à la répartition

De manière générale, la définition 2 d'une application parallèle nous place dans le contexte d'un système réparti. Par conséquent, le raisonnement sur le système ou l'application (état, ordre des événements) pose un certain nombre de problèmes liés à la répartition.

- ✓ **L'absence d'une mémoire commune**, alors que la mémoire est le support habituel de l'état d'une application ou d'un système.
- ✓ **L'absence d'une horloge commune**, or l'existence de cette horloge permet la définition d'un ordre sur les événements du système.
- ✓ **L'asynchronisme des communications** entre les processus induisant que la borne supérieure du temps de transmission d'un message est a priori inconnue.
- ✓ **L'hétérogénéité** des sites d'exécution.

En 1978, Lamport a proposé de définir pour les systèmes répartis une notion de temps logique permettant de comparer des événements selon leur ordre d'exécution. Sur un site, les événements locaux peuvent être ordonnés en se basant sur l'ordre de leur exécution. L'émission d'un message sur le site émetteur précède toujours sa réception sur le site récepteur. Ceci correspond à la notion de précédence causale que l'on verra par la suite.

2.4 Interaction avec le monde extérieur

Les applications réparties sont souvent amenées à interagir avec le monde extérieur pour recevoir des données ou produire des résultats. Le monde extérieur ne peut pas s'appuyer sur des mécanismes de retour arrière en cas de fautes. Par exemple, une imprimante ne peut pas revenir en arrière ou un distributeur de billet ne peut pas reprendre l'argent qu'il a donné à un client. Il est donc nécessaire de fournir au monde extérieur un comportement cohérent malgré les fautes.

Ainsi, avant chaque envoi vers le monde extérieur, le système doit garantir que l'état générant cet envoi pourra être reproduit en cas de faute. Chaque interaction avec le monde extérieur doit être validée. Classiquement un état global cohérent est sauvegardé atomiquement après chaque interaction. Cette validation avec le monde extérieur est appelée output commit.

2.5 Etat global d'un système réparti

Un état global d'un système réparti est un ensemble d'états locaux dans lequel on trouve un enregistrement d'état local par processus.

2.6 Message orphelin

Un message orphelin par rapport à un état global est un message qui a été envoyé après un enregistrement d'état local appartenant à un état global et reçu avant un enregistrement d'état local appartenant à un état global.

2.7 Message en transit

Un message est dit en transit (in-tansit message) par rapport à un état global, est un message qui a été envoyé avant un enregistrement d'état local appartenant à un état global, et reçu après un enregistrement d'état local appartenant à un état global.

2.8 L'effet domino

Sous quelques scenarios, la propagation du recouvrement arrière peut aller jusqu'à l'état initial du calcul, favorisant ainsi, la perte de tout le travail effectué avant l'apparition de l'erreur. Cette situation est connue sous le nom de l'effet domino [17]. La figure 4.2 montre un exemple de l'effet domino (the domino effect).

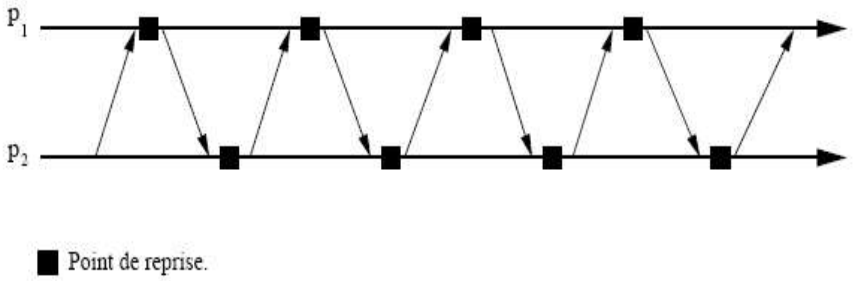


Figure 2. 2 : Effet Domino

2.9 La propagation de la reprise

Dans un système réparti les processus peuvent agir indépendamment les uns des autres ou coopérer pour l'accomplissement d'un service.

a. Cas des processus indépendants

Dans le cas ou le processus n'a pas eu de communication avec aucun des processus d'une application distribuée, la reprise est purement locale au processus défaillant [21].

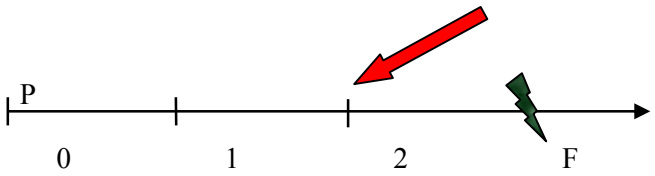


Figure 2. 3 : Reprise de processus indépendants

La figure 2.3, illustre l'évolution d'exécution de P dans le temps, P prend des points de reprises aux instants 0,1 et 2. A l'instant F, le processus P détecte une erreur, alors P exécute une reprise à partir de l'état 2.

b. Cas de processus communicants

La communication entre processus implique l'envoi ou la réception de messages entre eux, si P envoie un message à Q, Q devient **dépendant** de P. contrairement au premier cas, pour doter un système de processus communicants, d'un mécanisme de reprise, il faut prendre en compte le problème très complexe de la propagation d'erreur relative aux dépendances directes ou transitives [21], [22], [23]. Cette propagation peut conduire à la contamination de tous les processus communicants au processus défaillant et comme il peut y avoir une propagation de l'erreur, il serait normal d'avoir une propagation de reprise.

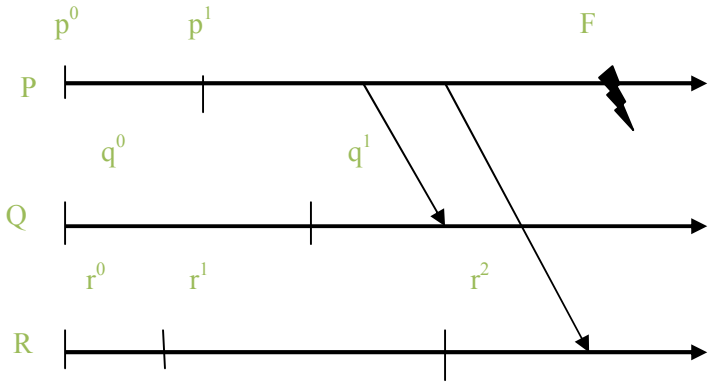


Figure 2. 4 : Propagation de reprise

La figure 2.4, montre l'évolution de trois processus communicants P, Q et R. à l'instant F, le processus P détecte une erreur, alors il lance une opération de reprise à partir du point P^1. Comme Q a reçu des données de P, ces données sont potentiellement erronées, alors Q doit aussi exécuter une reprise à partir de q^1. De même pour R doit restaurer l'état r^2 et effectuer une reprise à ce point.

La liste des **dépendants** d'un processus P, est l'ensemble des processus auxquels il a envoyé des messages. Pour le processus P, ce sont : Q et R.

La liste des **propagateurs** d'un processus P, est l'ensemble des processus de la part desquels il a reçus des messages.

A partir de ce point, nous remarquons que la communication entre processus nous force à aborder un autre axe qui est celui de ne pas considérer l'état local d'un processus pour la reprise, mais tous les états des processus communicants. On parle alors d'une ligne de recouvrement.

2.10 Ligne de recouvrement

La ligne de recouvrement (recovery line) est l'état global cohérent le plus récent. Cette ligne de recouvrement doit être assez récente, de façon à minimiser le retour arrière en cas de reprise, c'est-à-dire le nombre de points de reprise pris entre le dernier pris et celui utilisé pour la reprise. Nous verrons cependant que certains protocoles ne favorisent pas la création d'une ligne de recouvrement durant l'exécution (**à priori**), mais recherchent cette ligne dans l'ensemble des états globaux après qu'une erreur soit survenue dans le système (**à posteriori**).

3. Problématique de la reprise

Deux problèmes interviennent pour effectuer un recouvrement d'erreur par reprise : le déterminisme d'exécution et la cohérence d'état global de l'application parallèle (système réparti).

3.1 Exécution déterministe

Le déterminisme de l'exécution d'un programme P est le fait de pouvoir répéter l'exécution de P plusieurs fois et de la même façon. Cette propriété d'exécution permet de déterminer si la reprise peut être faite par une technique de journalisation de l'exécution ou non. En effet, le stockage de l'histoire de l'exécution de P dans une mémoire stable est inutile si la réexécution de P n'est pas équivalente à l'exécution de P.

Afin d'obtenir une exécution déterministe, un processus peut être modélisé par une séquence d'intervalles d'états [16] chacun débutant par un événement non déterministe pouvant être identifié.

L'exécution durant chaque intervalle d'état est déterministe. Ce concept d'intervalle d'état est appelé "hypothèse de déterminisme par morceaux" (PWD : Piecewise deterministic assumption).

L'hypothèse PWD permet de capturer suffisamment d'informations concernant les événements non déterministes qui initialisent les intervalles d'état.

3.2 État global cohérent d'une application parallèle

Comme nous l'avons déjà évoqué dans la section précédente, un **état global** d'une application parallèle est une collection des états locaux de tous les processus participant au calcul (un par processus) et les états des canaux de communications entre les processus.

L'exemple de Chandy et Lamport en parlant de:

Un groupe de photographe observe une scène dynamique, par exemple un vol d'oiseaux migrateurs. La scène est si vaste qu'elle ne peut être prise par un seul photographe. Les différents photographes doivent donc se charger de prendre chacun une partie de la scène tout en sachant :

- ✓ que les photographies ne peuvent être prises au même moment
- ✓ que ces opérations ne doivent pas perturber le phénomène à observer (par exemple, il n'est pas raisonnable de demander aux oiseaux d'arrêter leur vol pendant les prises de vue).

Il faut aussi que la scène reconstituée ait ''un sens'' : reste à définir ce qu'est ''avoir un sens'' et de déterminer la manière de faire ces prises de vues.

***Relation de précédence causale* :** Pour définir l'état d'un système distribué, il faut d'abord pouvoir ordonner les événements. Tout schéma global de datation doit pouvoir respecter l'ordre local sur chaque site, et doit être tel que la date d'émission d'un message soit antérieure à sa date de réception (principe de causalité). Une relation d'ordre entre les événements, vérifiant ces conditions, a été définie comme suit :

Soit a et b deux événements. On dit que a précède directement b si et seulement si l'une des deux conditions suivantes est vraie :

1- a et b se sont produits sur le même site, et a est antérieur à b sur ce site.
2- a est l'envoi d'un message m depuis un site, et b la réception de ce message m sur un autre site.

La relation précède (notée $\rightarrow$) est la fermeture transitive de la relation précède directement. Cette relation dite de précédence causale est un ordre partiel ; elle traduit une dépendance potentielle : pour que a puisse être la cause de b, il est nécessaire que a $\rightarrow$ b. Deux événements sont dits concurrents (ou causalement indépendants) si aucun des deux ne précède causalement l'autre, et ne peut donc influer sur lui.

On écrit : a // b $\Leftrightarrow$ (non (a $\rightarrow$ b) et non (b $\rightarrow$ a).

Un état global correspond à l'ensemble des états individuels des processus participants et des états des canaux de communication. Un état global **cohérent** est un état qui aurait pu se produire en l'absence de fautes, durant une exécution correcte d'une application distribuée. Ce n'est pas nécessairement un état qui s'est produit avant la reprise.

Dans un état global cohérent, si un processus possède un état local indiquant qu'un message m a été reçu, l'état du processus émetteur du message m, doit indiquer que le message m a été envoyé [1], autrement dit : Un état global est cohérent si est seulement s'il ne met pas en défaut la dépendance causale. La figure 2.5, montre un exemple d'état global cohérent C1, C2, C3. Les messages m2 et m3 ont été envoyés mais pas reçus, ce qui est normal durant une exécution correcte (sans faute), car à un instant donné, o**n** peut avoir dans le système des messages envoyés, mais pas encore reçus.

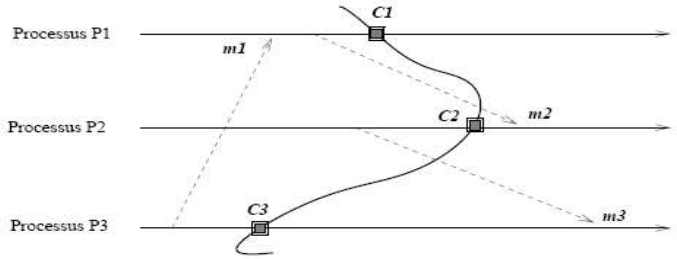


Figure 2. 5 : Exemple d'état global cohérent.

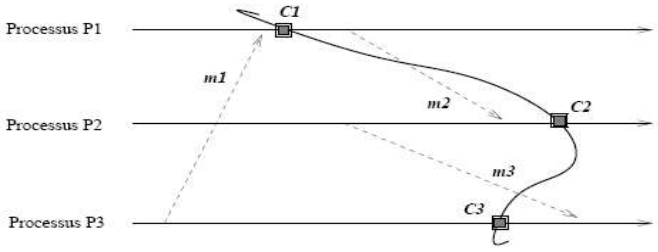


Figure 2. 6 : Exemple d'état global incohérent

Par contre, l'état global C1, C2, C3 illustré dans la figure 2.6, n'est pas cohérent. En effet, le message m2 n'a pas été envoyé par le processus P1 alors que m2 est reçu par le processus P2. Cette situation, ne peut jamais se produire dans une exécution correcte.

Nous avons vu dans la figure 2.5 que l'état global C1, C2, C3 contient certains messages (m2 et m3) envoyés mais pas reçus. Ces messages sont appelés messages en transit et ils font partie de l'état global du système (application).

Les messages en **transit** n'introduisent **aucune incohérence** dans l'état. Cependant, la garantie de la délivrance des messages en transit doit être assurée, soit par l'hypothèse d'un réseau de communication fiable, soit par le protocole de sauvegarde/reprise lui-même.

Par contre l'existence de message **orphelin**, i-e un message reçu sans être envoyé (message m2 dans le figure 2.6), cause l'**incohérence** de l'état global qui le contient. Cette notion d'état global cohérent a été formalisée par Chandy et Lamport dans [14].

4. Mise en œuvre d'une stratégie de recouvrement arrière

Cette section présente les différents mécanismes de base qui sont nécessaires à l'implémentation d'un mécanisme de sauvegarde/reprise pour les applications parallèles.

Ces mécanismes sont la détection des défaillances, la réalisation d'une mémoire stable et la sauvegarde de l'état d'un processus.

4.1 Technique de détection des défaillances

La première étape d'une stratégie de recouvrement arrière est la détection de la défaillance d'un processus. En l'absence de communication en provenance d'un processus pendant une durée t, celui-ci est considéré potentiellement défaillant. Un temporisateur est amorcé et un message spécial est envoyé à ce processus. En l'absence de réponse avant la fin du temporisateur, le processus est jugé défaillant et le recouvrement arrière peut commencer.

Cette technique simple n'est pas optimale, puisqu'elle a pour inconvénient de distinguer difficilement un processus lent d'un processus mort. D'autres techniques, comme le protocole HeartBeat, existent pour pallier ce type de problèmes et sont présentées dans [18].

Dans le cadre d'applications communiquant par messages, cette détection peut en pratique être réalisée de manière distribuée ou centralisée.

4.2 Réalisation d'une mémoire stable pour le stockage des points de reprise

La tolérance aux fautes par recouvrement arrière repose sur l'existence d'une mémoire stable. Les informations qui sont nécessaires en cas de panne et de reprise d'un ou plusieurs éléments du système doivent être elles-mêmes stockées de façon tolérante aux fautes et ne doivent pas être altérées. Lorsqu'une donnée est stockée sur la mémoire stable, on dit que cette donnée est stable.

Cela suppose une redondance des données à conserver. Par ailleurs, lors de la création d'un nouveau point de reprise, le précédent ne doit être invalidé qu'après finalisation du nouveau. Ainsi, s'il y a échec au cours de la sauvegarde d'un point de reprise, le précédent est toujours disponible et considéré comme point de reprise de référence. On dit qu'il y a atomicité des mises à jour des points de reprise.

Donc, un support peut être stable, s'il remplie les conditions suivantes :

- **L'accessibilité :** c'est-à-dire qu'il existe un chemin permettant d'accéder aux données en dépit des défaillances.
- **La non altération des données :** les défaillances ne doivent pas altérer les données se trouvant sur ce support.
- **L'atomicité des mises à jour des points de reprise :** les mises à jour des points de reprise sur ce support doivent se faire de manière atomique, c'est-à-dire lors d'un nouveau point de reprise, le précédent ne doit être invalidé qu'après finalisation du nouveau.

Cependant, si ces trois conditions sont vérifiées pour tous les sites impliqués dans un calcul, alors chaque site sauvegarde ses propres points de reprise dans son support de stockage. Sinon et en fonction du nombre de fautes à tolérer, des performances et des moyens disponibles, plusieurs solutions sont possibles pour réaliser une mémoire stable [19] :

- ✓ La solution la plus simple est l'utilisation d'un serveur considéré comme fiable, mais cette technique transforme parfois ce serveur en un goulot d'étranglement du réseau, il est alors nécessaire de dupliquer le serveur mais le système peut perdre une bonne partie de sa simplicité.

- ✓ Une deuxième solution consiste à sauvegarder les points de reprise dans la mémoire vive d'un site distant. Cette solution suppose qu'il est peu probable que le site de calcul et celui conservant ses points de reprise subissent simultanément une défaillance. Ceci ne permet de résister qu'à une seule faute, l'autre inconvénient de cette méthode est la forte augmentation de l'utilisation mémoire. Pour éviter le surplus d'utilisation mémoire, une variante consiste à sauvegarder les points de reprise sur un disque local d'un site distant. Le défaut est alors le coût en latence dû aux accès en écriture sur disque.
- ✓ Une autre méthode consiste à utiliser le double Checkpointing. L'idée est de sauvegarder les points de reprise sur deux sites différents, offrant ainsi une meilleure résistance aux fautes. Cette méthode propose le choix entre deux techniques :

    1. **Le double in-memory Checkpointing** qui consiste à sauvegarder les points de reprise sur les *mémoires vives* locales de deux sites différents.
    2. **Le double in-disk Checkpointing** qui consiste à sauvegarder les points de reprise sur les *disques locaux* de deux sites différents.

Des études et des expériences montrent l'avantage considérable en termes de performance qu'offre le mécanisme de double in-memory Checkpointing par rapport aux autres solutions. Il ne faut cependant pas oublier le surcoût en mémoire qui s'y attache et qui rend cette solution inutilisable lorsque la taille des données manipulées devient très importante.

4.3 Sauvegarde de l'état du processus

Cette section s'intéresse à la sauvegarde de l'état d'un processus, et a pour objectif de pouvoir relancer ce dernier dans l'état dans lequel il était lors de la sauvegarde. Nous nous intéressons tout d'abord aux besoins des utilisateurs. Ensuite, nous évoquerons les méthodes d'implémentation des points de reprise, ainsi que le contenu à sauvegarder.

4.3.1 Transparence de la reprise

Plusieurs méthodes de reprises sont envisageables : dans certains types d'applications, on peut décider dés leur conception d'intégrer au programme même de l'application des mécanismes de checkpointing et de tolérance aux fautes. Cette approche est souvent efficace en terme de performance, puisqu'elle laisse au programmeur le choix du moment où l'on provoque un point de reprise, mais elle est très coûteuse en temps de développement puisque le programmeur doit tenir compte d'une contrainte supplémentaire.

Cette solution est peu adaptée dans un réseau de communication : on ne souhaite pas investir le temps dans ce développement logiciel coûteux. On préférera une solution offrant une tolérance aux fautes par reprise **transparente** : c'est à dire que le programme n'a pas connaissance de la réalisation d'un checkpoint.

Notons que la notion de transparence possède un sens particulier en informatique, puisque son dual est l'opacité. On dit qu'une fonctionnalité est « transparente » lorsque le programmeur n'a pas besoin de la prendre explicitement en compte pour en bénéficier. Cette transparence provient du masquage de la complexité sous-jacente, par un mécanisme dont l'implémentation est opaque. Dans la suite de cette étude, nous ne nous intéressons donc qu'à cette possibilité.

4.3.2 Niveau de mise en œuvre des points de reprise

Les points de reprise peuvent être implémentés de trois manières : par l'application elle-même, par une bibliothèque liée à l'application, ou par le système d'exploitation.

- ✓ La sauvegarde au niveau applicatif consiste en l'insertion au sein de l'application, de code permettant la réalisation des points de reprise. Cette technique promet les meilleures performances. En effet, le programmeur sait parfaitement quelles données sauvegarder. Les inconvénients de la sauvegarde au niveau applicatif sont cependant nombreux. Premièrement, il est nécessaire de modifier le code source, ce qui n'est pas toujours forcément possible. De plus, les sauvegardes se font à des moments précis dans l'exécution du programme. Ainsi, il parait délicat d'effectuer une sauvegarde au milieu d'une boucle ! Dès lors, les sauvegardes risquent d'être espacées, perdant beaucoup de leur intérêt. Enfin, en plus de programmer la génération de points de reprise, le programmeur doit programmer également leur restauration.

- ✓ Une librairie (ou bibliothèque) liée à l'application peut initier les points de reprise. Les points de reprise sont alors sauvegardés par la librairie lors des appels de fonctions. Les implémentations au niveau librairie posent aussi quelques problèmes. Ainsi, s'il n'y a pas besoin de modifier le code source, il est nécessaire de réeffectuer l'édition de liens. La contrainte principale est le fait que l'utilisation d'une telle librairie limite les appels systèmes que l'application peut utiliser. Il en résulte que bon nombre d'applications ne peuvent être sauvegardées.

- ✓ La dernière solution, est l'intégration dans le système d'exploitation de la génération des points de reprise. Le système d'exploitation peut fournir une primitive d'initiation des points de reprise ou les initier de façon transparente. Dans ce dernier cas, aucune

modification des applications ou librairies n'est nécessaire. Les mises en œuvre système suppriment les limitations des approches précédentes. En effet, le noyau a accès à toutes les structures de données représentant l'état du processus, dès lors, il y a très peu de restrictions sur les applications qui peuvent être restaurées. Cependant, peu d'implémentations ont été réalisées, compte tenu de la complexité de la mise en œuvre de points de reprise dans le système d'exploitation [18].

4.3.3 Contenu à sauvegarder dans un point de reprise

Dans la suite de cette partie, on admettra que le point de reprise est pris au niveau système permettant ainsi le plus de flexibilité et de transparence.

- **Sauvegarde générale**

Pour réaliser un point de reprise, il est nécessaire d'extraire l'état du processus. Cet état doit également être extrait pour les fonctionnalités de duplication et de migration de processus. C'est pourquoi, les différentes implémentations de ces fonctionnalités au niveau système utilisent la notion de virtualisation de processus.

Un fichier de points de reprise doit contenir tout ce qui est nécessaire pour redémarrer le processus, éventuellement sur un autre nœud. Il faut donc normalement conserver le tas, la pile, le code, l'état des registres dans le système d'exploitation. Il est également nécessaire de sauvegarder le répertoire courant, les comptes à rebours, l'identité de l'utilisateur et celle du processus, l'arborescence des processus.

Une première optimisation peut être faite en proposant de ne pas sauvegarder le code puisque généralement celui-ci est déjà disponible sur le disque.

De la même manière, il faut conserver les librairies partagées liées à l'application. Cependant, celles-ci sont généralement présentes sur le disque du nœud de reprise, il n'est donc pas obligatoire de les sauvegarder.

La création du point de reprise pouvant avoir lieu à n'importe quel moment au cours de l'exécution du processus, il est nécessaire de conserver l'état des signaux et du gestionnaire de signaux [20].

- **Gestion des Entrées/Sorties**

La gestion des Entrées/Sorties est l'un des problèmes délicats pour la reprise d'un processus, et la gestion des fichiers ouverts en fait partie. Ainsi, il ne suffit pas de savoir quel fichier était ouvert lors du point de reprise, mais il s'agit bien de ramener le fichier dans son état lors de la restauration. Pour les fichiers ouverts en écriture, cela nécessite donc de les sauvegarder, pour ceux ouverts en lecture il suffit de les ouvrir à nouveau.

Chaque processus est rattaché à un terminal : une entrée standard et une sortie standard. Lors de la reprise, ce terminal peut exister ou non. Il faut donc s'en assurer pour réaliser le recouvrement et mettre à jour les structures de données correspondantes. Il est important de souligner que certains processus paramètrent le monde du terminal. Il est essentiel de le sauvegarder également [21].

4.4 Les algorithmes de Glanage de Cellules

Appelés aussi ramasse-miettes (Garbage Collection). Dans les applications distribuées, plus l'application progresse plus les informations rassemblées dans le support de stockage deviennent importantes. Un sous-ensemble de ces informations peut devenir inutile pour le recouvrement. Et puisque les points de reprise consomment des ressources de stockage alors la procédure de Ramasse-miettes est donc indispensable pour la suppression de telles informations inutiles. Cette procédure n'est lancée qu'après l'identification d'état global cohérent le plus récent [20]. Ainsi, toutes les informations concernant les événements qui se sont produits avant cette ligne de reprise seront détruites.

5. Bilan et Conclusion

Après quelques définitions et concepts généraux, nous avons présenté dans ce chapitre la problématique de la tolérance aux fautes par recouvrement arrière dans les systèmes réparti/parallèles, en mettant l'accent sur les aspects essentiels : la cohérence d'état global, le déterminisme d'exécution et les mécanismes de mise en œuvre de la stratégie de reprise. Ce dernier point repose principalement sur la détection de défaillance, réalisation et propriétés d'une mémoire stable pour le stockage des points de reprise et la sauvegarde de l'état de processus.

Une classification des différents protocoles de reprise sera présentée dans le chapitre suivant.

Chapitre 3

1. Introduction

Après avoir abordé dans le chapitre précédent la problématique de la reprise sur erreur par recouvrement arrière, nous allons examiner dans les sections qui suivent les différents protocoles de recouvrement arrière dédiés aux systèmes répartis fixes. Nous présentons également deux analyses comparatives : une comparaison d'adaptabilité en fonction de l'application et une autre selon les caractéristiques propres des différentes stratégies.

2. Taxonomie des protocoles de recouvrement arrière

Les protocoles de recouvrement arrière sont classés en deux classes : les protocoles de recouvrement arrière basés sur les points de reprises et les protocoles de recouvrement arrière basés sur la journalisation des messages.

2.1 Protocoles de reprise à base de point de reprise

La tolérance aux fautes par points de reprise (ou checkpointing) utilise le fait qu'un processus en cours d'exécution va être capable à tout moment de prendre un enregistrement complet de son état appelé point de reprise[2]. A la suite d'une erreur, ce type de protocole restaure l'état du système associé au plus récent ensemble consistant de points de reprise, cet ensemble est appelé la ligne de recouvrement [13]. Ces protocoles sont moins restrictifs et plus simples à implémenter par rapport aux protocoles du recouvrement arrière basés sur la journalisation, mais ils ne garantissent pas que l'exécution avant la panne puisse être régénérée de façon déterministe après la reprise [13]. Pour former cette ligne de recouvrement, trois stratégies sont possibles :

2.1.1 Points de reprise non coordonnés (indépendants)

Dans ce protocole, chaque processus décide de manière totalement indépendante, quand un point de reprise doit être effectué. L'avantage est que cette autonomie permet à chaque processus d'effectuer son point de reprise au moment le plus convenable. Par

[2] Dans tout le document les termes point de reprise et checkpoint ont le même sens ; par contre le mot checkpointing désigne la détermination du checkpoint global

exemple, un processus peut réduire le surcoût de temps en effectuant les points de reprise quand la quantité d'informations est minimale [33].

Ceci rend cette technique intéressante lors des exécutions sans fautes. Il faut tout de même pondérer cette affirmation, en effet, La cohérence des états globaux formés n'est donc pas assurée et c'est lors d'une reprise que le protocole recherche une ligne de recouvrement parmi les états globaux formés. Cette recherche s'effectue par la création d'un graphe de dépendance entre les points de reprises locaux [26], idée basée sur la dépendance causale de Lamport.

Les inconvénients de ce protocole sont d'abord, la possibilité de l'apparition de l'**effet domino**, qui a pour conséquence de ramener l'application dans son état initial, perdant ainsi tout le calcul déjà effectué. Un processus peut effectuer un point de reprise inutile quand ceci ne pourra faire partie d'un état global cohérent, le calcul de la ligne de recouvrement lors de la reprise peut s'avérer coûteux, surtout si le nombre de processus est grand et les communications importantes. De plus, plusieurs points de reprise doivent être sauvegardés par chaque processus, ce qui entraîne un surplus d'espace de stockage [13].

Enfin, ce protocole nécessite des algorithmes de ramasse-miettes (Garbage Collection) pour éliminer les points de reprise inutiles.

2.1.2 Points de reprise coordonnés (synchronisés)

Technique de base

Les protocoles adoptant cette approche (sauvegarde coordonnée) sont basés sur les travaux de Chandy et Lamport [14]. Ces algorithmes sont également appelés synchrones, cohérents ou globaux. Etablir un point de reprise coordonné consiste à synchroniser les processus lors de la sauvegarde des états locaux afin de garantir que l'état global obtenu à l'aide de l'ensemble des états locaux soit cohérent. La ligne de recouvrement est ainsi créée lors de l'établissement du point de reprise. De ce fait, l'application peut être relancée en faisant repartir chacun des processus au niveau de son dernier point de reprise, aucun autre calcul supplémentaire n'est nécessaire. On remarque qu'il suffit alors de conserver le dernier point de reprise de chaque processus, ce qui minimise l'espace de stockage. Ce protocole simplifie la réexécution et n'est pas susceptible d'entraîner l'effet domino puisque chaque processus est toujours ré-exécuté à partir de l'état correspondant à son point de reprise le plus récent.

Le principal inconvénient de cette stratégie est la latence importante qu'elle implique lors de la sauvegarde des points de reprise, étant donné qu'elle nécessite une coordination entre les processus. Le problème qui se pose ici est la perte de performances y compris en l'absence de fautes. En effet, plus le nombre de processus participant augmente, plus la latence risque d'être importante, chaque processus doit attendre la fin de l'opération avant de pouvoir reprendre son exécution normale.

Les stratégies de Checkpointing coordonné peuvent être à décision centralisée (c'est-à-dire qu'il y a un coordinateur qui coordonne les interactions entre les processus lors de l'exécution du protocole), ou distribuée (c'est-à-dire qu'il n'y a pas de coordinateur, mais plutôt tous les processus sont impliqués dans l'exécution du protocole).

Optimisation de la technique de base

Le principal inconvénient de cette technique est le surcoût introduit par la coordination de tous les processus participants. Aussi, afin d'optimiser les performances de la sauvegarde coordonnée, plusieurs techniques ont été proposées :

– **la sauvegarde coordonnée non bloquante** [14] évite de bloquer le processus durant la phase de sauvegarde en créant un processus clone chargé de la sauvegarde ;

– **la sauvegarde avec horloge synchronisée** [38] évite la phase de synchronisation par envois de messages, en utilisant une horloge synchronisée ;

– **la sauvegarde coordonnée minimale** [23] évite la phase de synchronisation globale entre tous les processus en se basant sur le fait qu'un processus n'a besoin de se coordonner qu'avec les processus avec lesquels il a des dépendances.

La quantité de donnée à sauvegarder peut être grandement diminuée en utilisant une technique de point de reprise **incrémentale**. Seules les données modifiées depuis le dernier point de reprise sont alors sauvegardées.

Une autre optimisation apportée dans [32] pour les systèmes à mémoire virtuelle partagée exploite la réplication des données inhérente à ces systèmes pour la sauvegarde des points de reprise. Ceci est réalisé en étendant le protocole de cohérence du système. La présence de réplique sur différents sites permet de minimiser les transferts de données. De

plus, le fait que les données soient conservées dans la mémoire de plusieurs nœuds et non sur disque permet des accès rapides aux données de reprise.

2.1.3 Points de reprise induits par les communications

Dans cette technique, la synchronisation se fait de manière « paresseuse », en utilisant les messages de l'application. Les points de reprise induits par les communications (Communication-Induced Checkpointing CIC) [25], est un compromis entre la sauvegarde coordonnée et la sauvegarde non coordonnée.

L'idée principale est d'une part, d'éviter la coordination des processus en permettant les points de reprise non coordonnés des processus, appelés points de reprise locaux, et d'autre part, d'éviter l'effet domino en forçant un processus à sauvegarder son état en cas d'évaluation de la ligne de recouvrement. Cette sauvegarde est alors appelée point de reprise forcé.

Cette technique [24] a pour idée d'initier les points de reprise en fonction des messages émis et reçus entre les processus. Aucun message spécifique de coordination 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 de ces informations 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.

Pour résumer, le principe de CIC est d'étendre un ensemble de points de reprise locaux (sauvegardes locales) de l'application (système) à un ensemble de points de reprise constituant un état global cohérent (sauvegarde globale). Ce principe a été proposé formellement dans [30], par la définition des chemins zigzagants qui formalise les conditions de cohérence d'un état global.

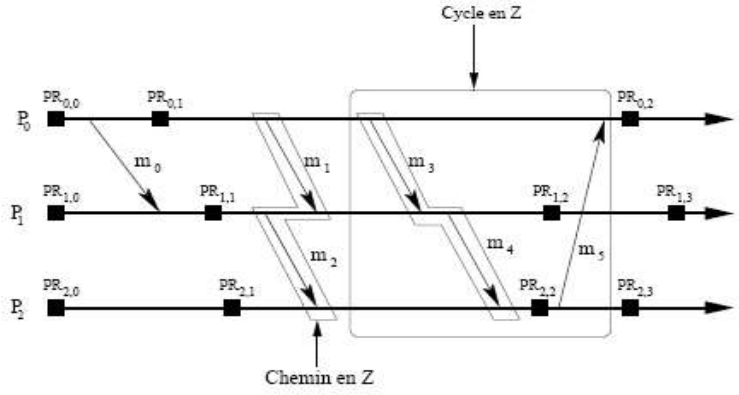


Figure 3.1 : les chemins en Z et les cycles en Z.

Un chemin en Z est une séquence particulière des messages qui relient deux points de reprise. Dans la figure 3.1, les messages [m1, m2] et [m3, m4] représentent des chemins en Z entre les points de reprise $PR_{0,1}$ et $PR_{2,2}$. Un cycle en Z est un chemin en Z qui commence et se termine par le même point de reprise. Le chemin [m5, m3, m4] est un cycle en Z qui commence et se termine par le point de reprise $PR_{2,2}$. On s'intéresse aux cycles en Z puisqu'il a été prouvé qu'un point de reprise est inutile si et seulement si il fait part d'un cycle en Z [27]. Ainsi, on peut éviter les points de reprise inutiles en s'assurant que les chemins en Z ne deviennent jamais des cycles en Z.

Ces techniques ont comme avantage *d'éviter l'effet domino* tout en ne nécessitant pas la coordination de tous les processus de l'application, les points de reprise forcés évitent la sauvegarde inutile, en plus contrairement aux points de reprise coordonnés, elles n'entrainent pas de surcoût de synchronisation.

En revanche, ces techniques présentent les inconvénients suivant : La difficulté dans la mise en œuvre de la méthode de décision quant à la nécessité de créer un point de reprise. De plus, si les messages de l'application sont très petits, l'encapsulation dans chaque message des informations du protocole entraîne un surcoût qui peut être important.

Nous distinguons deux types de checkpointing induit par les communications :

2.1.3.1 Les protocoles basés sur le modèle de communication

Dans cette technique, le système maintient les structures du point de reprise et de communications qui vont prévenir l'effet domino [27].

Elle a pour but d'empêcher les situations de communications et de Checkpointing qui pourraient amener le système dans un état inconsistant à partir des points de reprise existant. On peut alors détecter la possibilité de telles situations se présentant dans le système. Un point de reprise est habituellement forcé pour empêcher ces situations indésirables. Toutes les informations nécessaires pour la réexécution sont insérées dans les messages de l'application. La décision pour forcer un point de reprise est obtenue localement en fonction des informations disponibles.

Par conséquent, sous ce modèle de checkpointing, il est possible que deux processus

aient détecté le potentiel pour qu'il y ait un état global incohérent. Ils forcent indépendamment les points de reprise locaux pour empêcher la formation de cette situation indésirable, qui peut ne pas être réelle ou qui pourrait être empêchée par la prise d'un unique point de reprise forcé. Ainsi, ce type de checkpointing prend beaucoup de points de reprise forcés que d'habitude parce qu'il n'y a pas de coordination explicite ; aucun processus n'a d'informations complètes sur l'état du système [28].

2.1.3.2 Les protocoles basés sur les index

Dans la technique de coordination basée sur les index, on se base sur l'indexation des points de reprise locaux et obligatoires. Le but est d'assigner aux points de reprise des index qui croissent de manière monotone, de telle sorte que les points de reprise, ayant le même index au niveau de tous les processus, forment un état global cohérent. Les index sont insérés dans les messages d'application pour aider les récepteurs à décider quand ils devraient forcer un point de reprise [29],[30].

2.2 Protocoles de reprise fondés sur la journalisation des messages

Le principe du mécanisme de journalisation (message logging) est la sauvegarde de l'histoire d'exécution de l'application. Ces mécanismes s'appuient explicitement sur le fait qu'un processus, peut être modélisé par une séquence d'intervalles d'états déterministes (hypothèse PWD) [16]. Chaque intervalle d'état débute par un événement non déterministe, dont l'enregistrement permet la reconstruction de l'état du processus. Un événement non déterministe peut être la réception d'un message ou un événement interne au processus.

En effet, le principe de la reprise basée sur la journalisation est que durant l'exécution normale, c-à-d sans panne, chaque processus stocke dans un journal sur mémoire stable les informations correspondant à tous les événements non déterministes qu'il observe. Après une panne, le processus défaillant reconstruit son état d'avant la panne à partir de son état initial et en utilisant son journal, afin de rejouer les événements non déterministes exactement comme ils se sont produits avant la panne. Pour éviter la reconstruction de l'état d'un processus défaillant à partir de son état initial, des sauvegardes périodiques de l'état du processus (point de reprise) sont effectuées.

Dans le cadre d'un protocole à journalisation des messages, la sauvegarde de points de reprise n'est pas obligatoire mais est fortement conseillée puisqu'elle permet de diminuer le nombre de messages journalisés, par conséquent réduire l'envergure du retour arrière lors du recouvrement [16].

Par la suite, nous considérerons donc uniquement les protocoles à journalisation des messages qui utilisent en complément une sauvegarde non coordonnée de points de reprise locaux.

On distingue deux catégories de Journalisation :

- ✓ Journalisation basée sur la réception ou les messages entrant sont enregistrés (par le récepteur).
- ✓ Journalisation basée sur l'émission ou les messages sortant sont enregistrés (par l'émetteur).
- ✓ Journalisation dans un serveur centralisé [24].

Pour faire un retour arrière dans un protocole basé sur la journalisation des messages on dispose de deux variantes :

- ✓ **Retour arrière synchrone** : tous les processus sont impliqués dans le protocole de recouvrement c'est-à-dire qu'on calcule la ligne de recouvrement la plus récente en fonction des dépendances et de l'information dans les journaux.
- ✓ **Retour arrière asynchrone** : il y a une diffusion d'un message de retour arrière lorsqu'un processus effectue un retour arrière. A la réception du message, chaque processus décide s'il est concerné et propage à son tour le message de retour arrière [26].

Les protocoles à journalisation se classent en plusieurs catégories que nous détaillons par la suite: journalisation pessimiste, optimiste et causale.

2.2.1 Journalisation pessimiste (pessimistic logging)

Ce protocole fait l'hypothèse qu'une panne peut apparaître après chaque événement indéterministe durant l'exécution. Ainsi, le déterminant de chaque événement indéterministe est enregistré sur un support fiable de stockage avant que cet événement ne puisse affecter l'exécution, on parle alors d'enregistrement **synchrone** (synchronous logging). Dans le cas ou les événements non déterministes sont uniquement des réceptions de messages, aucun processus P ne peut envoyer un message m avant que tous les messages reçus avant l'émission m ne soient journalisés sur la mémoire stable [37].

De façon plus claire 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.

Les avantages de ce protocole sont d'abord que les processus qui tombent en panne peuvent être ré-exécutés à partir de leur plus récent point de reprise et que la ré-exécution des processus en panne est simplifiée puisque les effets d'une panne n'affectent que le processus concerné, en plus cette technique ne crée jamais de processus orphelin. Enfin, le glanage de cellules est simple puisque on peut supprimer les anciens points de reprise car ils ne seront plus jamais utilisés. Cette technique est utilisée dans les systèmes ou les interactions avec le monde extérieur sont très fréquentes.

Mais son inconvénient est qu'il bloque le processus avant chaque retrait de message et donc la journalisation pessimiste introduit un surcoût important durant l'exécution normale, c-à-d, sans panne.

2.2.2 Journalisation optimiste (optimistic logging)

Dans le cas de la journalisation optimiste, les messages reçus ne sont pas forcément enregistrés immédiatement sur une mémoire stable afin d'améliorer les performances. L'enregistrement des messages en transit dans le système se fait de façon **asynchrone,** de manière à grouper les enregistrements sur mémoire stable qui représentent un surcoût considérable. Les messages sont conservés en mémoire volatile avant un enregistrement sur mémoire stable dont le moment dépend du protocole. Ces protocoles font donc l'hypothèse « optimiste » que ces enregistrements seront terminés avant qu'une panne n'arrive, de façon à ce que tous les événements puissent être rejoués en cas de panne d'un processus.

Ils doivent cependant gérer le cas où une panne survient avant que tous les messages ne soient enregistrés : les informations enregistrées en mémoire volatile du processus tombé en panne sont alors perdues, et le processus fautif ne peut plus rejouer une exécution équivalente après sa reprise. Dans ce cas, les messages qu'il a envoyé après la dernière sauvegarde sur mémoire stable deviennent des messages orphelins, puisqu'ils ne seront peut être jamais envoyés dans la nouvelle exécution. On dit que les processus qui ont reçus

ces messages deviennent à leur tour des processus orphelins ; une fois orphelin, un processus rend orphelin tous processus à qui il envoie un message.

Ces processus orphelins ne sont plus dans un état cohérent avec le reste du système, et doivent donc eux aussi reprendre. Un état recouvrable est donc constitué de l'ensemble des points de reprise et journaux de messages de tous les processus orphelins de l'exécution.

Il est évident que, la journalisation optimiste ne garantit pas toujours la « condition de non orphelinité ». Ceci rend l'opération de reprise compliquée, car il faut éventuellement retourner en arrière plusieurs fois avant de trouver un état cohérent.

Les inconvénients de cette approche résident dans le risque d'un retour à l'état initial du calcul (effet domino). De plus, un surcoût important dû au calcul de l'état global cohérent est introduit durant la reprise.

Regardons par exemple la figure 3.2, et supposons que l'on utilise un protocole optimiste basé sur la réception. Si R tombe en panne avant que m4 soit enregistré (mais après émission du message m5) puis reprend depuis le point de reprise C_R, Q devient alors orphelin et doit reprendre en C_Q pour défaire la réception de m5.

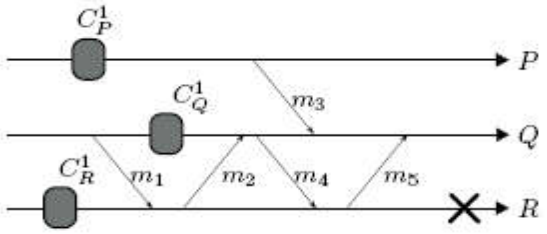


Figure 3.2 : Le processus Q doit lui aussi reprendre si m4 n'a pas été journalisé

Les processus doivent alors maintenir des informations sur les relations causales entre eux, pour pouvoir déterminer, au moment de la reprise, quels sont les processus qui doivent reprendre pour que l'état global reste cohérent.

2.2.3 Journalisation causale (causal logging)

La troisième catégorie de journalisation est la journalisation causale ou répartie [34]. Ce protocole possède les performances de l'enregistrement optimiste en retenant les avantages de l'enregistrement pessimiste.

Comme l'enregistrement optimiste, les messages ne sont pas forcément journalisés immédiatement sur une mémoire stable et comme l'enregistrement pessimiste, Il donne plus d'autonomie aux processus en les isolant des effets des pannes qui peuvent apparaître dans les autres processus (i.e. la panne d'un processus ne peut pas rendre orphelin un autre processus du système).

Pour ce faire, l'enregistrement causal assure que le déterminant de chaque événement indéterministe qui précède causalement l'état du processus soit stocké dans le support fiable de stockage ou bien disponible localement dans la mémoire volatile du processus lui même. Dans l'exemple de la figure 3.3, les messages m4 et m5 peuvent être perdus après une panne des processus P1 et P2, le processus P0 à l'état X doit avoir enregistré les déterminants des événements indéterministes qui précèdent causalement cet état, à savoir, la réception des messages m0, m1, m2 et m3.

Les déterminants de chacun de ces événements indéterministes sont soit enregistrés sur le support fiable de stockage, soit disponibles dans la mémoire volatile du processus P0. Ces déterminants contiennent l'ordre dans lequel leurs récepteurs d'origine ont reçu les messages correspondants. L'expéditeur du message enregistre le contenu de celui-ci. Ainsi, le processus P0 sera capable de guider la ré-exécution de P1 et P2 puisqu'il connaît l'ordre dans lequel P1 doit recevoir le message m0 et m2 pour atteindre l'état dans lequel P1 envoie le message m3 et ainsi de suite. Dans ce protocole, un processus fait porter les déterminants existants dans sa mémoire volatile sur les messages qu'il envoie. Après la réception d'un message, le processus d'abord ajoute tous les déterminants portés par celui-ci dans sa mémoire volatile, puis, délivre le message à l'application.

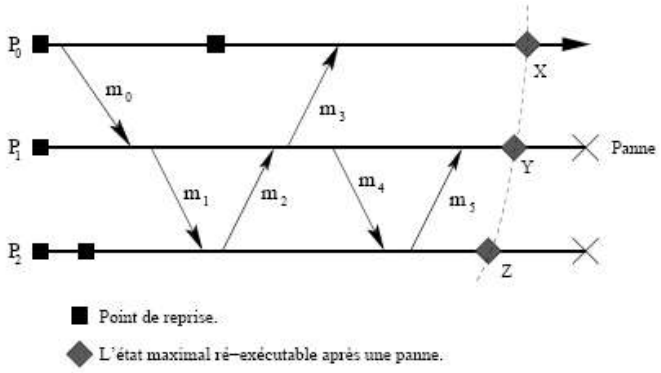


Figure 3.3 : l'enregistrement causal.

L'enregistrement causal limite le retour arrière d'un processus qui tombe en panne au plus récent point de reprise. Cela réduit l'espace de stockage utilisé et la quantité du travail à refaire en cas de panne (et le temps supplémentaire requis) au détriment d'un mécanisme de ré-exécution plus complexe.

3. Analyse comparative globale

Comme nous l'avons vu dans la section précédente, il existe plusieurs protocoles de recouvrement arrière basé sur la mémoire stable. Tous ces protocoles se basent sur la sauvegarde d'un état des processus du système d'une part, et sur la construction d'un état global cohérent d'autre part. Plusieurs critères peuvent être utilisés pour comparer ces différents protocoles.

Dans ce qui suit, on considère deux comparaisons: une comparaison d'adaptabilité en fonction de l'application et une autre selon les caractéristiques propres des différentes stratégies.

3.1 Comparaison en fonction du système et de l'application

Nous proposons ici une analyse comparative des protocoles de points de reprise et des protocoles de journalisation en fonction de certaines caractéristiques du système et de l'application. Nous détaillons ici quatre critères qui nous semblent déterminants et qui sont, hormis la fréquence des pannes, dépendants de l'application.

- ✓ **Fréquence des pannes**

La valeur caractéristique de la fréquence des pannes dans un système distribué est le MTBF (Mean Time Between Failures), c'est à dire le temps moyen entre deux pannes consécutives de n'importe quelle ressource physique du système. Cette valeur varie d'une minute pour les systèmes constitués de stations de travail [35], à deux semaines pour une grappe de 1024 machines dédiée au calcul haute-performance [36].

Dans le cas d'un système où le MTBF est très petit, de l'ordre de quelques minutes, il faut privilégier la rapidité de la reprise. On choisira alors une approche par journalisation de message, dans laquelle généralement seul le processus fautif doit reprendre. Lemarinier et al. ont montré dans [76] que le surcoût plus élevé des protocoles par journalisation de

messages par rapport à une approche par points de reprise synchronisés était compensé par le temps de recouvrement lorsque le MTBF devient petit. En effet, les protocoles de points de reprise nécessitent une reprise de tout le système. Dans un cas extrême, si le temps moyen entre deux pannes est inférieur au temps de reprise globale, le système pourrait alors rester indéfiniment dans l'état de reprise.

Il faut exclure les protocoles ne supportant pas les pannes simultanées (ou seulement k pannes simultanées, où k est un paramètre du protocole), comme certaines approches par journalisation optimiste et causale.

- ✓ **Nombre de processus**

Les protocoles qui nécessitent la reprise de tous les processus en cas de panne de l'un d'entre eux sont mal adaptés aux systèmes comportant un grand nombre de processus. Les protocoles de journalisation des messages sont donc plus adaptés aux applications de grande taille que les protocoles par points de reprise, à l'exception de certaines approches par journalisation causale, qui induisent des communications d'un processus vers tous les autres lors de la reprise.

- ✓ **Taux de communication, maillage**

Un taux de communication élevé va rendre l'utilisation de protocoles de journalisation très coûteuse pendant une exécution sans panne, surtout dans le cas d'une approche pessimiste. Ces protocoles nécessitent des accès fréquents à la mémoire stable, ce qui a pour effet, en plus du coût de la communication, de créer de la contention au niveau de cette mémoire.

De plus, l'espace de stockage nécessaire pour les points de reprise et les journaux de messages est important, en particulier pour les approches optimistes, pour lesquels le ramassage des points et des journaux devenus inutiles est complexe et nécessite souvent de conserver plusieurs points consécutifs par processus.

Une approche par journalisation causale, qui permet pourtant de minimiser l'utilisation de la mémoire stable, peut aussi induire un surcoût important puisque la taille et la quantité des informations à maintenir durant l'exécution sont proportionnelles au taux de communication.

Les protocoles par points de reprise synchronisés sont insensibles au taux de communication, puisqu'ils ne reposent que sur l'utilisation de messages spécifiques. Ils

représentent donc une solution intéressante pour les applications fortement communicantes.

Les protocoles par points de reprises induits par messages sont eux influencés par le taux de communication. Le comportement de ce type de protocole diffère selon les approches choisis. Dans [24], il a été montré en particulier que les solutions basées sur index génèrent un grand nombre de points de reprise forcés

Lorsque le taux de communication est important ; le surcoût induit par l'utilisation de tels protocoles devient alors prohibitif.

- ✓ **Indéterminisme**

Si le système contient des processus indéterministes, alors les protocoles de journalisation de messages ne peuvent pas être utilisés. En effet, si le comportement de certains processus est la conséquence d'actions indéterministes internes qui ne sont pas observables, ils ne sont alors plus déterministes par morceaux, et deux exécutions avec la même histoire de messages peuvent être différentes. L'état global du système après la reprise d'un processus peut donc être incohérent. On devra alors choisir dans ce cas un protocole par points de reprise.

- ✓ **Communication avec le monde extérieur**

Si le système doit communiquer avec des processus du monde extérieur, c'est-à-dire des processus qui ne peuvent pas être contrôlés par le protocole, ou dont le comportement a des effets de bord sur le monde réel (affichage, impression…), alors les protocoles de points de reprises ne sont pas adaptés. En effet, on ne peut pas faire reprendre ces processus dans un état antérieur en cas de panne.

Si le choix se porte quand même sur un protocole de ce type, il doit être modifié pour prendre en compte ce monde extérieur, par exemple en journalisant les messages entrants et sortants du système.

Le tableau suivant résume les choix adaptés de protocoles de tolérance aux pannes en fonction des caractéristiques de l'application. On note :

++, quand l'utilisation de ce protocole est idéale dans ces conditions.

+, quand le protocole utilisé doit être légèrement modifié pour prendre en compte les caractéristiques de l'application.

--, quand le protocole est utilisable, mais peut conduire à des surcoûts inacceptables.

-- --, lorsque le protocole ne peut pas fonctionner dans ces conditions.

	Point de reprise coordonné	Point de reprise induit par communications	Journalisation pessimiste	Journalisation optimiste
MTBF petit	+	+	++	++
Nombre de processus élevé	--	+	++	++
Taux de communication élevé	++	++	--	+
Non-déterminisme des processus	++	++	-- --	-- --
Communication avec le monde extérieur	+	+	++	++

Tableau 3.1 : Adéquation des différentes techniques en fonction des caractéristiques de l'application.

3.2 Comparaison en fonction des caractéristiques des différentes stratégies

Le tableau présente un résumé comparatif des implications liées aux différentes stratégies de recouvrement arrière, selon huit critères de comparaison qui sont :

- ✓ La possibilité de l'effet domino : (Possible, Non).
- ✓ Le nombre de points de reprise maintenues par processus : (Plusieurs, un seul).
- ✓ L'étendue du retour arrière : (illimité, dernier point de reprise, plusieurs points de reprise Possibles).
- ✓ Le type du protocole de reprise : (Distribué, local).
- ✓ L'existence de l'hypothèse de PWD : (Non, Oui).
- ✓ La complexité du protocole de ramasse-miettes : (Complexe, Simple).
- ✓ L'existence d'un processus orphelin : (Possible, Non).

- ✓ La complexité du recouvrement : (Non, Oui).

	Checkpointing			**Journalisation**		
	Non Coordonné	**Coordonné**	**Induit communication**	**Pessimiste**	**Optimiste**	**Causale**
Effet domino	Possible	Non	Non	Non	Non	Non
points de reprise/processus	Plusieurs	1	Plusieurs	1	Plusieurs	1
Etendue du retour arrière	Illimité	Dernier pt de reprise	Plusieurs pt. possibles	Dernier pt de reprise	Plusieurs pt. possibles	Dernier pt de reprise
Protocole de reprise	Distribué	Distribué	Local	Distribué	Distribué	Distribué
Hypothèse de PWD	Non	Non	Non	Oui	Oui	Oui
Ramasse-miettes	Complexe	Simple	Complexe	Simple	Complexe	Complexe
Processus orphelins	Possible	Non	Possible	Non	Possible	Non
Complexité de reprise	Oui	Non	Oui	Non	Oui	Oui

Tableau 3.2 : Comparaison des différentes stratégies de Checkpointing

4. Bilan et Conclusion

Nous avons présenté dans ce chapitre, les deux grandes familles de protocoles de tolérance aux pannes par recouvrement arrière, à savoir, les protocoles basés sur les points de reprise et ceux basés sur la journalisation des messages. Nous avons montré également, qu'il n'existe pas de solution unique qui soit adaptée à toutes les situations. Selon les propriétés de l'infrastructure physique utilisée mais aussi selon les propriétés de l'application, une solution donnée peut engendrer de mauvaises performances, voir même être totalement inapplicable.

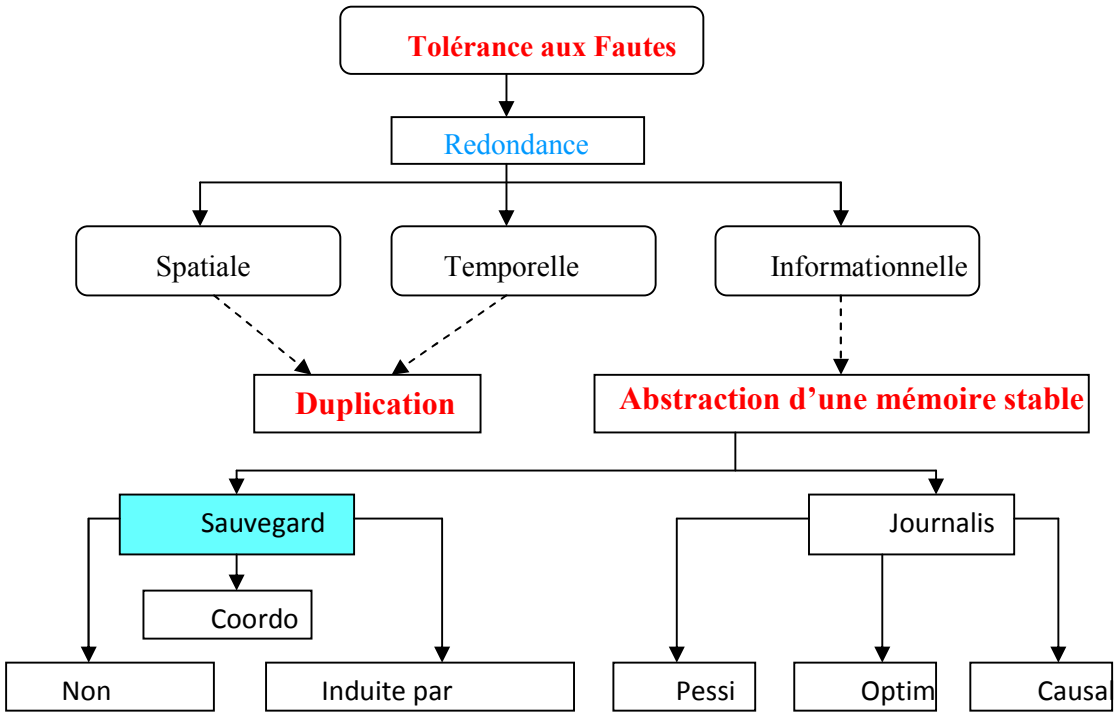


Figure 3.4 : Tolérance aux fautes par mémoire stable [39]

C'est cette constatation qui va motiver le développement d'un protocole par points de reprise adapté à notre contexte de travail présenté dans le chapitre suivant.

Nous allons voir en effet que si les protocoles de journalisation de messages ne peuvent facilement s'appliquer dans ce contexte, les protocoles par points de reprise coordonnés ou induits par message classiques ne peuvent pas s'appliquer directement. Il nous faut donc proposer une solution par points de reprise adaptée; cette solution est présentée dans le chapitre 7.

1. Introduction

Les progrès récents dans le domaine de la technologie des réseaux sans fils et des appareils portables (PC portable, PDA, périphériques PC, capteur, etc.) ont introduit un nouveau paradigme dans le domaine de l'informatique, il s'agit de l'informatique mobile.

Ce nouveau paradigme est caractérisé par la mobilité de ses usagers qui utilisent de plus en plus souvent de multiples environnements ayant des caractéristiques très différentes qui entravent leur mobilité. De plus, avec les avancées récentes, en termes de performances et de miniaturisation, réalisées en microélectronique (microcontrôleur, DSP, transceiver RF, etc.), les applications liées au traitement mobile sans fil devraient être de plus en plus répandues. En effet, les réseaux sans fil offrent une grande flexibilité d'emploi. En particulier, ils permettent la mise en réseaux de sites dont le câblage serait trop onéreux à réaliser dans leur totalité, voir même impossible dans certains cas.

Ce nouvel environnement résultant appelé (Mobile) permet aux unités de calcul une libre mobilité et ne pose aucune restriction sur la localisation des usagers.

La mobilité (ou le **nomadisme**) est le nouveau mode de communication utilisé. Elle engendre de nouvelles caractéristiques propres à l'environnement mobile: une fréquente déconnexion, un débit de communication modeste, et des sources d'énergie limitées. C'est pourquoi les contraintes de l'environnement mobile nous obligent à changer la vision classique des problèmes liés aux systèmes distribués. En effet, les processus distribués doivent tenir compte de l'apparition de la nouvelle entité: "unité mobile" avec toutes ses propriétés (mode de communication, ressources, méthode d'acheminement des données «routage » …etc.).

Les réseaux sans fil peuvent être classés en deux catégories : les réseaux avec infrastructure fixe préexistante, et les réseaux sans infrastructure. Dans la première catégorie, le modèle de la communication utilisé est généralement le modèle cellulaire. Dans ce modèle, un point d'accès assure la liaison entres les terminaux mobiles et le réseau câblé, les utilisateurs peuvent se déplacer de manière transparente (sans perte de connectivité) d'un point d'accès à un autre à l'intérieur du réseau. La deuxième catégorie est celle des réseaux ad hoc que l'on verra en détail dans le chapitre suivant.

Dans ce chapitre, nous allons faire un état des technologies actuelles des environnements mobiles, puis nous faisons une comparaison avec les environnements fixes.

2. Du fixe vers l'environnement mobile …

De récentes percées technologiques dans le domaine des terminaux portables (unités mobiles) et dans celui des réseaux sans-fil ont conduit à une forte évolution des environnements mobiles. Les environnements mobiles se caractérisent par la présence d'un ou de plusieurs terminaux portables ayant chacun un ou plusieurs moyens de communication sans-fil. Ces interfaces de communication sans-fil permettent aux terminaux, tout en se déplaçant, de communiquer entre eux ou avec des stations fixes. Ces environnements présentent de grandes différences par rapport aux environnements traditionnels ou fixes [40]. Pour des raisons de taille et de poids, les terminaux portables disposent de ressources moins importantes par rapport à celles qu'offrent des stations fixes. De plus, l'utilisation de ces ressources est limitée dans le temps puisqu'elle dépend d'une source d'énergie limitée, la batterie. En ce qui concerne les réseaux de communication sans-fil, ils offrent une bande passante beaucoup plus faible et variable que les réseaux filaires. En effet, ces communications sont soumises à de fortes variations résultant des interférences du signal avec les éléments physiques alentours. Ces variations conduisent, dans le cas extrême, à la déconnexion lorsque le signal ne parvient plus au terminal portable.

2.1 Les terminaux portables

Les terminaux portables ou les unités mobiles peuvent être décrits par trois caractéristiques : les ressources, l'encombrement et l'autonomie. Ces différentes caractéristiques ne sont pas indépendantes les unes des autres. En effet, réduire grandement la taille et le poids du terminal portable ne permet pas d'avoir d'importantes ressources mais permet d'avoir une bonne autonomie. À l'inverse, un terminal portable plus volumineux peut offrir plus de ressources, mais il consomme alors plus d'énergie et bénéficie donc d'une autonomie moindre. Différents compromis existent pour satisfaire aux exigences des utilisateurs (voir la synthèse dans le tableau 4.1). Parmi les terminaux portables, on cite :

A. ***Les assistants personnels numériques*** (PDA : Personal Digital Assistants) sont conçus pour tenir dans la main ou être mis dans une poche. Ils sont dépourvus de clavier et la saisie s'effectue au moyen d'un stylet et d'un écran tactile. En ce qui concerne les ressources et l'autonomie, deux compromis existent :

(i) **Les organiseurs (Organizers, Palmtops)** [55], [54] possèdent des ressources moindres (processeur entre 16 et 33 MHz et mémoire entre 8 et 16 Mo) pour une excellente autonomie (entre 2 et 8 semaines); ces restrictions limitent grandement leurs fonctionnalités aux applications de consultation et saisie d'informations personnelles (agenda, répertoire, liste de tâches, mémentos, etc.), et

(ii) L**es assistants personnels de type Pocket PC, Palm-size PC** [43] possèdent de plus grandes ressources (processeur entre 131 et 400 MHz et mémoire de 16 à 64 Mo) pour une autonomie moindre (de 6 à 15 heures); les fonctionnalités offertes permettent d'implanter les systèmes d'exploitation (Windows CE , Linux) avec la plupart des applications prévues pour ces systèmes (Internet Explorer, Netscape, etc).

B. ***Les ordinateurs de poche*** (**Handheld Computers)** [46] sont un peu plus grands que les assistants personnels. En plus de l'écran tactile et du stylet, ils disposent d'un petit clavier pour la saisie de texte. Au niveau des ressources, la puissance du processeur varie entre 90 et 350 MHz, la mémoire varie entre 16 et 32 Mo. Du fait de l'écran, l'autonomie diminue encore et varie entre 6 et 10 heures. Les fonctionnalités disponibles sont similaires à celles des assistants personnels, si ce n'est que le clavier rend plus facile l'utilisation des applications de type saisie et traitement de texte.

C. ***Les ordinateurs portables*** (Notebooks, Laptops) [44], [45] nécessitent l'utilisation d'une sacoche pour le transport. La saisie peut se faire aisément à l'aide du clavier et la souris est remplacée par une tablette sensitive (Touchpad) ou un ergot de pointage (Trackpoint). Les ressources sont comparables à celles d'une station de travail fixe : la puissance du processeur varie de 600 MHz à 2.2 GHz et la mémoire de 128 à 2048 Mo. Ces ressources permettent d'utiliser les mêmes systèmes d'exploitation et les mêmes applications que sur une station fixe, mais limités à une autonomie entre 1.5 et 3.5 heures.

De plus, quel que soit le terminal portable, de nouveaux périphériques peuvent être insérés en court d'exécution (à chaud.) en utilisation les possibilités d'extension qu'offrent les standards PC Card1, USB (Universal Serial Bus) et FireWire. Ainsi, il est possible d'ajouter aux terminaux portables des périphériques permettant d'améliorer :

a) le confort d'utilisation des entrées/sorties avec, par exemple, le branchement possible d'un clavier et/ou d'une souris, l'utilisation d'un écran extérieur, ou le branchement d'une imprimante,

b) les possibilités de stockage avec de la mémoire supplémentaire ou des disques durs,

c) l'autonomie avec l'ajout de batteries.

2.2 Comparaisons avec les stations fixes

Les stations fixes sont conçues pour offrir à l'utilisateur le meilleur confort d'utilisation possible. Elles n'ont pas de contraintes de portabilité et d'autonomie et peuvent donc se permettre d'utiliser des ressources importantes (voir leurs caractéristiques dans le tableau 4.1).

	Ressources				**Encombrement**		**Autonomie**
	Taille de l'écran (pouces) Résolution max. (pixels) Nombre de couleurs	Processeur (MHZ)	Mémoire (extensible à) (Mo)	Disque Dur (Go)	Dimensions (Lxlx e cm)	Poids (Kg)	
Assistants personnels (PDA) ✓ **Organiseurs** (organizers, palmtops)	3,6 - 3,8 160 x 160 16(derniers modèles à 65536)	16 – 33	8 -16(64)	–	15 x 10 x (1-25)	0.1– 0.25	2 – 8 Semaines
✓ **Pocket Pc, Palm Size Pc**	360 x 320 65536	131 - 400	16 -64(128)				6 -15 h
Ordinateurs de poche (handheld computers)	6,5 – 10 640 x 480 65536	90 - 350	16 -32(96)	_	(18x25)x (10-30) x (2-5)	0.25 -1	6 -10 h
Ordinateurs portables (notebooks, Laptops)	10 -15,1 1400 x 1050 16 millions	600 - 2200	128 -512 (1024)	10 -60	(26-33) x (22-28) x (2-5)	1.5 - 4	1.5-3.5 h
Stations fixes	17 -22 2048 x 1536 16 millions	800 - 3000	256-512 (2048)	20 - 80	_	_	_

Tableau 4.1 : Caractéristiques physiques des terminaux.

Les ordinateurs portables sont les seuls terminaux portables pouvant offrir un confort d'utilisation et des ressources proches de celles des stations fixes. Cependant, si l'on compare les meilleurs ordinateurs portables actuels avec les meilleurs stations fixes actuelles, on constate que ces dernières sont, quand même, plus puissantes que leurs homologues portables : résolution 45% plus grande, processeurs 40% plus puissants, mémoire pouvant aller jusque 100% plus importante et disques durs pouvant aller jusque 300% plus grands. De plus, les ordinateurs portables sont contraints par une source d'énergie limitée et des restrictions de taille et de poids, limitant ainsi les utilisations possibles.

Ces différences ne sont pas nouvelles mais elles ne sont pas non plus appelées à disparaître car les évolutions technologiques bénéficient aussi bien aux terminaux portables qu'aux stations fixes [42].

3. Le modèle des environnements mobiles

Un environnement mobile est un système composé de sites mobiles et qui permet à ses utilisateurs d'accéder à l'information indépendamment de leurs positions géographiques. Les réseaux mobiles (sans fil), peuvent être classés en deux catégories : les réseaux avec infrastructure et les réseaux sans infrastructure (Figure 4.1).

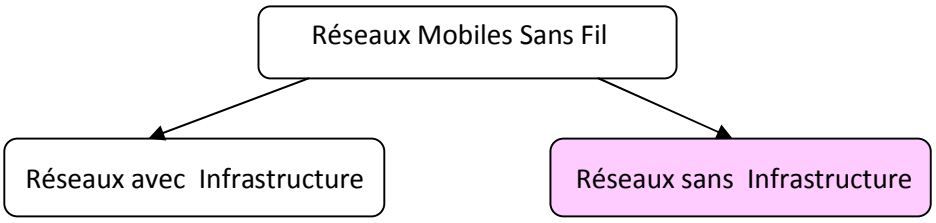


Figure 4.1: schéma de décomposition des réseaux mobiles sans fil

3.1 Les réseaux avec infrastructure

Appelés aussi réseaux sans fil **à point d'accès**, ou encore réseaux **cellulaires**. La figure 4.2 présente l'architecture d'un réseau sans-fil à point d'accès. On peut y distinguer deux sortes d'entités [57] :

1- **Les stations fixes** (les **sites fixes** du réseau filaire : Wired Network)

2- **Les terminaux portables** (les **sites mobiles** : Wireless Network).

Les stations fixes sont interconnectées entre elles à l'aide d'un réseau classique de communication filaire, comme un réseau Ethernet local. Certaines de ces stations, appelées **stations de base (SB)**, jouent le rôle d'infrastructure de communication pour le réseau sans fil. Elles possèdent une interface de communication sans fil leur permettant de communiquer avec les terminaux portables se trouvant à portée de communication. Cette portée de communication définit une zone géographique appelée **cellule**.

La station de base est le passage obligé pour toute communication sans fil entre terminaux portables ou avec l'extérieur. Une unité mobile ne peut être, à un instant donné, directement connectée qu'à une seule station de base. Elle peut communiquer avec les autres sites à travers la station à laquelle elle est directement rattachée.

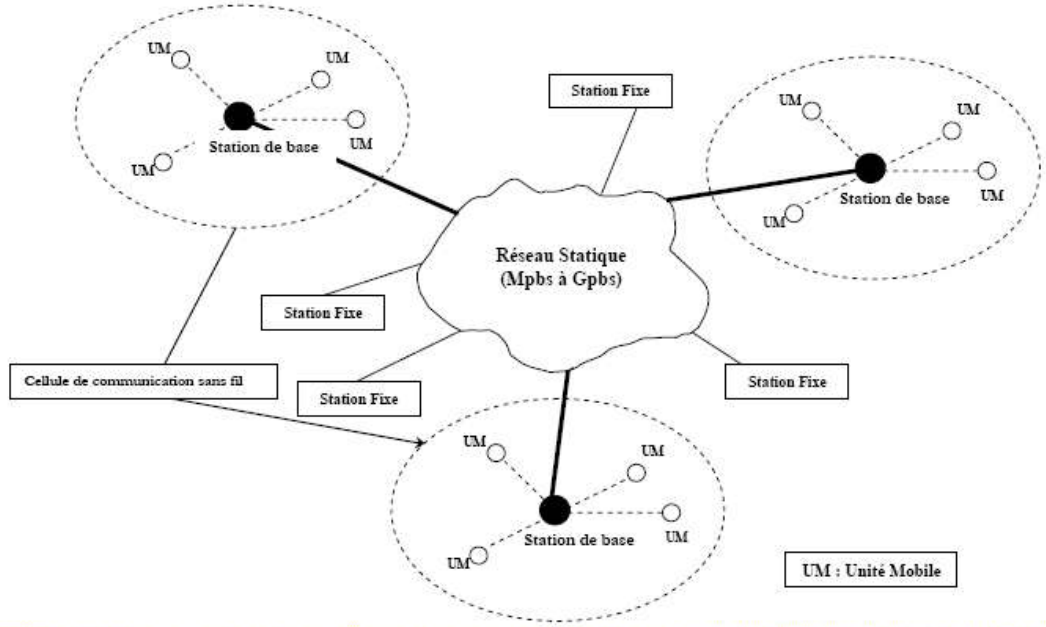


Figure 4.2 : le modèle des réseaux mobiles avec infrastructure.

Les terminaux portables étant par nature mobiles, ceux-ci peuvent changer de cellule. Lorsqu'un terminal portable a établi une communication, différents protocoles de changement de cellule (**Handoff Protocols**) permettent de changer de cellule tout en conservant cette connexion. Ces protocoles assurent automatiquement le transfert de prise en charge d'une station de base à une autre. Bien que des recouvrements soient possibles entre cellules contiguës

3.2 Les réseaux sans infrastructure

A l'inverse des réseaux sans fil avec infrastructure, les réseaux sans fil ad hoc ne nécessitent pas d'infrastructure de communication. La figure 4.3 présente l'architecture générale d'un réseau sans fil ad hoc ou MANET (Mobile Ad hoc NETworking). Elle est constituée d'un ensemble autonome de nœuds. Chaque nœud est muni d'un moyen de communication sans fil et est capable de router les paquets lui arrivant. À un instant donné, en fonction de la position des nœuds, de la configuration de leur émetteur-récepteur (niveau de puissance de transmission) et des interférences, il y a une connectivité sans fil qui existe entre les nœuds, sous forme de graphe multi-sauts.

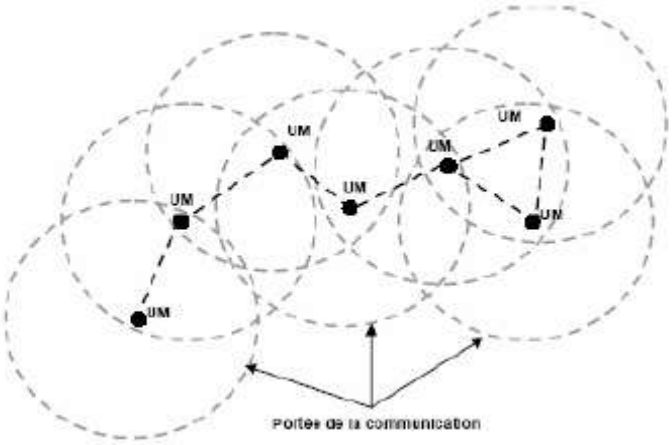


Figure 4.3 : Le modèle des réseaux mobiles sans infrastructure

Comme nous allons considérer l'existence simultanée de plusieurs réseaux ad hoc, par analogie avec les réseaux sans fil à point d'accès, nous allons appeler cellule chaque réseau ad hoc. Ce terme s'applique bien ici puisqu'il correspond également à la zone géographique délimitée par la portée de communication.

La communication entre nœuds d'une même cellule s'effectue en utilisant des protocoles de routage par voisinage. Différents protocoles (proactifs, réactifs, etc.) font l'objet de recherches en ce sens. Une cellule peut être isolée et donc n'avoir aucune possibilité de communication avec l'extérieur, mais elle peut aussi avoir des interfaces la reliant à un réseau fixe.

Dans un réseau sans fil à point d'accès, les cellules sont figées et sont attachées à la zone géographique de la station de base. Dans un réseau sans fil ad hoc, la topologie et l'emplacement des cellules peuvent changer avec le temps en fonction du mouvement des nœuds ou de l'ajustement de leurs paramètres d'émission-réception.

La mobilité des nœuds entraîne donc plusieurs cas possibles : (i) aucune incidence : juste une mise à jour des tables de routage, (ii) la scission de cellules : la cellule se divise en deux, ou (iii) la fusion de cellules. Ces différents changements sont assurés par les protocoles de routage, chaque nœud connaissant à tout moment ses nœuds voisins.

Le tableau 4.2 suivant, résume les caractéristiques des architectures des réseaux sans fil.

	Réseau sans fil à point d'accès	**Réseau sans fil ad hoc**
Cellule		
✓ **Portée de communication**	Fixe	Selon les nœuds présents dans la cellule
✓ **Zone géographique**	Fixe	Selon le déplacement des nœuds dans la cellule
Communication		
✓ **A l'intérieur de la cellule**	En passant par la station de base	Par voisinage
✓ **Vers l'extérieur de la cellule**	Toujours (en passant par la station de base)	Possible si un des nœuds de la cellule est connecté à l'extérieur

Tableau 4.2 : caractéristiques des architectures des réseaux sans fil

4. Comparaison avec les réseaux filaires

Les réseaux filaires sont très utilisés pour permettre des communications à haut débit notamment dans des réseaux locaux ou grappes de stations. Ces réseaux offrent une bande passante bien supérieure à celle que l'on peut obtenir dans les réseaux sans fil: ATM (Asynchronous Transfert Mode) (622 Mb/s), Gigabit Ethernet (IEEE 802.3z) (1 Gb/s), SCI (Scalable Coherent Interface) (1.33 Gb/s), Myrinet (_2 Gb/s), SONET (Synchronous Optical NETwork) [SONET] (2.5 Gb/s), standard 10-Gigabit Ethernet (IEEE 802.3ae).

Par rapport aux réseaux filaires, les réseaux sans fil souffrent également d'importantes variations de la bande passante. Ces variations proviennent de plusieurs phénomènes, comme l'atténuation ou le bruit des signaux. L'atténuation du signal survient lorsque le signal doit passer à travers un objet, comme un mur, ou lorsqu'il doit parcourir une distance plus importante que prévue, par exemple, en se réfléchissant sur un mur. Le bruit peut être causé par des interférences avec le signal lui-même, par exemple lorsqu'il se réfléchit plusieurs fois, ou avec des signaux extérieurs générés, par exemple, par un poste radio ou une antenne de télévision. Ces interférences provoquent des pertes de données qui doivent ensuite être retransmises, diminuant ainsi la bande passante.

Type de réseau	technologie	Portée de transmission	Bande passante	Mobilité supportée	Topologie possible
Réseau infrarouge ✓ Direct	Ondes infrarouges directes	2 m	4 Mbits	Statique	Ad hoc
✓ diffus	Ondes infrarouges diffuses	10 m	4 Mbits	Statique-piéton	Point d'accès
Réseau radio courte portée	Ondes radio	10 m	1 Mbits	Statique-piéton	Ad hoc
Réseau local sans fil	Ondes radio	150 m	54 Mbits	Piéton	Ad hoc et point d'accès
Réseau de téléphonie	Ondes radio	50 Km	2 Mbits	Véhicule	Point d'accès
Réseau satellitaire	Ondes radio	3000 Km	2 Mbits (émission) 64 Mbits (réception)	véhicule	Point d'accès

Tableau 4.3 : Caractéristiques des technologies des réseaux sans fil

Le cas extrême d'une variation est la **déconnexion.** Trois types de déconnexions peuvent se produire :

1. **Les déconnexions de courte durée** : se produisent lors des interférences ou lors d'un changement de cellule. La mobilité du terminal permet en général de rétablir rapidement la connexion.

2. **Les déconnexions de durée indéterminée** : se produisent lorsqu'un terminal portable sort en dehors de toutes zones de couverture. Dans ce cas, la reconnexion ne surviendra que lorsque le terminal rentrera dans une zone de couverture.

3. **Les déconnexions volontaires** : l'utilisateur peut vouloir économiser sa batterie et éteindre son portable. La reconnexion ne se produira alors que selon le bon vouloir de l'utilisateur.

4. **Les déconnexions permanentes :** Le terminal mobile subit une défaillance permanente.

5. Bilan et conclusion

Dans ce chapitre, nous avons présenté l'environnement mobile, en mettant l'accent sur les caractéristiques des terminaux portables et sur la différence entre ces derniers et les stations fixes. Nous avons ensuite présenté le modèle des environnements mobiles à savoir les réseaux sans fil avec infrastructure et sans infrastructure, et parlé brièvement de la technologie des réseaux sans fil. Une comparaison a été faite avec les réseaux fixes. Nous sommes arrivés à la conclusion suivante : exécuter une application conçue pour un environnement fixe dans un environnement mobile peut s'avérer délicat, et il est nécessaire que les applications puissent s'adapter à leur nouveau contexte.

Dans notre étude, nous nous sommes intéressés particulièrement aux réseaux mobiles ad hoc, que nous détaillerons dans le chapitre qui suit.

1. Introduction

Après avoir présenté les environnements mobiles en général dans le chapitre précédent, nous allons nous intéresser particulièrement dans ce chapitre, aux réseaux mobiles ad hoc, puisque c'est notre contexte de travail, pour lequel nous allons proposer un algorithme de checkpointing que nous détaillerons dans le chapitre 7.

Contrairement aux réseaux mobiles avec infrastructure, les réseaux ad hoc sont une collection d'entités mobiles interconnectées par une technologie sans fil formant un réseau temporaire sans l'aide de toute administration ou de tout support fixe. Nous allons dans les sections qui suivent, examiner de près ces réseaux afin de montrer les caractéristiques spécifiques de cet environnement, et la difficulté à y exécuter les applications des environnements fixes.

2. Le concept des réseaux Mobiles Ad hoc

De nos jours, les gens bougent de plus en plus et désirent pouvoir échanger de l'information quel que soit l'endroit où ils se trouvent, la distance qui les sépare et la vitesse à laquelle ils se déplacent. Pourtant, on ne peut envisager de brancher un câble à chaque fois pour des raisons de commodité. Cela sous-entend des réseaux spontanés, des réseaux dépourvus d'infrastructure, où chaque entité est mobile, sans aucun point d'accès centralisé et communique uniquement par radio.

Le concept des réseaux ad hoc mobiles tente d'étendre les notions de la mobilité à toutes les composantes de l'environnement. Ici, contrairement aux réseaux basés sur la communication cellulaire, aucune administration centralisée n'est disponible, ce sont les nœuds mobiles eux-mêmes qui forment, d'une manière ad hoc, une infrastructure du réseau. Aucune supposition ou limitation n'est faite sur la taille du réseau ad hoc, le réseau peut contenir des centaines ou des milliers d'unités mobiles.

Ces réseaux s'organisent automatiquement de façon à être déployables rapidement et qui doivent pouvoir s'adapter aux conditions de propagation (canal variable, interférences, etc. …), aux trafics et aux différents mouvements pouvant intervenir au sein des nœuds mobiles [60].

Les réseaux mobiles présentent une architecture originale. En effet, l'atténuation des signaux avec la distance fait que le médium peut être réutilisé simultanément en plusieurs endroits différents sans pour autant provoquer de collisions. On appelle ce phénomène la réutilisation spatiale, il sert de base aux concepts de réseau cellulaire.

La contrepartie de la réutilisation spatiale est que certaines paires de stations peuvent être hors de portée mutuelle, ce qui nécessite l'emploi d'un routage interne par des stations intermédiaires.

3. Bref historique des réseaux sans fil

Le nom des « Réseaux mobiles ad hoc » (Mobile Packet Radio Networking) est l'appellation couramment donnée à une configuration de réseaux sans fil dont le concept remonte à plus d'une trentaine d'années. Nous donnons ici rapidement un bref historique. Les recherches sur les réseaux sans fil financées principalement par le gouvernement américain, ont débuté sous la supervision de la DARPA (Defense Advanced Research Project Agency), l'ONR (the Office of Naval Research), et l'armée américaine.

Le premier projet portait le nom de '' Packet Radio Program'' ; il a débuté en 1972, et avait pour objectif d'explorer l'utilisation des réseaux radios faciles à déployer pour l'échange de données tactiques. Après onze ans, en 1983, un deuxième projet, a vu le jour sous le nom de ''SURAN'' (SURvivable Adaptative Networks). Il avait pour but, d'étendre les algorithmes développés pour son prédécesseur dans des réseaux avec des centaines d'utilisateurs. En parallèle, la DARPA, a lancé aussi, les programmes LRP (Low-cost Packet Radio) et le SCN (Survivable Communication Networks). Aujourd'hui, il y a plusieurs projets qui portent sur ce domaine, notamment, les projets TI (Tactical Internet) et NTDR (the Near-Term Digital Radio) financés, par le gouvernement américain.

Aujourd'hui, les recherches continuent avec encore plus de ferveur et d'enthousiasme suscitant beaucoup d'intérêt partout dans le monde. Voir [61] et [62] pour plus d'informations sur les projets des réseaux ad hoc.

4. Utilité et applications des réseaux Mobiles Ad hoc

De très nombreux systèmes utilisent déjà ces techniques relatives à la technologie sans fil, et connaissent une très forte expansion à l'heure actuelle (notamment la radio téléphonie mobile) mais requièrent une importante infrastructure logistique et matérielle fixe.

Les applications tactiques comme les opérations de secours, militaires ou d'explorations trouvent en Ad Hoc, le réseau idéal. La technologie Ad Hoc intéresse également la recherche. En effet, des applications civiles sont apparues.

Parmi ces applications, on distingue :

- Les applications militaires.
- Les services d'urgence comme les opérations de secours des personnes (incendies, tremblement de terre, feux, inondation) et les missions d'exploration.
- Les bases de données parallèles.
- L'enseignement à distance, les systèmes de fichiers répartis.
- La simulation distribuée interactive et plus simplement les applications de calcul distribué ou Métacomputing.
- Le travail collaboratif et les communications dans des entreprises, les campus universitaires ou bâtiments: dans le cadre d'une réunion ou d'une conférence par exemple.
- Home network: partage d'applications et communications des équipements mobiles.
- Applications commerciales: pour un paiement électronique distant (taxi) ou pour l'accès mobile à l'Internet, où service de guide en fonction de la position de l'utilisateur.
- Réseaux de senseurs: pour des applications environnementales (climat, activité de la terre ''cratères des volcans, faille géologique'', suivi des mouvements des animaux, etc.) ou domestiques (contrôle des équipements à distance).
- Réseaux en mouvement: informatique embarquée et véhicules communicants.
- Réseaux Mesh: c'est une technologie émergente qui permet d'étendre la portée d'un réseau ou de le densifier.

Remarque:

De façon générale, les réseaux ad hoc sont utilisés dans toute application où le déploiement d'une infrastructure réseau filaire est trop contraignant, soit parce que difficile à mettre en place, soit parce que la durée d'installation du réseau ne justifie pas un tel investissement en terme de câblage et de configuration.

5. Modélisation d'un réseau Mobile Ad hoc

Un réseau mobile ad hoc, appelé généralement MANET (Mobile Ad hoc NETwork), consiste en une grande population, relativement dense, d'unités mobiles qui se déplacent dans un territoire quelconque et dont le seul moyen de communication est l'utilisation des interfaces sans fil, sans l'aide d'une infrastructure préexistante ou administration centralisée. Un réseau ad hoc peut être modélisé par un graphe $\mathbf{Gt=(V_t,E_t)}$ où $\mathbf{V_t}$ représente l'ensemble des nœuds (i.e. les unités ou les hôtes mobiles) du réseau et $\mathbf{E_t}$ modélise l'ensemble les connections qui existent entre ces nœuds.

Si **e = (u,v)**, cela veut dire que les nœuds **u** et **v** sont en mesure de communiquer directement à l'instant **t**. La figure 5.1 représente un réseau Ad hoc de 10 unités mobiles sous forme d'un graphe:

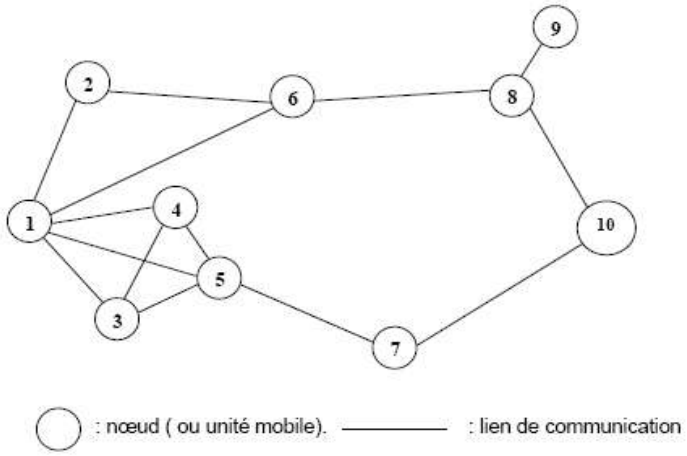


Figure 5.1 : La modélisation d'un réseau ad hoc

La topologie du réseau peut changer à tout moment, elle est donc dynamique et imprévisible ce qui fait que la déconnexion des unités soit très fréquente. Exemple :

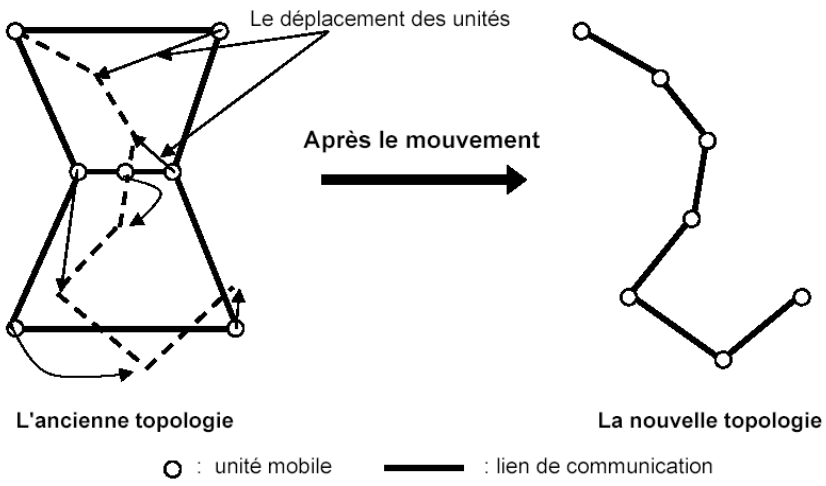


Figure 5.2 : Changement de la topologie dans les réseaux ad hoc

6. Caractéristiques et contraintes des réseaux Mobile Ad hoc

Les réseaux ad hoc présentent des avantages non négligeables, outre la mobilité qui est l'avantage principal, le prix peut également être un atout, puisqu'un peu d'électronique peut compenser un câblage manquant. Ces réseaux peuvent être facilement installés dans les endroits difficiles à câbler, ce qui élimine une bonne part du travail et du coût généralement liés à l'installation et réduit d'autant le temps nécessaire à la mise en route. De plus, les périphériques peuvent être déplacés à tout moment lorsque l'utilisateur décide de refaire ou d'étendre son installation.

Cependant, les réseaux ad hoc imposent de nombreuses contraintes, en voici quelques unes [60]:

- ✓ **Les interférences** : elles peuvent être liées à des transmissions simultanées ou à des stations cachées. D'autres interférences sont provoquées par des équipements, comme le cas des fours micro-ondes avec le WiFi, qui émettent dans la même bande de fréquences (2,4 GHz).
- ✓ **L'atténuation radio**: comme tout réseau sans fil la portée est limitée et les obstacles rencontrés atténuent fortement le signal.
- ✓ **Une bande passante limitée:** Une des caractéristiques primordiales des réseaux basés sur la communication sans fil est l'utilisation d'un médium de communication partagé. Ce partage fait que la bande passante réservée à un hôte est modeste.
- ✓ **Le débit**: les contraintes physiques et la faible bande passante rendent ces réseaux moins performants que les réseaux filaires.

- ✓ **La mobilité**: c'est l'atout principal de ce type de réseau mais ceci constitue, aussi, une contrainte majeure que le protocole de routage utilisé doit gérer. Les unités mobiles du réseau, se déplacent d'une façon libre et arbitraire. Par conséquent, la topologie du réseau peut changer, à des instants imprévisibles, d'une manière rapide et aléatoire.
- ✓ **La consommation** (contrainte d'énergie): les équipements sans fil sont destinés à être portables et donc à utiliser des batteries. Comme la durée de vie des batteries est assez courte, vu les consommations des dispositifs disponibles actuellement, les réseaux sans fil doivent alors être capables de gérer leur consommation énergétique.
- ✓ **L'absence d'infrastructure**: Les réseaux ad hoc se distinguent des autres réseaux mobiles par la propriété d'absence d'infrastructures préexistante et de tout genre d'administration centralisée. Les hôtes mobiles sont responsables d'établir et de maintenir la connectivité du réseau d'une manière continue.
- ✓ **La sécurité** : Les réseaux mobiles ad hoc sont plus touchés par le paramètre de sécurité, que les réseaux filaires classiques. Un réseau sans fil est potentiellement utilisable par toute personne se trouvant dans la zone de couverture radio des stations émettrices. Sans mesure de sécurité, il est assez facile d'écouter le trafic, de rejouer les transmissions, de manipuler les en-têtes des paquets ou de saturer le signal. Donc une protection pour réduire les risques d'attaque, est souhaitable.

7. Bilan et conclusion

Ce chapitre a été axé sur le concept des réseaux ad hoc et ses caractéristiques. Les environnements mobiles sont caractérisés par de fréquentes déconnexions et des restrictions sur les ressources utilisées, surtout si tous les usagers du système sont mobiles ce qui est le cas pour les réseaux ad hoc. Ces limitations transforment certains problèmes, ayant des solutions évidentes dans l'environnement classique, en des problèmes complexes et difficiles à résoudre.

Nous allons présenter dans le chapitre qui suit, quelques algorithmes de checkpointing dans les environnements mobiles.

1. Introduction

Nous avons évoqués dans les deux chapitres précédents, les caractéristiques des réseaux mobiles et leur différence par rapport aux réseaux fixes. Un site mobile est sujet à des déconnexions et aux déplacements, son support de stockage n'est pas considéré comme stable, il possède une faible capacité de stockage et d'énergie, ainsi qu'une puissance de calcul et largeur de bande passante limitées, et enfin une grande vulnérabilité aux erreurs.

Les algorithmes de Checkpointing conçus pour les environnements mobiles devraient considérer ces facteurs de distinction dans leurs définitions. Autrement, soit ils encourraient des coûts très élevés, soit ils ne fonctionneront pas correctement ou simplement ils ne fonctionneront pas du tout.

Ceci nous amène à dire que la sûreté de fonctionnement dans ces nouveaux environnements est plus que nécessaire ; c'est un besoin afin de pouvoir Profiter de la flexibilité et la de la liberté des contraintes temporelles et spatiales, que nous offre la technologie sans fil.

Nous verrons dans ce chapitre, une nouvelle classification des fautes en environnement mobile, puis nous parlerons de l'implémentation des algorithmes de reprise en prenant en compte de nouveaux aspects liés aux caractéristiques des réseaux mobiles. Une discussion est faite afin de démontrer que les algorithmes de checkpointing traditionnels ne sont plus adéquats en environnement mobile, et que de nouvelles solutions doivent être conçues pour s'adapter aux nouvelles spécifications du système sans fil. Et enfin, nous faisons une étude de quelques propositions d'algorithmes.

2. Classification des fautes en environnement mobile.

La prise en compte de la mobilité dans les systèmes répartis, a rendu l'implantation de la sûreté de fonctionnement plus complexe, et les caractéristiques des réseaux mobiles ont donné naissance à de nouvelles fautes, pour ne pas dire que parfois, ces caractéristiques sont elles mêmes des fautes. Parmi ces fautes, on trouve :

2.1 Les fautes du logiciel

Ce sont le résultat des problèmes dans le logiciel fonctionnant sur terminaux mobiles. Dans ce cas, l'utilisateur doit recommencer toutes les opérations.

2.2 Les fautes de l'environnement

Ce sont le résultat de l'interaction entre l'environnement et les terminaux mobiles. Dans le cas de réseaux sans fils, l'interaction entre l'environnement et les signaux du réseau sans fil est très importante. Par exemple, des fautes comme les pertes de paquet peuvent survenir.

2.3 Les Contraintes architecturales

A cause des conditions de mobilité, les appareils mobiles sont petits et légers et ont une faible capacité de ressources en termes de batterie, de mémoire et de stockage.

Le tableau ci-dessous présente la classification des fautes.

Faute	Nature de la faute	Présence de la faute
Réduction de la largeur de bande passante	Environnementale	Transitoire
Capacité mémoire	Architecturale	Permanente
Problèmes de batterie	Architecturale	Permanente
Perte de paquet	Environnementale	Transitoire
Hors de portée	Environnementale	Transitoire
Arrêt du système	Logicielle	Transitoire
Panne de transaction	Logicielle	Transitoire

Tableau 6.1 : Classification des fautes en environnement mobile.

Le manque de ressource représente lui-même une faute, parce qu'il peut perturber l'exécution d'une application. Contrairement aux fautes de logiciel et fautes environnementales, les fautes architecturales sont toujours présentes et de natures différentes.

Il est à prévoir que les contraintes architecturales seront améliorées en fonction de l'évolution technologique. Des recherches actives dans ce domaine envisagent l'extension

de la durée de vie de la batterie, la puissance des CPU et aussi l'accroissement de la taille de la mémoire.

3. Implémentation des algorithmes de reprise dans les réseaux mobiles

En environnement mobile, exécuter une application conçue pour un environnement fixe peut s'avérer délicat. Comme nous l'avons déjà mentionné auparavant, d'importantes différences existent entre ces deux types d'environnements et elles ne sont pas, en général, prises en compte par les applications. En effet, la conception d'un algorithme de recouvrement arrière se base la plupart du temps sur un environnement connu, statique et suffisant pour les besoins de l'application. Cela n'est plus vrai dans un environnement mobile où des terminaux portables peuvent arriver et repartir, disposer de ressources faibles à un instant donné, puis en avoir suffisamment l'instant d'après. Afin de prendre en compte ces changements possibles, il est nécessaire que ces algorithmes puissent s'y adapter. Trois axes principaux d'adaptations sont définis ci-dessous :

3.1 Adaptation de l'algorithme au terminal portable

Les contraintes qu'imposent les unités mobiles, sont les nouvelles considérations à prendre en compte dans la conception des algorithmes de reprise.

- ✓ **Ressources restreintes :** la restriction des ressources due aux contraintes de minimisation de poids et de volume sur un terminal portable peut empêcher l'exécution de l'algorithme sous des conditions optimales. L'algorithme doit changer son comportement pour utiliser moins de ressources ou des ressources différentes. Par exemple, lorsque le terminal portable n'a plus assez de mémoire pour charger ses données, l'algorithme peut décider de stocker ses données sur le disque dur, ou bien de ne charger que les données les plus utilisées, ou même utiliser les ressources d'un autre terminal portable.

- ✓ **Autonomie limitée :** la durée de vie de la batterie est un paramètre limitant incontournable. Un algorithme de reprise doit minimiser l'effort du terminal portable, en le déchargeant par exemple de la tâche de coordination, et en évitant de le déranger inutilement, car il peut être en mode veille pour économiser l'énergie.

- ✓ **Vulnérabilité aux défaillances :** les unités mobiles sont exposées aux pertes, vols, pannes physiques…et donc leur disque dur ne peut être considéré comme support

stable de stockage ni fiable ni encore moins stable. Une solution raisonnable est d'utiliser un stockage stable dans les stations de bases [63], pour sauvegarder les points de reprises des unités mobiles.

3.2 Adaptation de l'algorithme au réseau sans fil

- ✓ **Bande passante limitée :** la contrainte de la bande passante limitée est imposée par le réseau lui-même, en effet les variations de la bande passante peuvent introduire des délais inacceptables dans les algorithmes de checkpointing qui engendrent un grand trafic sur le réseau sans fil. Il serait plus judicieux de choisir des algorithmes générant peu de trafic, en terme d'échange de messages de coordinations, de sauvegarde sur le support stable ou encore l'utilisation de messages énormes.
- ✓ **Déconnexions réseau :** les systèmes mobiles sont caractérisés par les déconnexions fréquentes qui peuvent parfois totalement paralyser l'algorithme de checkpointing si des mesures appropriées ne sont pas prises en compte. L'anticipation des déconnexions en utilisant des techniques de pré-chargement pourrait permettre à l'utilisateur de travailler en mode déconnecté. Un terminal mobile peut prévenir sa station de base courante avant de se déconnecter et pré-charger la sauvegarde de son état, pour que s'il est invoqué lors de sa déconnexion, par la procédure de checkpointing, la station de base répond avec l'état sauvegardé à son niveau.

3.3 Adaptation de l'algorithme à l'environnement

- ✓ **Localisation géographique :** la mobilité des terminaux portables permet une utilisation depuis n'importe quel endroit. Le changement de localisation appelé Handoff, complique le routage des messages. Le message envoyé d'un nœud à un autre peut être routé à nouveau, si le nœud destinataire a changé de cellule, ce qui nécessite dans la majorité du temps une phase de recherche qui peut être très coûteuse en temps. Pour que la mobilité soit transparente, deux solutions ont été proposées dans les algorithmes de checkpointing déjà proposés, pour gérer les handoffs :

  a) **L'enregistrement des déplacements d'une unité mobile** : cela consiste en la sauvegarde de l'itinéraire emprunté [64] par n'importe quelle unité mobile, en associant un graphe des stations de bases visitées par ce nœud.

  b) **Notification du site lui-même de son déplacement** : l'unité mobile, elle-même doit informer le **système** de sa position. Par exemple l'algorithme de checkpointing dans [65] est doté de deux routines, la première, :**Leave-Message** : lorsque l'unité

mobile décide de quitter une station de base MSS_p vers une autre MSS_q, il envoie à MSS_p un message l'avisant qu'il va la quitter. Ce message contient l'identité de cette unité mobile. La deuxième routine, **Join_Message** : de la même façon que lorsqu'il veut quitter une MSS, l'unité mobile envoie un message lorsqu'il veut rejoindre la nouvelle MSSq. Ce message encapsule l'identité de l'ancienne MSS.

4. Conception de nouveaux algorithmes de checkpointing

Les mécanismes de Checkpointing dans les réseaux fixes ont été largement étudiés, et plusieurs algorithmes de Checkpointing ont été proposés. En plus de la difficulté de la conception de ces algorithmes, déjà évoquée, qui vise à éviter certains phénomènes (effet domino, perte de message…), un deuxième constat s'impose également. Il est évident que les algorithmes de Checkpointing traditionnels conçus pour les environnements fixes, ne sont plus adaptés pour les environnements mobiles. Ils doivent adopter de nouvelles spécifications et solutions adaptées à la mobilité du système. On peut classifier ces algorithmes en deux catégories principales : coordonnées et non coordonnées.

- Les algorithmes coordonnés peuvent assurer un état global cohérent, et nécessitent de maintenir la taille de deux états pour chaque site. Ils requièrent la coordination pour la sauvegarde des points de reprise locaux. Par conséquent, des messages de contrôle doivent être envoyés aux sites mobiles pendant l'exécution de l'application ; ce qui induit un coût de recherche important pour localiser l'emplacement courant d'un site mobile via sa station de base (MSS). Cette localisation permet de router les messages de contrôle vers les stations de base appropriées. Ce problème est compliqué d'avantage si le site mobile change de cellule avant que l'algorithme de Checkpointing ne soit terminé ou s'il se déconnecte du réseau pendant un temps inacceptable.

- Les algorithmes induits par la communication et les algorithmes non coordonnés, donnent plus d'autonomie aux sites, puisqu'ils sauvegardent leurs points de reprise locaux indépendamment sans synchronisation. En revanche, chaque site mobile doit sauvegarder plusieurs points de reprises durant l'exécution de l'application, et les transférer dans un support de stockage stable de leur MSS.(à cause de la faible capacité et l'instabilité du support local); le coût de ce genre d'algorithme est trop élevé. Les protocoles non coordonnés ont en plus le désavantage du coût de recouvrement qui est très élevé car cela nécessite un grand échange de message pour sélectionner (après l'échec) un point de reprise local pour chaque site afin de former ensemble un état global cohérent.

On conclue donc que les algorithmes de Checkpointing coordonnés et non coordonnés traditionnels sont inadéquats pour un environnement mobile.

A partir de là, de nouvelles techniques sont nécessaires pour réaliser un algorithme de checkpointing adapté à l'environnement mobile. Plusieurs approches existent pour obtenir le maximum des avantages des différentes classes. Il n'existe pas une véritable classification des techniques de checkpointing mobile. Dans les algorithmes proposés, il y a ceux qui sont hybrides et combinent les approches coordonnées et les approches non coordonnées, afin de tirer profit de leurs avantages, en évitant ou du moins en minimisant leurs inconvénients. Et il y a ceux qui utilisent de nouvelles notions telles que le mutable checkpointing et le successive checkpointing. Dans la section qui suit, nous verrons quelques propositions.

5. Etude de quelques propositions d'algorithmes

Beaucoup d'algorithmes ont été proposés, depuis 1993 à ce jour, nous allons présenter quelques uns en se basant sur les deux facteurs suivants : le premier réside dans l'originalité des idées apportées, car ces algorithmes ont servi de points de départ pour de nouvelles propositions (certaines propositions n'ont fait que formuler les anciennes). Le deuxième concerne la primeur des propositions émises faisant ainsi de leurs auteurs les pionniers du contexte mobile.

Dans la plupart des propositions, le modèle conventionnel utilisé est celui du réseau mobile avec infrastructure, dont l'architecture se compose de deux ensembles d'entités distinctes : les sites mobiles et les stations de base. L'ensemble des stations de base avec les liens de communication forment un réseau fixe, tandis que chaque station de base et les sites mobiles appartenant à sa cellule forment un réseau sans fil.

On présente dans ce qui suit les hypothèses et des définitions communes admises dans toutes les propositions :

- Le réseau fixe garantit une livraison fiable et ordonnancée (FIFO) des messages entre n'importe quel couple de stations de bases.
- Le délai de transfert d'un message entre deux sites fixes est arbitraire mais fini.
- Une adresse fixe (identificateur) est associée à un site mobile, cet identificateur ne dépend pas de l'emplacement de ce site dans le réseau.

- Un site mobile peut directement communiquer avec sa station de base (et vice versa) seulement si ce site est physiquement situé dans la cellule entretenue par cette station de base, c'est-à-dire que le site mobile est local à cette station de base.

5.1 Les propositions

1. Recovery in Distributed Mobile Environment. (1993)

Auteurs : **P. Krishna, N. H. Vaidya et D. K. Pradhan**

Krishna, Vaidya et Pradhan ont été les premiers à introduire l'idée de la remise en cause de l'adaptabilité des algorithmes de checkpointing traditionnels aux caractéristiques spécifiques de l'environnement mobile, c'était en 1993.

L'idée introduite dans [64], est qu'il fallait aussi sauvegarder l'itinéraire des sites mobiles, pour minimiser le temps de recherche et de routage des messages vers ces sites. Cet algorithme utilise les stratégies traditionnelles proposées pour les environnements fixes pour sauvegarder l'état d'un site à savoir : la stratégie coordonnée et la stratégie non cordonnée. Il est doté aussi de deux approches pour la sauvegarde des déplacements d'un site mobile entre les stations de base, qui sont :

1. L'approche Pessimiste : chaque fois qu'un site se déplace, son point de reprise sauvegardé dans la MSS courante, est transféré vers la nouvelle MSS.

2. L'approche Lazy : A chaque site mobile est associée une liste contenant l'historique des MSSs visitées par ce dernier. Cette liste est enregistrée au niveau de la MSS courante, et transférée et modifiée chaque fois que le site se connecte à une nouvelle MSS.

Discussion : Cet algorithme travaille en combinant deux algorithmes, un pour la sauvegarde des points de reprises et l'autre pour la sauvegarde des déplacements, sans proposer au préalable un moyen de les coordonner. Cela pourrait s'avérer très coûteux en messages si on choisissait par exemple la méthode non coordonnée pour le checkpointing et l'approche pessimiste. Ce qui impliquerait le transfert de plusieurs points de reprise entre les stations de base, chaque fois que le site mobile se déplace.

2. Checkpointing Distributed Applications on Mobile Computer. (1994)

Auteurs : **Arup Acharya et B. R. Badrinath**

En 1994, Acharya et Badrinath, ont été les premiers à proposer un algorithme de checkpointing prenant en considération les caractéristiques imposées par la mobilité et ayant pour but la réduction du coût de recherche des sites impliqués.

Cet algorithme [63] se compose de deux parties : la première partie exécutée par chaque site mobile pour sauvegarder de manière asynchrone son point de reprise, en se basant sur le protocole à deux phase (avec des modifications appropriées) ; et la partie exécutée par la station de base, pour le compte d'un site mobile, afin de déterminer une ligne de recouvrement cohérente.

Discussion : Les auteurs présentent dans leur article un algorithme de Checkpointing utilisant le support de stockage stable de la station de base pour sauvegarder les points de reprises locaux des sites mobiles. Cependant, l'inconvénient majeur de cet algorithme est le grand nombre de point de reprise qu'il faille sauvegarder. Ainsi, un site mobile doit sauvegarder son point de reprise chaque fois qu'il se déplace vers une nouvelle cellule avant de se déconnecter du réseau. Le coût de transfert/sauvegarde de ces points de reprise devient alors très élevé.

3. Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems. (1996)

Auteurs : **Ravi Prakash et Mukesh Singhal**

Prakash et Singhal ont proposé un algorithme [65] de Checkpointing coordonné, qui ne force pas chaque nœud à prendre un point de reprise, ni bloque l'application pendant son exécution. Si un site lance l'algorithme, seulement les sites qui ont directement ou indirectement communiqué avec le site initiateur, depuis leur dernier point de reprise, doivent prendre des points de reprises.

Ils proposent également un algorithme minimal de recouvrement arrière dans lequel le calcul d'un site est repris seulement si ce nœud dépend des opérations qui ont été la cause de l'échec. Cet algorithme a un coût de communication et de sauvegarde bas et réduit la consommation d'énergie. Ainsi, il réduit les contraintes de la largeur de la bande passante des réseaux mobiles.

Un des aspects intéressant de l'algorithme est qu'il est réalisé en deux phases : la phase Lazy et la phase aggressive. La première est quasi coordonnée, dans laquelle tous les sites qui sont dépendants de l'initiateur prennent un point de reprise provisoire. La deuxième est coordonnée, de sorte que les sites transforment ces points de reprise provisoires en des

points de reprises permanents. Ainsi l'algorithme est basé aussi sur la relation d'antériorité (happen before) de Lamport, pour détecter les dépendances causales entre les sites.

Discussion : Puisque le recouvrement, concerne seulement les sites participant à l'opération qui a causé l'échec, cet algorithme a un coût de communication et de sauvegarde bas et réduit la consommation d'énergie. Ainsi, il réduit les contraintes de la largeur de la bande passante des réseaux mobiles. Cependant, il ne prend pas en considération les contraintes de l'instabilité du support de stockage d'un site mobile et le faible espace de stockage de ces supports. L'algorithme est complexe et utilise beaucoup de structures de données (4 vecteurs et 5 variables) qu'il faut maintenir dans chaque site.

Mais le problème et l'inconvénient majeur est que les mêmes auteurs, prouvent après deux ans (en 1998) dans [66] que cet algorithme résulte des états d'incohérence dans quelques situations. Ils prouvent ensuite dans [69] qu'il n'existe pas d'algorithme de Checkpointing qui est à la fois non bloquant et force un nombre minimum de sites pour prendre un point de reprise.

4. Adaptive Recovery for Mobile Environnement. (1997)

Auteurs : Nuno Neves et W. Kent Fuchs

Neves et Kent Fuchs proposent dans leur article [67] un protocole de Checkpointing basé sur le temps physique. L'algorithme utilise une coordination indirecte pour sauvegarder un point de reprise globale cohérent. Les sites sauvegardent périodiquement leurs états, chaque fois qu'un temporisateur local expire.

Les auteurs de cet article, présentent une architecture un peu différente pour un réseau mobile. Chaque site mobile est associé à une station de base originale HS (Home Station). Il peut ainsi envoyer et recevoir les paquets comme un site fixe tant qu'il est dans la cellule de cette station de base. Quand il se déplace vers une autre une nouvelle MSS, sa station de base originale (HS) le représente, elle reçoit les messages qui lui sont destinés et les expédie vers la station étrangère (FH) à laquelle il est actuellement rattaché. Cependant, le site mobile informe sa station de base originale HS de chaque déplacement. Il utilise la station de base HS aussi pour sauvegarder ses points de reprise.

L'algorithme crée deux types de points de reprise : les points de reprise Soft, sauvegardés localement et utilisés pour tolérer les échecs Soft (Soft Checkpoints), et les points de reprise Hard sauvegardés dans le support de stockage stable et utilisés pour tolérer les échecs Hard (Hard Checkpoints). Il adapte le comportement des points de reprises à la qualité de service du réseau, donc pour un réseau à une QOS minime, alors l'algorithme prend beaucoup de points soft avant de les valider en un seul point hard.

Discussion : cet algorithme peut sauvegarder un état global cohérent et récupérable sans devoir échanger des messages. Il est donc bien adapté aux caractéristiques des environnements mobiles. Il diminue grandement la consommation d'énergie. Néanmoins, son inconvénient majeur réside dans la déviation entre les horloges qui pourrait être grande dans certaines situations. Ainsi, si par exemple, la communication entre deux parties des sites mobiles est absente pendant un temps assez long, et puisque la synchronisation entre les horloges physiques se base sur les messages de l'application, alors les horloges de ces deux parties peuvent avoir une grande déviation. Ainsi, si une opération de recouvrement, pendant cet intervalle d'absence de communication est exigée, alors le dernier état global peut ne pas être cohérent.

5. Low-Cost Checkpointing with Mutable Checkpoints in Mobile Computing Systems. (1998)

Auteurs : **Guohong Cao et Mukesh Singhal**

Dans l'article [68], Cao et Singhal présentent un nouveau concept de Checkpointing par la proposition d'un nouveau type de points de reprise : ce sont les points de reprise Mutables, qui sont ni des points de reprise provisoires ni permanents. De plus, on peut les sauvegarder n' importe où, même sur le support de stockage d'un site mobile. De cette manière, on peut éviter le coût élevé de transfert des points de reprise inutiles sur les supports de stockages stables dans les stations de bases.

Le but de cet algorithme est de concevoir une approche coordonnée non bloquante, minimisant le nombre de checkpoints sauvegardés, minimisant le nombre de sites impliqués dans la procédure de checkpointing et réduisant le coût de synchronisation.

Lorsqu'un site mobile envoie un message, il y embarque la valeur courante de son numéro de point de reprise csn (checkpoint sequence number). Quand le site de destination reçoit ce message, soit il le traite si son numéro de point de reprise est supérieur ou égal au csn reçu. Sinon le site récepteur prend un point de reprise Mutable, et le sauvegarde localement avant de traiter le message. Il faut noter qu'avec ce nouveau concept de Mutable checkpoint, le site récepteur n'envoie pas de requête de Checkpointing aux autres processus qui dépendent de lui. Si par la suite, le site reçoit une requête de checkpointing, il transforme ce point de reprise Mutable en un point de reprise provisoire et le transfère sur stockage stable. Il envoi ensuite une requête de Checkpointing à tous les sites qui dépendent de lui pour qu'ils prennent des points de reprise provisoires. Autrement, il détruit ce point de reprise Mutable.

Discussion: Le concept de Mutable checkpoint est très intéressant pour éviter le coût élevé de transfert des points de reprise (qui peuvent être inutiles) aux supports de stockages stables dans les stations de bases. Seulement, on remarque que l'algorithme proposé est très complexe. De plus, la grande quantité de structures de données proposées qu'il faille maintenir sur chaque site (mobile ou fixe) ne prend pas en considération la faible capacité et l'instabilité du support de stockage d'un site mobile.

6. **On the Impossibility of Min-Process Non-Blocking Checkpointing And An Efficient Checkpointing Algorithm for Mobile Computing Systems.** (Aout 1998)

Auteurs : **Guohong Cao et Mukesh Singhal**

Dans cet article [69] Cao et Singhal critiquent leur algorithme proposé en 1996. Ils sont même allés jusqu'à prouver qu'il n'existe pas d'algorithme de Checkpointing non bloquant et qui force le minimum de sites à prendre des points de reprise. Cette preuve est venue après que les auteurs aient détecté dans [66] un état d'incohérence dans l'algorithme [65] proposé précédemment. Ils ont rectifié leur position par le théorème suivant :

« Il n'existe pas d'algorithme de Checkpointing qui réalise conjointement le non blocage et force le minimum de sites à prendre des points de reprise. »

L'algorithme proposé est basé sur deux phases : lors de la première phase, des points de reprise provisoires sont créés. Et lors de la deuxième phase, ces points de reprise sont transformés en des points de reprise permanents.

Chaque site est muni d'un vecteur booléen de dépendance de N éléments, Ce vecteur est toujours maintenu par la MSS courante de ce site. Ainsi, lorsqu'un site mobile initie la procédure de Checkpointing, il prend un checkpoint provisoire et envoie une requête à sa MSS qui jouera le rôle de station de base initiatrice. Cette dernière envoie une requête à toutes les stations de base pour rassembler les informations de dépendances et se bloque. A la réception des demandes, chaque MSS lui envoie les vecteurs de dépendances de tous les sites qui sont dans sa cellule et se bloque à leur tour.

A la réception de tous les vecteurs, la MSS initiatrice, calcule la matrice de dépendance ($N \times N$), qui servira à établir la liste des sites impliqués par ce checkpointing. Cette liste sera envoyée avec une requête de checkpointing à toutes les MSSs. A la réception de la requête, chaque MSS vérifie si un site qui lui est lié, appartient à cette liste reçue. Si oui, elle lui transmet la requête de Checkpointing. A la réception de cette requête, le site mobile prend un point de reprise provisoire et envoie un message d'acquittement à sa station de base. Lorsque chaque MSS reçoit des acquittements de tous les sites mobiles qui dépendent du site initiateur, elle envoie alors à son tour un message d'acquittement à la MSS du site

initiateur. Et de là, les actions de la première phase sont terminées. Pendant la deuxième phase, quand la station de base du site initiateur reçoit tous les acquittements de toutes les MSS, elle envoie alors un message de validation à toutes les stations de base. Ces dernières, à leur tour, envoient ce message à tous les sites mobiles qui dépendent du site initiateur afin de transformer leurs points de reprise provisoires en des points de reprise permanents.

Discussion : cette fois, Cao et Singhal proposent un algorithme clair et simple à implémenter, qui satisfait les contraintes des réseaux mobiles ou du moins les allège. Ils précisent ainsi de façon transparente le rôle de chaque composant des réseaux. Par contre, son inconvénient majeur, est le blocage des MSSs après l'envoi des vecteurs de dépendances jusqu'à l'arrivée de la liste des sites impliqués. Cet inconvénient peut être inacceptable surtout si le calcul effectué a des contraintes critiques.

7. **Checkpointing-Recovery for reliable Mobile Systems. (1998)**

Auteurs : **Hiroaki Higaki et Makoto Takizawa**

L'algorithme [70] de checkpointing proposé par Higaki et Takizawa, est un algorithme Hybride, combinant la stratégie coordonnée pour la sauvegarde d'un état global cohérent pour les sites fixes, et la stratégie non cordonnée pour la sauvegarde de l'état global des sites mobiles. Cette stratégie permet de respecter les caractéristiques spécifiques d'un réseau mobile où l'utilisation d'une stratégie totalement coordonnée peut ne pas être adéquate. Puisque l'état global sauvegardé pour les sites mobiles peut ne pas être cohérent avec celui qui est sauvegardé pour les sites fixes, les auteurs utilisent la technique de journalisation des messages envoyés de et vers les sites. Cette action permet aux points de reprise des sites mobiles sauvegardés séparément de rattraper l'état global construit pour les sites fixes, et ce, afin de former un état global cohérent pour tout le système.

Discussion : L'inconvénient majeur de cette proposition est la complexité élevée au niveau de la technique de journalisation proposée, qui peut croitre de manière remarquable, dans le cas ou l'application exécutée est à échange intensif de messages de calcul entre les sites.

8. **Efficient Checkpoint-Based Failure Recovery Technique in Mobile Computing Systems. (2001)**

Auteurs : Cheng-Min Lin et Chyi-Ren Dow

Dans leur article [71] Lin et Dow proposent un nouvel algorithme hybride de Checkpointing, en combinant cette fois l'approche coordonnée pour la sauvegarde des

points de reprise des sites fixes et l'approche induite par les communications, pour la sauvegarde des points de reprises des sites mobiles.

Cet algorithme est basé sur trois phases :

- La première est l'attribution de la tâche de coordination aux sites fixes pour éviter que les messages de synchronisation ne submergent le réseau mobile.
- A la réception d'une requête de checkpointing, chaque MSS déclenche un temporisateur local et attend l'expiration de ce délai. Ainsi, si un site mobile reçoit un message portant un index plus élevé que le sien, il doit établir un checkpoint forcé
- Chaque MSS dont le temporisateur a expiré, envoie une requête de checkpointing vers les sites qui n'ont reçu aucun message depuis la réception de la requête de checkpointing par cette MSS.

Discussion : Par l'utilisation de la coordination, cet algorithme non bloquant, assure un état cohérent global et un coût d'exécution minimum. Son inconvénient est la complexité grandissante lorsque l'application exécutée exige un échange important de messages de calcul.

9. Successive Checkpointing Approch for Mobile Computing Environment. (2003)

Auteurs : Pushpendra Singh et Gilbert Cabillic

Pushendra et Cabillic ont présenté dans leur article [72] un algorithme coordonné avec une nouvelle notion nommée successive checkpointing. Cette approche est basée sur la prise de checkpoints retardés et utilise le partial blocking qui consiste à bloquer un processus pendant le checkpointing si le message qu'il reçoit risque de devenir orphelin. En plus cet algorithme implique seulement un nombre minimum de sites dans la procédure de checkpointing. Un message peut devenir orphelin dans les deux cas suivants : Lorsqu'un processus envoie un message, après avoir pris un checkpoint et lorsqu'un processus reçoit un message avant de prendre un checkpoint. En utilisant ces informations, les auteurs ont développé le concept de successive checkpoint qui minimise le blocage et préserve la consistance. C'est un « extra local checkpoint » pris pour éviter les messages orphelins.

Dans l'exemple qui suit, si P1 prend un autre checkpoint local après l'envoi de m1, alors m1 ne devient plus orphelin. Ce checkpoint est appelé « **successive checkpoint** », il est pris dans la deuxième phase du checkpointing lorsqu'il reçoit un commit de l'initiateur afin de rendre son checkpoint permanent, seulement s'il a envoyé au moins un message après avoir pris un checkpoint dans la première phase.

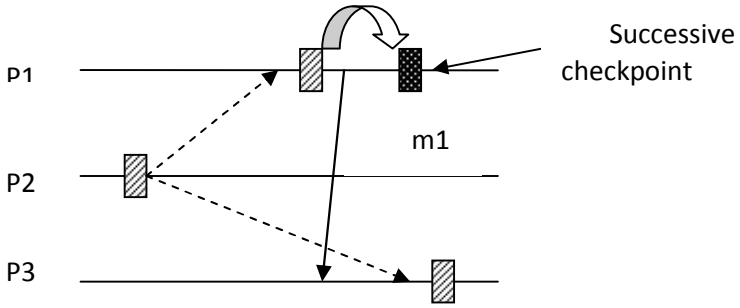


Figure 6.1 : Successive checkpoint.

Dans la deuxième phase, le processus P1 rend ce nouveau checkpoint permanent à la place du précèdent. Cette approche est différente de celle du mutable checkpoint de Cao et Singhal car ce successive checkpoint n'est jamais effacé, en conséquence, la consommation d'énergie pour prendre ce checkpoint n'est pas gaspillée.

Discussion : La notion de successive checkpointing est intéressante, puisqu'elle permet d'évitant de refaire les calculs déjà faits. Des checkpoints retardés sont pris **incluant le maximum de calcul possible,** ceci garantit un recouvrement rapide et donc une sauvegarde d'énergie pendant le recouvrement. Néanmoins, cet algorithme n'a pas pris en compte le cas de déconnexion des sites. Et si les mesures appropriées ne sont pas prises, il y a un risque de blocage indéfini des processus, et ceci peut invalider complètement l'algorithme.

10. Movement-Based Checkpointing and Logging for Recovery in Mobile Computing Systems (2006)

Auteurs : Sapna E. George, Ing-Ray et Chen Ying Jin

Les auteurs ont proposé un algorithme [73] de checkpointing basé sur le déplacement des sites mobiles. Il combine l'approche non coordonné et la journalisation optimiste des messages.

Un site mobile prend un point de reprise de manière asynchrone, à chaque fois qu'il atteint un seuil de déplacement M. (i.e qu'il a fait M handoffs). Ce seuil dépend du taux de défaillance, du taux des messages arrivés et du taux de la mobilité de l'application et du site lui-même. Lors du recouvrement, seul le site défaillant doit retourner en arrière. Les auteurs ont développé un modèle de performance basé sur les réseaux de Pétri, afin d'identifier la valeur optimale du seuil de déplacement M.

Discussion : C'est la première fois que le paramètre de la mobilité est introduit pour la sauvegarde des points de reprise. Cet algorithme est donc parfaitement adapté aux caractéristiques des réseaux mobiles. Il n'induit aucun coût de synchronisation, puisque la sauvegarde est faite de manière asynchrone. En plus il réduit la consommation d'énergie et de la bande passante, puisque lors du recouvrement, seul le site défaillant, doit retourner en arrière. Son inconvénient, réside dans la détermination de la valeur optimale du seuil de déplacement M, et dans le fait que la journalisation optimiste ne garantit pas toujours la « condition de non orphelinité ». Ceci rend l'opération de reprise compliquée, car il faut éventuellement retourner en arrière plusieurs fois avant de trouver un état cohérent. Le risque d'un retour à l'état initial du calcul (effet domino) est possible. De plus, un algorithme de Ramasse-miettes est indispensable pour la suppression des informations inutiles.

11. A Novel Roll-Back Mechanism for Performance Enhancement of Asynchronous Checkpointing and Recovery (2007)

Auteurs: Bidyut Gupta and Shahram Rahimi

Gupta et Rahimi ont proposé dans [74] un algorithme de recouvrement efficace et simple. Ils considèrent l'approche de checkpointing non coordonnée. Les sites disposent de plus de liberté et décident du moment opportun pour prendre leur point de reprise de façon asynchrone. Lors d'un échec, un état global cohérent (EGC) est calculé.

L'objectif de ce travail, est de proposer un algorithme de reprise qui réduit considérablement de nombre de comparaisons pour déterminer cet EGC. Il consiste à déterminer à priori les points de reprises qui ne seront pas inclus dans l'EGC. On peut donc exclure ces checkpoints de la comparaison. Pour cela chaque processus (site) doit maintenir les structures de données suivantes : (1) un vecteur V_i de longueur n (n nombre de processus), il enregistre le nombre de messages de calcul que Pi a envoyé aux autres processus. $V_{i,j} = V_i(j)$. i.e le nombre de messages que Pi a envoyé à Pj. (2) une liste linéaire R_i. A l'instant t, la longueur de la liste est égale au nombre de checkpoints pris par Pi. $R_i=[r_{i,1},\ldots r_{i,r},\ldots.]$ ou $r_{i,r}=R_i(r)$ représente le nombre de messages reçus par Pi de la part des autres processus depuis son $r^{ème}$ checkpoint. (3) Et un flag booléen.

Lorsque Pi tombe en panne, il devient l'initiateur, i.e responsable de l'algorithme de recouvrement. **(a)** Il diffuse une requête aux autres processus Pj pour qu'ils lui envoient leur vecteur V_j. Lorsqu'il reçoit tous les vecteurs V_j, **(b)** il calcule la matrice V_N dont chaque ligne est constituée par V_j (j=1,n). Puis calcule le vecteur $V_c=[V_{c,0},V_{c,1},\ldots.Vc,j,..V_{c,n-1}]$ ou $V_{c,i}$ est la somme de la colonne j de la matrice V_N, et

représente le nombre total des messages envoyés à Pj par tous les autres processus. **(c)** l'initiateur Pi envoie $V_{c,j}$ à tous les Pj. chaque Pj qui reçoit $V_{c,j}$, calcule $D_j=R_j(r)-V_{c,j}$. si D_j >0, alors D_j nous donne le nombre exact des messages orphelins reçus par Pj depuis son checkpoint $C_{j,r}$. dans ce cas Pj cherche dans la liste R_j jusqu'à trouver le plus grand entier m (m<r) qui satisfait $R_j(r) - R_j(m) \geq Dj$. Il met son flag à un et considère V_j qui correspond à son $C_{i,m}$ (i.e. $C_{i,r}$ sera remplacé par $C_{i,m}$ dans la prochaine itération de l'algorithme et donc $C_{i,r}$, $C_{i,r-1}$,..$C_{i,m+1}$ seront les points de reprises non considérés dans la détermination de l'EGC . Si Dj ≤ 0 alors Pj met son flag à zéro. **(d)** lorsque l'initiateur Pi reçoit tous les flags et les V_j, alors si tous les flags sont égaux à zéro, il demande à chaque Pj de retourner à son dernier point de reprise correspondant à Dj, sinon il fait une nouvelle itération et retourne à l'étape **(b)** pour recalculer V_c.

Discussion : l'algorithme de checkpointing est asynchrone, ce qui n'induit aucun coût de synchronisation et donc l'énergie et la bande passante sont conservées. L'algorithme de recouvrement offre les avantages suivants : durant chaque itération, chaque processus Pj détermine les checkpoints qui ne seront pas considérés dans l'EGC, ce qui permet d'éviter un calcul inutile de V_c correspondant à ces points de reprise exclus. En conséquence cela permet d'éviter d'inutiles comparaisons. Tout ceci nous garantit un recouvrement rapide et efficace. En revanche, le principal inconvénient de cet algorithme est le suivant : puisque l'approche utilisée est totalement asynchrone, le risque de l'effet domino n'est malheureusement pas écarté, ce qui peut invalider complètement l'algorithme. Rajouter à cela la complexité du calcul de la ligne de recouvrement, ainsi que de nombreux points de reprise doivent être maintenues par chaque processus.

6. Bilan et Conclusion

La naissance des environnements mobiles a introduit de nouveaux challenges et défis dus aux caractéristiques spécifiques des réseaux mobiles. De ce fait, on constate que les applications des réseaux fixes ne sont pas applicables dans ces nouveaux environnements, de même les approches de checkpointing traditionnelles ne sont pas adaptées. Il est donc nécessaire voire indispensable de concevoir de nouvelles techniques de checkpointing qui prennent en considération les facteurs suivant : mobilité, fréquentes déconnexions, largeur de bande et batterie limitée, et enfin vulnérabilité des unités mobiles aux défaillances.

Nous avons vu dans ce chapitre une nouvelle classification des fautes en environnement mobile, et comment y implémenter un algorithme de reprise pour qu'il puisse d'adapter aux nouvelles caractéristiques. Une étude de quelques propositions d'algorithmes a été faite, et une critique a été élaborée pour chaque proposition afin de montrer les points positifs et les points négatifs de chaque algorithme. On remarque aussi qu'il n y a pas d'algorithme avec zéro inconvénient, et que dans chaque proposition il y a un compromis à faire, l'un optimise sur la mémoire et l'énergie des sites mobiles, l'autre optimise la bande passante, et le temps de latence. Ceci dit, tout ce qui a été proposé jusqu'à maintenant ne sont que des algorithmes dédiés à l'environnement fixe qu'on essaye à chaque fois de les adapter aux réseaux mobiles.

Dans le chapitre qui suit, nous proposons notre contribution qui consiste en un algorithme de checkpointing dans un réseau mobile Ad Hoc.

Chapitre 7

1. Introduction

Nous avons étudié dans les chapitres précédents les techniques de tolérance aux fautes par reprise dans les systèmes répartis fixes, ainsi que dans les environnements mobiles. Ces derniers, nous oblige à changer de vision, quant à la conception de nouveaux protocoles adaptés à cet environnement qui en même temps nous offre plus de flexibilité mais aussi plus de contraintes, plus de vulnérabilité et donc moins de résistance aux fautes. Afin de permettre l'exécution jusqu'à son terme d'une application parallèle dans un tel système, il est essentiel de mettre en œuvre des mécanismes de tolérance aux fautes. Ces mécanismes doivent tenir compte de l'architecture particulière que représente un tel système, ceci afin d'obtenir un surcoût le plus faible possible. Il faut prendre en considération un certain nombre de paramètres, tels que les déconnexions, les handoff, batterie et mémoire limitées, faible bande passante partagée.

Nous avons étudiés quelques algorithmes de checkpointing dans les réseaux mobiles avec infrastructure, et on peut dès lors constater que ces derniers ont été largement exploités depuis 1993. Toutefois, les réseaux mobiles ad hoc n'ont pas bénéficié suffisamment de cette attention à ce jour. A notre connaissance, il n'y a pas eu de propositions satisfaisantes, et c'est ce qui a motivé notre travail, qui consiste à explorer cet environnement peu trivial et tenter d'apporter une quelconque contribution aussi minime soit elle.

Dans ce chapitre, nous présentons notre protocole de checkpointing en environnement mobile. C'est un algorithme adapté aux réseaux mobile de type ad hoc. Le modèle et hypothèses considérés sont exposés. La suite de ce chapitre présente la conception proposée, une preuve de cohérence et enfin un exemple de scénario d'exécution.

2. Modèle du système considéré

2.1 Architecture

Le système est constitué d'un ensemble de processus P1, P2, … Pn résidant chacun sur un terminal mobile appelé *MH*. L'inondation pure peut être très coûteuse et risque de déranger inutilement les *MH* en mode « sleep », dans le but de réduire cette inondation

lors du routage, nous supposons que le réseau soit hiérarchisé et auto- organisé. Il est constitué de groupes appelés **cluster**. Chaque groupe est constitué d'un coordinateur appelé **clusterhead (*CLH*)** et éventuellement de nœuds ordinaires. Un *MH* communique avec d'autres *MH* à travers son clusterhead local.

Les processus ne disposent ni de mémoire commune ni d'horloge globale. Chaque processus a donc accès directement à la mémoire de son *MH* ou à son *CLH* via l'envoi de messages. La communication interprocessus n'est réalisée qu'à travers l'émission de messages. Tous les terminaux mobiles, *MH* et *CLH* forment un réseau sans fil.

L'algorithme proposé, traite particulièrement les messages *Orphelin* [17], puisque ces derniers sont la cause de l'inconsistance de l'état global.

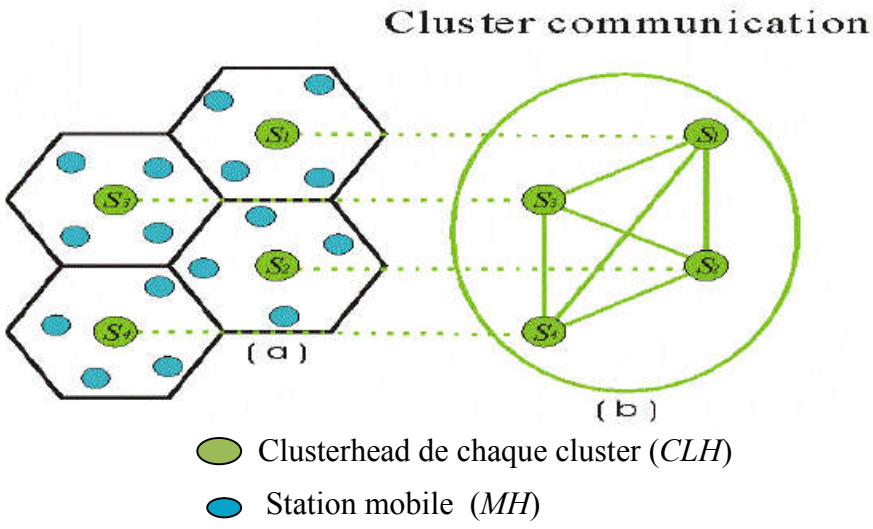


Figure 7.1 : Modèle du réseaux mobile.

2.2 Hypothèses sur le réseau

Nous supposons qu'en absence de défaillance de nœuds, le réseau est fiable, c'est-à-dire qu'un message envoyé sera reçu dans un temps arbitraire mais fini. Nous supposons aussi que les clusterheads seront choisis parmi les nœuds les plus faiblement mobiles, avec une autonomie en énergie élevée, de grande capacité de stockage et de mémoire et que lors de la procédure de checkpointing, les *CLH* ne se déconnectent pas et ne se déplacent pas.

2.3 Modèle de fautes

Nous considérons le modèle de défaillance « *fail-stop* » [6], c'est-à-dire qu'un nœud cesse de fonctionner en cas de faute. Nous ne considérons pas les fautes d'omissions ni les fautes byzantines. Nous considérons aussi qu'un seul site peut être défaillant à un moment donné.

3. Principe de conception du protocole

La Conception d'un algorithme de checkpointing pour les réseaux mobiles ad hoc, doit prendre en compte ses caractéristiques spécifiques et les traiter de manière appropriée. Elle doit aussi utiliser une hybridation entre les différentes approches afin de tirer profit de leurs avantages et adapter au mieux l'algorithme à son contexte d'exécution.

La faible largeur de la bande passante du réseau sans fil ne nous permet pas de l'encombrer avec un trafic intensif. De même l'inexistence de station de base fixe (qui résous beaucoup de problèmes dans les réseaux avec infrastructure) et la limitation de la batterie des sites mobiles, ne nous permet pas de faire trop de calcul. Sans oublier la faible capacité des sites mobiles et l'instabilité de leurs supports de stockage par rapport aux stations de base.

Ceci nous amène à concevoir un algorithme qui utilise l'approche coordonnée et l'approche induite par la communication.

- l'approche induite par la communication (basée sur index), n'engendre aucun coût de synchronisation, puisque la synchronisation est implicite et les informations permettant cette synchronisation sont encapsulées avec les messages de calcul. Donc, il n y a aucun risque de déranger inutilement les sites en mode « *Doze* ». lorsqu'un site reçoit un message avec un index supérieur à son index, il doit prendre un point de reprise *forcé* avant de traiter le message. Ceci permet d'éviter les points de reprise inutiles [32].

  - L'approche coordonnée, nous permet d'avoir un état global cohérent apriori formé par tous les points de reprises locaux (pas de messages orphelins). Et ceci est très important pour un recouvrement facile, car après une défaillance, il suffit que chaque site retourne à son dernier point de reprise permanent sauvegardé et recommence

l'exécution. De même pour chaque site, il maintient seulement un point de reprise au maximum, ce qui aide à minimiser l'espace de stockage nécessaire pour la sauvegarde des checkpoints. Cette approche est combinée avec l'approche induite par la communication, ce qui nécessite une synchronisation avec seulement les sites qui n'ont pas participé lors de cette dernière approche (i.e induite par la communication). Ce qui diminue grandement le coût de synchronisation et donc le trafic sur le réseau sans fil.

- La coordination est non bloquante, c'est-à-dire qu'un site ne force pas le blocage momentané de l'exécution de l'application distribuée pendant l'exécution de l'algorithme de Checkpointing. Ce qui augmente les performances de façon significative, puisque l'algorithme est exécuté de façon transparente.

- La coordination est globale, c'est à dire que tous les sites sont impliqués par la procédure de checkpointing. Ceci peut paraître à première vue, contraignant, par rapport à la coordination minimale qui elle n'oblige pas tous les sites à prendre un point de reprise. Cependant la coordination minimum engendre un surcoût de recherche des sites, et une propagation de la requête de checkpointing. Plusieurs passes sont nécessaires pour déterminer tous les sites dépendants du site initiateur. Ceci entraîne une consommation de la bande passante et un calcul considérable ainsi qu'une latence dans l'algorithme de checkpointing. Nous préférons donc une coordination coordonnée globale qui s'exécute en une seule passe.

3.1 Traitement de la mobilité des sites

L'architecture hiérarchisée a un **avantage** quant à la mobilité d'un MH au sein du même cluster. Il peut bouger librement sans rien changer du moment qu'il est toujours dans le même cluster. L'auto-organisation offre une vue virtuelle plus stable de la topologie et l'impact de la mobilité sur la topologie est donc optimisé par ce biais.

Lorsque le site mobile MH_i *se déplace,* il envoie une requête de déplacement à la nouvelle station de base MSS_j. Cette dernière envoie une requête à la l'ancienne station de base MSS_i pour récupérer les informations concernant le site mobile MH_i. A la réception de la requête du Handoff, l'ancienne station de base MSS_i envoie à la nouvelle station de base MSS_j les informations du site mobile concerné.

Routines du Hand off

Rôle du site mobile MH_i

1- Lorsque MH_i envoie une requête de déplacement au nouveau clusterhead CLH_j **:**

SendHandOfReq (MH_i , CLH_i) *to new clusterhead* CLH_j

Rôle du nouveau clusterhead CLH_j:

2- Lorsque le clusterhead CLH_j reçoit une requête de déplacement **:**

ReceiveHandOfReq (MH_i, CLH_i) ;
Cluster[i]:= 1 ;

SendTransReq (CLH_i, CLH_j, MH_i) to old clusterhead CLH_i ;

3- Lorsque le clusterhead CLHj reçoit des informations :

ReceiveInformation (CLH_j, MH_i, $index_i$) ;
SaveInformation ($index_i$) ;

Rôle de l'ancien clusterhead CLH_i :

4- Lorsque le clusterhead CLHi reçoit une requête de transmission des informations :

ReceiveTransReq (CLH_i, CLH_j, MH_i) ;
SendInformation (CLH_j, MH_i, $index_i$) to new clusterhead CLH_j ;

Cluster [i] := 0 ;

3.2 Traitement des déconnexions des sites

La déconnexion est aussi un autre problème qui affecte les performances du checkpointing dans les réseaux ad hoc. Il ya deux types de déconnexions : la déconnexion volontaire et la déconnexion involontaire.

Nous prenons en compte les *déconnexions volontaires*, tandis que celles qui sont involontaires, elles sont considérées comme des fautes.

La technique utilisée est celle de Cao et Singhal dans [11]. Lorsque MH_i veut se déconnecter de son cluster local, CLH_i, il prend un checkpoint local et transfère son checkpoint dans son CLH_i. Si lors de l'intervalle de déconnection, MH_i a été invoqué pour prendre un checkpoint, le CLH_i répond avec ce checkpoint et convertit ce checkpoint en un checkpoint permanent.

Après la reconnexion, le site mobile MH_i exécute une routine de reconnexion par l'envoi de l'identité du dernier clusterhead CLH_i avec lequel il a été connecté avant la déconnexion, au nouveau clusterhead courant CLH_j. CLH_j pourra ainsi récupérer le numéro du dernier point de reprise de MH_i, et le journal des messages arrivés au profit de MH_i pendant que MH_i était déconnecté.

Routines de déconnexion

Rôle du site mobile MH_i :

1- Avant la déconnexion :

If ($\exists$ Tentative$Ckpt_i$) Then
  SendDecReq (MH_i , $TentativeCkpt_i$) to local CLH;
Else
  $index_i := index_i + 1$;
  Take $DeconnexionCkpt_i$;
  SendDecReq (MH_i , $DeconnexionCkpt_i$,$index_i$) to local CLH ;
EndIf

Rôle Du clusterhead CLH_i:

2- Lorsque le clusterhead reçoit une requête de déconnexion :

If (*ReceiveDecReq* (*MHi* , $TentativeCkpt_i$,$index_i$)) Then
  SaveCkpt (MH_i , $TentativeCkpt_i$, $index_i$) ;
Else
  If *ReceiveDecReq* (MH_i , $DeconnexionCkpt_i$,$index_i$) Then
    SaveCkpt (MH_i , $CkptDéconnexion_i$,$index_i$) ;
  EndIf
EndIf
Connect [i] := 0 ;

Until (*ReceiveTranstReq* (CLH^i , CLH^j , MH^i)) Do
  A la réception d'un message de calcul de MH_j pour MH_i:
  ReceiveMessage (MH_j, MH_i , m_index_j , *Message*) ;
  If $m\_index_j > index_i$ Then $index_i := m\_index_j$; // pour éviter les checkpoints inutiles
  SaveMessage (MH_j , MH_i , m_index_j, *Message*) in Log_i ;
EndDo

Routine de Reconnexion :

Rôle du site mobile MH_i:

1- A la reconnexion :
SendRecReq (MH_i , CLH_i) to new clusterhead CLH_i ;

2- Lorsque le site mobile MHi reçoit un point de reprise :

ReceiveCkpt (MH_i , $DeconnexionCkpt_i$, $index_i$) ;
SaveCkpt (Deconnexion$Ckpt_i$, $index_i$) ;

3- Lorsque le site mobile MH_i reçoit un message de calcul enregistré par CLH_j :

ReceiveLogMessage (CLH_j , MH_i , m_index_j, *Message*) ;
If $m\_index_j > index_i$ then $index_i := m\_index_j$;
EndIf ;
Process the message

4- Lorsque le site mobile MH_i reçoit un message de calcul de MH_j :

ReceiveMessage (MH_j , MH_i , *m_index, Message)*
If $m\_index_j > index_i$ then $index_i := m\_index_j$;
EndIf ;
Process the message

Rôle du nouveau clusterhead CLH^j :

5- A la réception d'une requête de reconnexion :

ReceiveRecReq (MH_i , CLH_i) ;
Cluster [i] := 1 ;
Connect [i] := 1 ;
SendTransReq (CLH_i , CLH_j , MH_i) to CLH_j;

6- A la réception des informations d'un site mobile MH_i :

ReceiveInformation (CLH_j , MH_i , $index_i$, $CkptDéconnexion_i$, Log_i) ;
SendCkpt (MH_i , $DeconnexionCkpt_i$, $index_i$) ;
For (All (CLH_i , MH_j , *MHi* , m_index_j, *Message*) $\in$ Log_i) Do
SendLogMessage (MH_j , MH_i , m_index_j, *Message*) to MH_i;
Fin
If level2 = true then Recept [i] := 1 ; EndIf

Rôle de l'ancien clusterhead CLH^i :

7- A la réception d'une requête de transmission des informations :

ReceiveTransReq (CLH*i* , *CLHj* , *MHi*) ;
SendInformation (CLH_j , MH_i , $index_i$, $DeconnexionCkpt_i$, Log_i) ;
Cluster [i] := 0 ;
Connect [i] := 0 ;

3.3 Constitution du support de stockage stable

Les points de reprises doivent être sauvegardés dans un endroit sûr et qui leur permettra de rester accessibles même après une défaillance. Le support de stockage d'un site mobile est instable et n'est donc pas considéré comme stable. Dans les réseaux avec infrastructure, les points de reprises des sites mobiles sont stockés dans le support des stations de bases, puisque ces dernières sont dotées d'une grande capacité de stockage et sont stables.

Dans notre cas, tous les sites sont pareils, du point de vue instabilité et mobilité, l'existence d'une infrastructure est inexistante. La solution retenue, pour constituer un support de stockage stable, est l'utilisation de la mémoire des sites voisins [75]. Ainsi, un site mobile MH_a qui veut rendre son checkpoint permanent, sauvegarde son point de reprise dans la mémoire de ses sites voisins. Chaque voisin qui a pu sauvegarder ce checkpoint, renvoie son identificateur au site mobile MH_a. Cette méthode offre une meilleure résistance aux fautes.

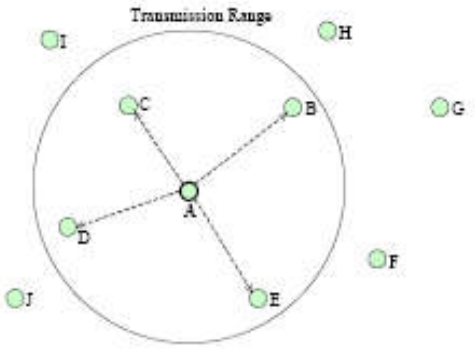


Figure 7.2 : Constitution d'un support de stockage stable.

4. Description du protocole

Cette partie décrit le protocole de reprise d'applications parallèles que nous avons conçu pour les réseaux mobiles ad hoc. Ce protocole se veut être aussi *simple* que possible, le premier but est d'assurer la propriété de *sûreté* (i.e, assurer qu'il y ait toujours des états locaux sauvegardés permettant un retour arrière, et que lors d'un retour arrière, l'état global restauré soit cohérent). Ensuite il doit être aussi *efficace* que possible (respectant les spécificités du réseau ad hoc).

Les mécanismes de points de reprise coordonnés sont intéressants de part leur simplicité, et font l'unanimité dans la littérature, mais malheureusement ils engendrent un surplus de trafic du aux messages de coordination. Par ailleurs, les protocoles induits par les communications basés sur index sont un compromis entre la sauvegarde coordonnée et la sauvegarde non coordonné. Afin de tirer profit de ces deux mécanismes, nous avons proposé un algorithme *hybride* inspiré de [71], il joint l'avantage des protocoles coordonnés et les protocoles basés index grâce aux trois phases qui sont :

1. **La phase de pré-synchronisation** qui consiste à coordonner les clusterhead entre eux Dans cette phase, l'algorithme déplace la tache de la synchronisation du site mobile ordinaire vers les clusterhead, afin de réduire la complexité et le coût de synchronisation de *O(n)* à *O(m)*, où n et m sont respectivement le nombre de sites mobiles et le nombre de clusterhead (m << n). Ainsi, un site coordinateur envoi une requête de Checkpointing à tous les clusterhead. (y compris lui-même).

2. **La phase quasi-synchrone** se déroule entre *CLH* et les *MHs* locaux Dans cette phase, chaque clusterhead qui reçoit la requête de Checkpointing, lance son temporisateur local T_{lazy} et attend l'expiration de ce délai. Durant ce temps là, le *CLH* profitera pour transférer la requête à chaque MH qui recevra un message de calcul, en embarquant son numéro de séquence de checkpoint (index) dans le message. Ainsi, chaque site mobile MH_i, qui reçoit un message de calcul avec un numéro de séquence plus grand que le sien, MH_i prend un point de reprise *forcé* qu'il sauvegardera dans sa mémoire locale (comme les checkpoints mutables). Ceci évite le surplus de trafic sur le réseau sans fil, et donc économise la bande passante.

3. **La phase post-synchronisation**, pendant cette phase, chaque clusterhead, dont le temporisateur T_{lazy} vient d'expirer, envoie une requête de Checkpointing vers tous les sites qui n'ont reçu aucun message de calcul depuis que ce clusterhead a reçu la requête

de Checkpointing (i.e pendant la deuxième phase) et prend lui aussi un checkpoint qu'il sauvegardera dans sa mémoire locale. Lorsque le *CLH* reçoit toutes les réponses des sites auxquels il a envoyé une requête, il envoie une réponse au Coordinateur initiateur. Ce dernier lorsqu'il récolte toutes les réponses des *CLH* invoqués, il envoie à son tour une validation à ces derniers, qui vont à leur tour diffuser la validation au site mobiles membres de leur cluster. A la réception de validation, ces derniers vont transférer leur point de reprise sauvegardés localement, vers le support de stockage stable.

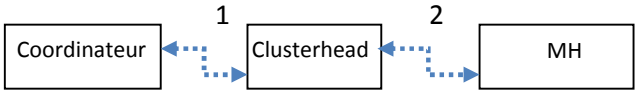
Phase	Situation	Stratégie
1	 Entre *CLH coordinateur* et *les CLHs*	**Pré-synchronisation**
2	 Entre *CLH local* et les membres de son cluster	**Quasi-synchrone**
3	 *CLH* informe le coordinateur et informe ses *MHs* membres	**Post-synchronisation**

Tableau 7.1 : Etapes de l'algorithme.

A. Détermination du temps T_{Lazy}

Pour fixer le temps T_{Lazy}, nous proposons qu'il soit fixé selon le système. Il ne doit pas être assez grand (tend vers l'infini) car le protocole de checkpointing deviendra quasi-synchrone, et ne doit pas être non plus assez court (tend ver zéro) car le protocole deviendra coordonné. Il est possible de prendre T_{Lazy} selon les statistiques du système. Il correspondra, par exemple, au temps moyen pour qu'au moins 50 % des *MHs* dans le cluster reçoivent au moins un message de calcul.

B. Structures de données :

Les structures de données utilisées sont les suivantes :

Index$_i$: numéro de séquence du point de reprise.

Cp_state $_i$: état du site ; cp_state$_i$ =1 => le site MHi est en état de checkpointing.

Terminaison : variable de détection de la terminaison de l'algorithme de checkpointing, m*aintenue au niveau du clusterhead coordinateur*

Weight$_i$, : variable de détection de la terminaison de synchronisation entre le clusterhead et les membres de son cluster, *maintenue au niveau de chaque clusterhead.*

Coordinateur : identificateur du clusterhead coordinateur du checkpointing.

Cluster : vecteur de bit. Cluster[i] =1 => le site MH_i est membre du le cluster.

Connect : vecteur de bit. Connect[i] =1 => le site MH_i est connecté au cluster.

Recept : vecteur de bit. Recept[k]= 1 => le MH_k a reçu un message pendant la 2ème phase.

Level2 : variable booléenne, elle est égale à vrai si l'algorithme est en phase2.

Pour chaque site mobile *MH$_i$* ordinaire:

Index $_i$,cp_state$_i$: integer ;

Pour chaque clusterhead *CLH $_i$* :

Coordinator : Identificateur;
index $_i$: Integer ;
weight $_i$, Terminaison : real;
Cluster, Connect, Recept : array [1..*n*] of Bit;
Level2 : Boolean ;

C. Initialisations de l'algorithme

Pour chaque site mobile *MH$_i$* :

index ^{i} := 0 ; weight$_i$:0; cp_state$_i$:=0 ;

Pour chaque Clusterhead *CLH $_i$* :

For k := 1 to n Do
Recept [k] := 0 ;
EndDo ;
Cluster := 1 for all MH_i / $MH_i \in Cell_i$;
Connect = Cluster ;
weight$_i$:=0; coordinator := ϕ ; Terminaison := 0 ;

5. Algorithme de checkpointing :

A/ Initiation de l'algorithme :

Rôle du site initiateur MH_{init} :

1. Lorsque le site mobile MH_{init} initialise une procédure de checkpointing

```
If (MHinit is Clusterhead ) Then            // MHinit est le coordinateur
    Terminaison :=1;
    Cp-stateinit :=1
    indexinit :=indexinit + 1 ; // l'incrémentation se fait seulement au niveau du coordinateur
    For All CLHj Do
        Terminaison :=terminaison /2 ;
        SendChpReq (CLHinit, CLHj, indexinit, terminaison ) To All CLHj ;
    EndDo
Else    SendCoordReq (MHinit, CLHj) to Local Clusterhead CLHj ;
EndIf
```

Rôle du ClusterHead Coordinateur :

2. Lorsque le clusterhead CLH_i reçoit une requête de coordination

```
ReceiveCoordReq (MHinit, CLHi)
  If cp-statei < >0 then ignore request
  Else
     Cp_statei :=1 ;
     Terminaison :=1;
     index i :=index i + 1 ;     // l'incrémentation se fait seulement au niveau du coordinateur
     For All CLHj Do
         Terminaison :=Terminaison /2 ;
         SendChpReq (CLHi, CLHj, index i, terminaison) To All CLHj ;
     EndDo
  EndIf
```

B/ Pendant l'exécution de l'algorithme:

Procedure Checkpointing (MHi, index, m_weight_i)

```
Take Tentative checkpoint Ci, index
If (MHi is ordinary) Then   SendReply (MHi, CLHj, m_wheighti) to local clusterhead ;
EndIf
```

Function Empty (vecteur) : Boolean ;

```
k :=1 ; Continue := True ;                          // si vecteur vide =>empty =true
While k <= n AND continue Do
   If vecteur [k]= 1 then continue := False EndIf
   K:=k +1 ;
EndDo ;
If (continue) Then empty :=True   Else  Empty :=  False  ;
EndIf;
```

Rôle de chaque clusterhead CLH_i :

3. Lorsque CLH_i reçoit une requête de checkpointing de la part du Coordinateur CLH_{init}

ReceiveChpReq (CLH_{init}, CLHi, m_index, m-weight)

```
If  cp_state_i < > 0 then          //  ignore le request
Else        Cp_state_i :=1 ;
            Coordinator:= CLH_init ;
            weight_i := m_weight ;
            If  m_index  > index_i  Then   index_i := m_index  ;     EndIf ;
            For  k := 1 to n Do  Recept [k] :=0  EndDo ;
            Level2 :=True;          // début de la 2ème phase
            SetTimer T_Lazy ;
EndIf
```

4. Lorsque T_{Lazy} arrive à terme pour CLH_i :

```
Level2 := False ;       // fin de la 2ème phase
List := (Cluster) AND (Connect) AND (Not (Recept))   // Les MH du cluster  qui n'ont pas reçu de
                                                                  // message et ne sont pas  déconnectés
If Empty (List )    Then                          // la liste est vide; tous les MH ont pris un checkpoint
      SendReply (CLH_init, weight_i);
Else   For k := 1 to n Do
          If  k = i then  Checkpointing (CLH_i , index_i , 0)
           Else  If List [k] = 1 and (k <> i) Then    mw_i := mw_i /2 ;
                                        SendChpReq (CLH_i, MH_k, index_i, mw_i ) to MH_k ;
                 EndIf
           Endif
        EndDo
EndIf
```

5. Lorsque CLHi envoie un message de calcul à MH_j (clusterhead ou ordinaire) :

SendMessage (CLHi, CLHj, $index_i$, Message) ;

6. Lorsque CLHj reçoit un message de calcul de MH_i (clusterhead ou ordinaire) :

ReceiveMessage (MHi, CLHj, m_index, message)

```
If  m_index > index_i  Then     index_i := m_index ;
                                Checkpointing (CLH_j , m_index, 0)
EndIf
Process the message ;
```

7. Lorsque CLH_p reçoit un message de calcul envoyé par MH_i vers MH_j :

ReceiveMessage (MHi, MHj, m_index, message)

```
if  m_index > index_p  Then     Ckpt_p := m_index ;      // pour éviter les checkpoint inutiles
EndIf
If  (Recept [j] =0) AND (level 2= true)  Then Recept [j] := 1  EndIf ;
If  (Connect[j] =0 ) AND ( Cluster [k] = 1 )
     Then  SaveMessage( MH_i, MH_j, ckpt_p, Message)  in  Log_j  EndIf;
If  (Connect [k] = 1 ) AND (Cluster [k] =1 )
     Then   SendMessage (CLH_p, MH_j, index_p, Message)  EndIf ;
If  Cluster [k] =0  Then Search MH_j and Send Message to CLH_q   EndIf ;
```

8. Lorsque CLHj reçoit une réponse de MHi (ordinaire) :

ReceiveReply (MHi, CLHj, m_weight)
Mw_i :=mw_i + m_weight ;
If (mwi = $weight_i$) AND (weighti <>1) AND (level2 =False)
Then *SendReply (CLH_j, Coordinator, $weight_i$)* ;
EndIf;

9. Lorsque CLHj reçoit une validation du coordinateur initiateur CLHinit :

ReceiveCommit (CLH_{init}, CLH_j)
For K := 1 to n Do
If (Cluster [k] =1) AND (Connect [k] =1) Then
SendCommit (CLH_j, MK_k) to MH_k ;
Else
If (Cluster [k] =1) AND (Connect [k] =0) Then Make $DeconnexionCkpt_k$ Permanent
EndIf
EndIf
Make TentativeCkpt$_j$ permanent
cp_state $_j$:=0 ;

10. Lorsque CLH_{init} reçoit une réponse de CLH_j

ReceiveReply (CLH_j, CLH_{init}, m_weight)
Terminaison ;= Terminaison + m_weight ;
If Terminaison = 1 Then
For ALL *CLHj* Do *SendCommit (CLH_{init}, CLH_j)* EndDo
EndIf

Rôle pour chaque site MHi

11. Lorsque MHi reçoit un message de calcul de MHj

ReceiveMessage (MHj, MHi, m_index, Message)
If (m_index > $index_i$) Then $index_i$:= m_index;
Checkpointing (MH_i, $index_i$, $weight_i$)
EndIf ;
Process the message;

12. Lorsque MHi reçoit une requête de checkpointing :

ReceiveChpReq (CLHj, MHi, m_index, m_weight) ;
$weight_i$:= m_weight ;
If (m_index > index $_i$) Then $index_i$:= m_index;
Checkpointing (MH_i, $index_i$,$weight_i$) EndIf ;

13. Lorsque MHi envoie un message de calcul à MHj (clusterhead ou ordinaire) :

SendMessage (MH_i, CLH_j, index $_i$, Message) ;

14. Lorsque MHi reçoit une validation de son clusterhead local :

ReceiveCommit (CLHj, MHi);
Make $TentativeCkpt_i$ permanent ;

6. *Validation du protocole*

Afin de valider notre protocole de checkpointing, nous allons prouver analytiquement la cohérence de l'état global, ainsi que sa convergence (terminaison).

6.1 Preuve de cohérence

Lemme1 : Si MH_i et MH_j prennent respectivement les points de reprise $C_{i,\ indexi}$ et $C_{j,\ indexj}$ avec $index_i = index_j$ Alors les deux points de reprise sont cohérents.

Preuve : On suppose que MH_i et MH_j prennent respectivement les points de reprise $C_{i,indexi}$ et $C_{j,indexj}$ avec $index_i = index_j$, Mais que les deux checkpoints ne sont pas cohérents. D'après Netzer et xu dans [32], il existe **un chemin ZigZag** ZP entre $C_{i,indexi}$ et $C_{j,indexj}$. Plusieurs cas peuvent se présenter :

Cas 1 : si la longueur de ZP est ''un'', MH_i envoie un message m avec l'index $index_i$ après avoir pris $C_{i,\ indexi}$. MH_j reçoit m avant de prendre $C_{j,\ indexj}$. Donc $index_i$ envoyé avec le message m doit être égal à $index_j$*-1*. Ce cas ne peut se présenter car MH_j prend $C_{j,indexj}$ avant de recevoir le message seulement si $index_i$ est égal à $index_j$.

Cas 2 : Si la longueur de ZP est '' deux'', il y a trois situations :

(1) Après avoir pris $C_{i,\ indexi}$, MH_i envoie le message m1 à MH_k. MH_k prend d'abord un checkpoint $C_{k,indexk}$ avant de traiter m1 et après envoie le message m2 à MH_J avant que MH_j prenne le checkpoint $C_{j,indexj}$. Cette situation est la même que dans cas1. Donc elle ne peut se présenter.

(2) MH_k n'a pas pris de point de reprise $C_{k,indexk}$ avant de recevoir le message m1 mais cette situation a le même résultat que pour la situation (1) et donc elle ne peut se présenter.

(3) l'ordre de réception de m1 et m2 chez MH_k est inversé. Aucun état d'incohérence ne peut se présenter si $index_i$ est égal ou inférieur à $index_k$. MH_k doit prendre le point de reprise $C_{k,indexk}$ avant de recevoir m1 si $index_i$ est plus grand que $index_j$, ainsi, le chemin ZigZag ZP est rompu.

En résumé, le cas 2 ne peut se présenter.

Cas 3 : si la longueur de ZP est n, les points de reprise $C_{i,\ indexi}$ et $C_{j,\ indexj}$ sont cohérents. Dans le cas précédent lorsque la longueur était deux, nous avons prouvé que $C_{i,indexi}$ et $C_{j,indexj}$ ne peuvent être incohérents. On peut réduire n à (n-2) itérativement

jusqu'à n égal à un ou à deux et Si MH_i et MH_j prennent respectivement les points de reprise $C_{i,\ indexi}$ et $C_{j,\ indexj}$ avec $index_i = index_j$ Alors les deux points de reprise sont cohérents.

Lemme 2 : Si MH_i et MH_j prennent respectivement les points de reprise $C_{i,\ indexi}$ et $C_{j,\ indexj}$ avec $index_i = index_j$ Alors les deux points de reprise $C_{i,indexi}$ et $C_{j,indexj}$ sont pris pour le même initiateur.

Preuve : l'incrémentation de l'index du point de reprise se fait au niveau du clusterhead coordinateur ''seulement''. D'après cette affirmation et d'après le lemme1, $C_{i,\ indexi}$ et $C_{j,indexj}$ sont cohérents (puisque $index_i = index_j$). Donc nous déduisons que ces deux checkpoints sont pris pour la même initiation de checkpointing. Et c'est le *CLH* coordinateur qui initie la procédure du checkpointing.

Lemme 3 : un état global est obtenu dans un temps fini après que le coordinateur initie la procédure de checkpointing.

Preuve : l'algorithme doit accomplir les étapes suivantes :

1- le *CLH* coordinateur envoie une requête de checkpointing à tous les CLH_i (y compris lui-même).

2- chaque CLH_i qui reçoit la requête, initialise un temporisateur T_{Lazy}.

3- lorsque le temporisateur est expiré, CLH_i envoie une requête de checkpointing aux membres de son cluster MHs qui n'ont pas reçu de messages durant la phase2 (i.e avant que T_{Lazy} ne soit expiré).

4- lorsque CLH_i reçoit toutes les réponses de ses membres MHs invoqués, il envoie une réponse au clusterhead coordinateur.

5- lorsque le *CLH* coordinateur reçoit toutes les réponses des CLH_i, il envoie un message de validation à tous les CLH_i (y compris lui-même).

Chaque étape est achevée dans un temps fini, par conséquent, nous avons prouvé qu'un état global est obtenu par l'algorithme de checkpointing dans un temps fini.

Théorème : l'algorithme de checkpointing assure un état global cohérent.

Preuve : d'après le lemme1, deux points de reprise avec le même index sont cohérents, en outre, d'après le lemme1 et le lemme2, on déduit que l'état global cohérent est constitué d'états locaux avec des points de reprise ayant le même index. On sait aussi d'après le lemme3 que cet état global est obtenu dans un temps fini.

Cependant, d'après les trois lemme, l'algorithme de checkpointing assure un état global cohérent seulement si les MHs ne se déplacent pas et ne se déconnectent pas. Désormais nous considérons les trois cas suivants :

Cas 1 : pour les Mhs concernés par la déconnexion, il prennent un checkpoint de déconnexion avant de se déconnecter. Leur point de reprise deconnexionCkpt est utilisé comme checkpoint initial après leur reconnexion. Donc aucune incohérence n'est due aux déconnexios.

Cas 2 : pour les MHs en mode ''Doze'' (ou Sleep), l'algorithme envoie une requête de checkpointing pour les réveiller. Donc tous les MHs qui sont en mode ''Doze ''ne causent aucune incohérence de l'état global.

Cas 3 : lorsque un MH se déplce d'un cluster dont le chef est CLH_i vers un cluster dont le chef est CLH_j, le nouveau clusterhead CLH_j envoie une requête de Hand off à l'ancien clusterhead CLH_i pour récupérer le point de reprise de MH et son index. Donc la cohérence de l'état global est sauvegardée lorsque *MH* se déplace.

Selon les trois cas, on peut déduire que : *l'algorithme de checkpointing proposé assure un état global cohérent.*

6.2 Preuve de convergence

La terminaison de la coordination est détectée par les variables réelles dont la somme totale ne dépassera pas 1 comme c'est le cas dans [65] et [68]. Dans notre protocole, nous avons utilisé les variables réelles *Terminaison* au niveau du clusterhead coordinateur et *Weight* au niveau de chaque clusterhead. Elles permettent de détecter la fin de coordination vu que le nombre de site n'est pas connu à l'avance.

Lors de la première phase, la variable *Terminaison* est initialisée à 1 au niveau du *CLH* coordinateur. A chaque envoie de requête de checkpointing aux clusterhead (y compris lui-même), la variable *Terminaison* est divisée par 2. Une portion de la variable est envoyée avec la requête et l'autre est sauvegardée.

Lorsque les *CLH* reçoivent la requête, chacun initialise sa variable *weight* $_i$ avec la valeur reçue avec la requête. Cette variable est divisée par deux à chaque envoie de requête à un membre du cluster. Une partie de la variable *weight* $_i$ est envoyée avec la requête et l'autre partie est sauvegardée au niveau du clusterhead.

A chaque réponse à la requête, de la part d'un site mobile ordinaire, la portion de la variable $weight_i$ reçue est envoyée avec la réponse. Le *CLH* additionne son $weight_i$ avec la valeur reçue par la réponse. Si tous les sites invoqués ont répondu, la variable $weight_i$ de chaque clusterhead sera égale à la valeur reçue par le clusterhead coordinateur. Dans ce cas, le clusterhead CLH_i répond au clusterhead coordinateur.

A chaque réception d'une réponse de la part d'un *CLH*, le clusterhead coordinateur additionne sa variable Terminaison avec la valeur reçue. Lorsqu'il reçoit toutes les réponses des *CLH* invoqués, sa variable Terminaison sera égale à 1. Ainsi, il pourra envoyer un message de validation aux clusterhead pour rendre les points de reprise permanents.

En utilisant les variables *Terminaison* et $weight_i$, nous pouvons assurer que l'algorithme proposé converge.

7. Exemple de scénario

Voici le déroulement de l'algorithme sur un exemple de réseau ad hoc constitué de dix sites mobiles et de trois clusters, chacun géré par un clusterhead.

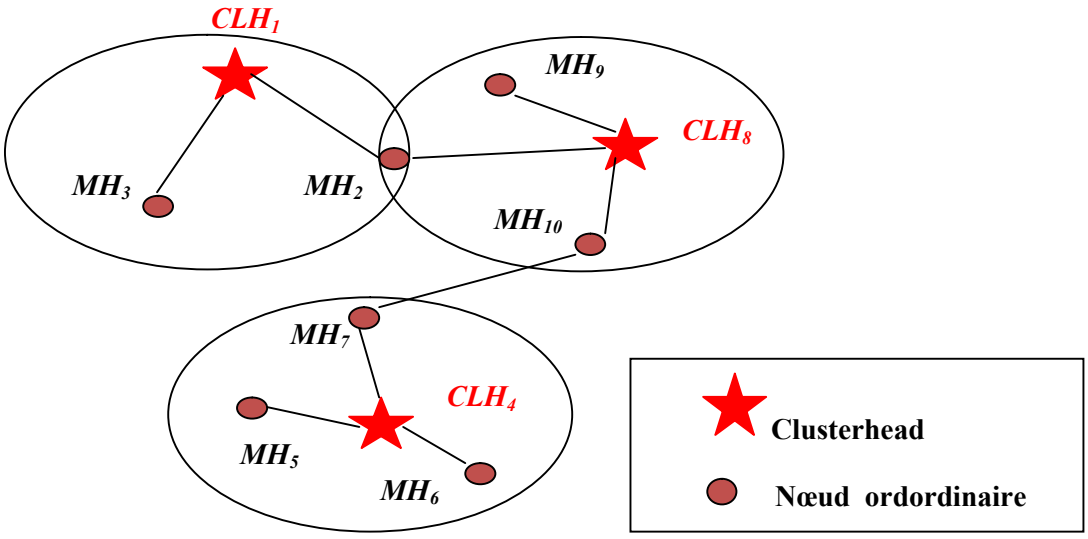
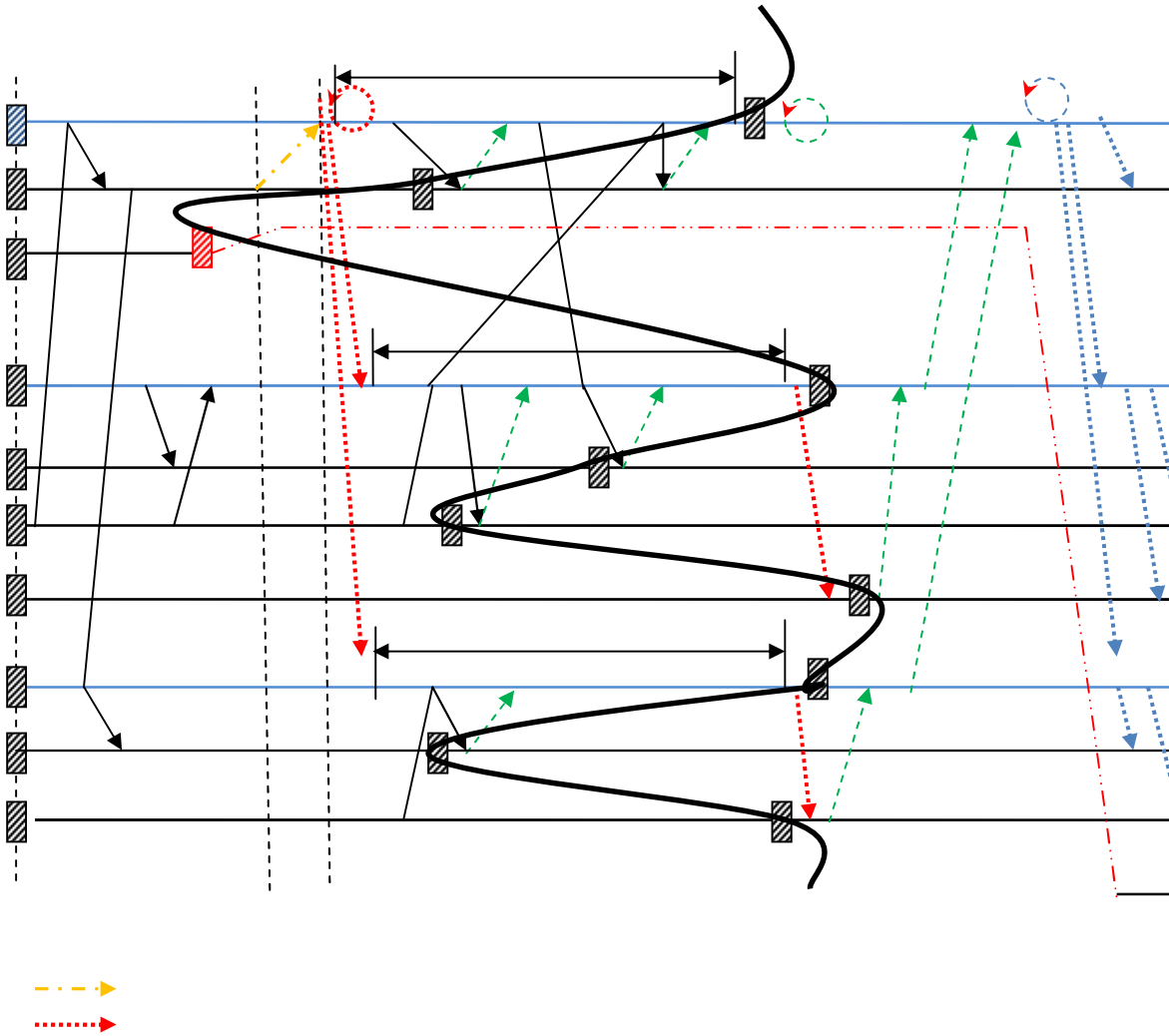


Figure 7.3 : Modélisation du réseau ad hoc.



8. Déroulement de l'exemple

A l'instant T1, MH_2 initie une procédure de checkpointing, il envoie une requête de coordination à CLH_1, pour que ce dernier devienne le coordinateur initiateur : c'est lui qui diffuse les requêtes, récolte les réponses et diffuse par la suite les validations.

Phase 1 :

(1) A l'instant T2, CLH_1 reçoit la requête de coordination(en orange pointillé), s'il est en état de checkpointing, il ignore la requête sinon il initie la procédure de checkpointing : il incrémente le numéro de séquence du checkpoint (index $_{init}$) et envoie des requêtes à tous les clusterhead (en rouge pointillé), à lui-même aussi, la requête contient l'index et la variable de terminaison partagée.

(2) chaque clusterhead qui reçoit la requête, met à jour son index (index $_i$:= index $_{init}$), sauvegarde l'id du coordinateur, puis attend pendant un temps T_{Lazy} avant d'envoyer les requêtes aux membres de son cluster.

Phase2 :

Pendant ce temps,

(1) *MH2* a reçu les message m1 et m2 de la part de CLH_1 et MH_6 (respec.), avant de traiter m1, il a pris un point de reprise C2,1 et envoyé une réponse à CLH_1.

(2) MH_6 a reçu un message m3 de la part de CLH_4, il a pris un point de reprise C6,1 avant de le traiter et envoie une réponse à CLH_4.

(3) MH_5 a reçu un message m4 de la part de CLH_1, il a pris un point de reprise C5,1 avant de le traiter et envoie une réponse à CLH_4.

(4) MH_9 a reçu un message m5 de la part de MH_{10}, il a pris un point de reprise C9,1 avant de le traiter et envoie une réponse à CLH_8.

Phase 3 :

- Lorsque le temporisateur est expiré pour :

  (1) CLH_1, il n'envoie aucune requête de checkpointing aux membres de son cluster, puisque MH_2 a déjà pris un point de reprise de manière non coordonnée (induite par communication), et MH_3 est déconnecté. Il prend son point de reprise C1,1 et envoie une réponse à lui-même, en renvoyant la portion de $weight_1$ qu'il a reçu.

  (2) CLH_4, il envoie une requête seulement à MH_7, puisque MH_5 et MH_6 ont déjà pris chacun un point de reprise de manière non coordonnée (induite par communication), Il prend son point de reprise C4,1 et attend la réponse de MH_7.

  (3) CLH_8 envoie une requête seulement à MH_{10}, puisque MH_9 a déjà pris chacun un point de reprise de manière non coordonnée (induite par communication), Il prend son point de reprise C8,1 et attend la réponse de MH_{10}.

- à la réception de la requête de checkpointing (en rouge pointillé) : MH_7 et MH_{10} prennent chacun un point de reprise (respec.) C7,1 et C10,1 et envoient chacun une réponse (en vert) à (respec.) CLH_4 et CLH_8.
- Lorsque CLH_4 reçoit la réponse deMH_7, il envoie à son tour une réponse au coordinateur *CLH1*.
- Lorsque CLH_8 reçoit la réponse de MH_{10}, il envoie à son tour une réponse au coordinateur CLH_1.
- Lorsque CLH_1 reçoit les réponses des clusterhead, il additionne sa variable terminaison, et dès que la somme devient égale à un, il sait qu'il a reçu toutes les réponses, il envoie des validations aux clusterhead (en bleu).
- Lorsque les clusterhead reçoivent les validations, ils diffusent à leur tour les validations à tous les membres du cluster et transforment leur point de reprise en un checkpoint permanent, et mettent à jour leur variable cp_state.
- Chaque site mobile membre d'un cluster qui reçoit une validation, transforme son point de reprise en un checkpoint permanent et cela en le sauvegardant dans le disque de ses sites voisins. FIN de l'algorithme.

Les points de reprises C1,1, C2,1, C3,1, C4,1, C5,1, C6,1, C7,1, C8,1, C9,1 et C10,1 forment un *état global cohérent*.

9. Analyse de performance

L'étude de l'art que nous avons effectué sur les algorithmes de checkpointing en environnement mobile, a concerné seulement les réseaux mobiles avec infrastructure. Ces derniers ont été largement exploités. Par contre les réseaux mobiles ad hoc n'ont pas bénéficié de cette attention. Il est donc difficile, voire même inadéquat de comparer notre algorithme avec ceux déjà étudiés, puisque l'architecture du réseau et les hypothèses de travail ne sont pas identiques.

Nous pouvons tout de même démontrer que notre protocole hybride par nature, est nettement meilleur en terme de performance par rapport à un algorithme complètement coordonné. Pour cela nous donnons quelques paramètres d'évaluation.

9.1 Notation

Nous utilisons les notations suivantes pour évaluer les performances de l'algorithme :

N : Nombre total de sites mobiles dans le système.(incluant les Clusterhead et les sites ordinaires).

$\mathbf{N_{CLUSTER}}$: Nombre de site mobiles ordinaires dans chaque cluster.

M : Nombre total des Clusterheads dans le système. $M << N$.

$\mathbf{N_{MIN}}$: le nombre exact de sites mobiles dépendants directement ou indirectement du site mobile initiateur. $1 \leq N_{MIN} \leq N$.

9.2 Paramètres d'évaluation

Pour évaluer la performance de notre algorithme, nous utilisons les cinq paramètres suivants :

1- Le nombre de messages de synchronisation
2- Le temps de blocage dans le pire des cas.
3- Le nombre de sites impliqués dans la procédure de checkpointing
4- Le nombre de points de reprises maintenus par chaque site.
5- Le surcoût dû à la recherche d'un site mobile.

9.3 Performance de l'algorithme

1) Le nombre de messages de synchronisation :

L'algorithme proposé pour les applications parallèles s'exécutant sur un réseau mobile ad hoc est un algorithme hybride, non bloquant. Il joint les avantages des **protocoles coordonnés** (nombre de points de reprise sauvegardés sur le support de stockage stable, recouvrement rapide et sans effet domino) et les avantages des **protocoles non coordonnés** (déranger le moins possible les *MHs*, pas de messages de contrôle).

L'algorithme est composé de trois étapes :

Etape 1 : le nombre de messages de coordination est égal à : **1+M**.
Etape 2 : le nombre de messages de coordination est **nul.**
Etape 3 : le nombre de messages de coordination est égal à :

- Dans le meilleur des cas à **0.** (Lorsque tous les sites ordinaires du cluster ont reçu un message de calcul pendant la période T_{lazy})
- Dans le pire des cas à $\mathbf{N_{CLUSTER}}$**.** (Lorsqu' aucun des sites ordinaires du cluster n'a reçu de message de calcul pendant la période T_{lazy}).

Donc, on conclue que le nombre de messages de synchronisations est égal à $1+M$ dans le meilleur des cas, et égale $1+M+N_{CLUSTER}$ dans le pire des cas.

2) Le temps de blocage :

Puisque l'algorithme de checkpointing s'exécute en même temps que le calcul proprement dit des sites mobiles, le temps de blocage est donc nul.

3) Le nombre de sites mobiles impliqués dans la procédure de checkpointing :

La coordination est globale, c'est à dire que tous les sites sont impliqués par la procédure de checkpointing. Ceci peut paraître à première vue, contraignant, par rapport à la coordination minimale qui elle n'oblige pas tous les sites à prendre un point de reprise. Cependant la coordination minimum engendre un surcoût de recherche des sites, et une propagation de la requête de checkpointing. Plusieurs passes sont nécessaires pour déterminer tous les sites dépendants du site initiateur. Ceci entraîne une consommation de la bande passante et un calcul considérable ainsi qu'une latence dans l'algorithme de checkpointing.

Donc le nombre de sites mobiles qui doivent prendre leur point de reprise est égal à N.

4) Le nombre de points de reprises maintenus par chaque site :

Parmi les avantages retenus des algorithmes coordonnées est la constitution d'un état global cohérent et le maintient d'au plus deux points de reprises pour chaque site.

Donc le nombre de points de reprises maintenus par chaque site est égal à ***2.(***Un point de reprise permanent et un autre provisoire).

5) le surcoût dû à la recherche d'un site mobile :

Les *MHs* ne recevront jamais de message de coordination de la part des *MHs* ou des *CLH* non locaux. Ceci supprimera le surcoût dû à la recherche d'un *MH*, et en même temps réduira le nombre de message de coordination, ce qui a en revanche a une implication directe sur **l'économie de l'énergie** de la batterie des sites mobiles et de **la bande passante**.

Donc le surcoût du à la recherche d'un site mobile est nul.

En plus de tout ce qu'on a dit, le fait de sauvegarder les points de reprise localement (en mémoire) nous permet de diminuer le transfert d'information à travers le réseau sans fil. Ainsi que la sauvegarde au niveau de la mémoire des sites voisin (diskless checkpointing) nous offre de meilleure performance au niveau du temps de transfert du checkpoint vers le support de stockage stable.

9.3 Comparaison

Comme nous l'avons déjà mentionné, l'algorithme que nous avons proposé est hybride, et joint les avantages de la stratégie coordonnée et ceux de la stratégie induite par la communication (basée index). Avec la même architecture de réseau et les mêmes hypothèses de travail, nous allons comparer notre protocole avec deux algorithmes coordonnés. L'un utilise la coordination globale et l'autre utilise la coordination partielle.

Cette comparaison est résumée dans le tableau suivant :

	Notre algorithme Hybride	**Algorithme Coordonné Global**	**Algorithme Coordonné Partiel**
Nombre de messages de synchronisation	Meilleur cas : $1+M$ Pire cas : $1+M+N_{CLUSTER}$	$1+M+N_{CLUSTER}$	$1+M+N_{MIN}$
Temps de blocage	**0**	**0**	**0**
Nombre de sites impliqués	N	N	N_{MIN}
Nombre de points de reprise par site	**2**	**2**	**2**
Surcoût de recherche de sites	**0**	**0**	*Valeur positive* **(plusieurs passes)**

Tableau 7.2 : Comparaison avec les algorithmes coordonnés.

Pour conclure, on peut dire que l'algorithme proposé offre les avantages suivants :

- Assurer un point de reprise global cohérent.
- Seulement deux points de reprises sont sauvegardés pour chaque site.
- Aucun site mobile ne doit être bloqué pendant l'exécution de l'algorithme.
- Ne consomme pas beaucoup d'énergie.
- N'encombre pas le réseau avec trop de message de coordination (économie de la bande passante).
- N'engendre pas de surcoût de recherche des *MHs*.
- Un coût minimum d'exécution.
- Supporte la déconnexion ou le déplacement d'un site mobile.

10. Bilan et conclusion

En plus des caractéristiques spécifiques des réseaux mobiles en général, (bande passante et batterie limitée, mobilité, déconnexion, vulnérabilité...), dans les réseaux mobiles ad hoc certaines de ces contraintes sont accentuées. Dépourvus d'infrastructure, la topologie est complètement dynamique et libre.

Contrairement aux réseaux avec infrastructure, ou dans un algorithme de checkpointing, la plus grande charge de calcul est effectuée par les stations de base (qui par rapport aux sites mobiles sont plus puissante et plus stables), dans les réseaux ad hoc, nous n'avons pas de station de base ; tous les sites sont pareils. Nous devons donc concevoir un algorithme simple et efficace, qui ne demande pas trop de calcul.

La majorité des algorithmes conçus pour les réseaux à point d'accès aussi performants qu'ils soient, sont totalement inadéquats pour les réseaux ad hoc.

Face à cette difficulté, nous avons essayé de concevoir un algorithme de checkpointing qui soit adapté aux caractéristiques des réseaux ad hoc. Et ceci en utilisant une hybridation entre la méthode coordonnée et la méthode induite par la communication. Joignant ainsi les avantages de chaque approche, et surtout faire le moins possible de calcul et déranger le moins possible de sites mobiles.

CONCLUSION GENERALE

Conclusion générale

Les travaux consignés dans cette thèse concernent l'étude, l'analyse et la proposition de protocole de tolérance aux fautes par recouvrement arrière (reprise) pour les applications parallèles s'exécutant sur des réseaux mobiles ad hoc. En effet, assurer la continuité de service dans un tel réseau (réseau sans infrastructure), malgré l'occurrence d'une défaillance, est un problème très complexe vu la dynamicité et l'évolution rapide de la topologie. Les unités mobiles sont dynamiquement et arbitrairement éparpillées d'une manière où l'interconnexion peut changer à tout moment.

Les réseaux mobile sans fil en général présentes des spécificités telles que : la mobilité et la vulnérabilité des stations de travail, les fréquentes déconnexions, les contraintes imposées par les ressources mobiles telle que la bande passante limitée des réseaux sans fil, la durée de vie de la batterie du terminal portable, la capacité mémoire et la vitesse d'exécution. Ces caractéristiques posent de nouveaux défis en termes de sûreté de fonctionnement, ce qui nous oblige à changer la vision classique des problèmes liés aux systèmes distribués.

Il existe donc un réel besoin en mécanismes de tolérance aux fautes pour pallier les défaillances lors de l'exécution des applications surtout si elles sont de longues durées, comme en témoigne d'ailleurs l'abondance des travaux sur les réseaux mobiles avec infrastructure. Toute défaillance est susceptible d'engendre une perte de donnée et de temps pour des applications qui, si elles ne sont pas critiques en terme de vies humaines, le sont souvent d'un point de vue économique.

Après l'étude que nous avons effectuée sur différents algorithmes de reprise dédiés aux réseaux sans fil avec infrastructure, nous avons constaté que tous ces protocoles sont en réalité ceux des réseaux fixes adaptés aux spécificités des réseaux mobiles. On remarque aussi qu'à chaque nouvel algorithme, il y a eu tentative d'optimisation soit sur la mémoire, soit sur l'espace, soit sur l'énergie. La présence des stations de bases (infrastructure) reste tout de même une *solution clé* et résout pas mal de problème. Dans chaque protocole quelque soit sa difficulté, la plus grande charge du travail est supportée pas les stations de base. Les stations mobiles (MH) sont déchargées de la synchronisation et du calcul ce qui leur permet d'économiser de l'énergie (batterie) et de la mémoire. La bande passante du réseau sans fil est par ce biais également économisée. Ajouté à cela, la stabilité et la grande capacité des stations de bases, ce qui leur permet de constituer un support stable pour sauvegarder les points de reprise des terminaux mobiles.

Par contre dans notre configuration concernant les réseaux mobile ad hoc (MANET), on ne peut s'offrir ce luxe. L'inexistence d'infrastructure constitue une contrainte *handicapante* qu'il faut détourner en changeant la conception d'un algorithme de checkpointing. Toutes les stations du réseau sont mobiles et identiques dans leurs performances, il n'y a aucune qui peut être considérée comme super station, ni comme support stable de stockage. Il est donc impératif de concevoir un algorithme peut gourmant en mémoire, en capacité de stockage, et qui nécessite pas trop de calcul ni trop de trafic. Ceci dans le but de conserver l'énergie des stations mobiles et la bande passante du réseau sans fil.

Face à cette difficulté, nous avons tenté d'apporter une contribution en concevant un algorithme de checkpointing dédié aux applications s'exécutant sur des MANET. C'est un algorithme *hybride*, qui combine l'approche coordonnée et l'approche induite par la communication. Les mécanismes de points de reprise coordonnés sont intéressants de part leur simplicité, et font l'unanimité dans la littérature, mais malheureusement ils engendrent un surplus de trafic du aux messages de coordination. Par ailleurs, les protocoles induits par les communications basés sur index sont un compromis entre la sauvegarde coordonnée et la sauvegarde non coordonné. Afin de tirer profit de ces deux mécanismes, nous avons proposé un algorithme qui joint l'avantage des protocoles coordonnés et les protocoles basés index et qui s'adapte au mieux à son contexte d'exécution.

Les avantages de notre algorithme peuvent être résumés comme suit :

- Assurer un point de reprise global cohérent.
- Seulement deux points de reprises sont sauvegardés pour chaque site.
- Aucun site mobile ne doit être bloqué pendant l'exécution de l'algorithme.
- Ne consomme pas beaucoup d'énergie.
- N'encombre pas le réseau avec trop de message de coordination (économie de la bande passante).
- N'engendre pas de surcoût de recherche des *MHs*.
- Un coût minimum d'exécution.
- Supporte la déconnexion ou le déplacement d'un site mobile.

Comme perspective, plusieurs axes sont à poursuivre. A court terme, il serait intéressant de faire une simulation de l'algorithme proposé afin de pouvoir déterminer la durée de temps T_{Lazy}, qui optimise au mieux le protocole. Ensuite, à long terme, il est également intéressant de proposer d'autres stratégies de tolérance aux fautes qui s'adaptent aux réseaux mobiles ad hoc. Enfin, il convient d'étudier le passage à l'échelle dans un MANET étendu.

Bibliographie

[1] J-C. Laprie.ARAGO15 : «Concepts de base de la tolérance aux fautes ». MASSON, Paris, 1994.

[2] A. Avizienis, J-C. Laprie, and B. Randell. «Dependability and its threats- a taxonomy». In IFIP Congress Topical Sessions, pages 91–120, 2004.

[3] D. Conan. « Tolérance aux fautes par recouvrement arrière dans les systèmes informatiques répartis ». Thèse de Phd, informatique, Université PARIS 6, 1996.

[4] A. Avizienis. «Toward systematic design of fault-tolerant systems ». Computer, 30(4):51-58, 1997.

[5] D. Powell. «Failure mode assumption coverage ». In 22th IEEE Sumposium on Fault tolerant Computing, 1992.

[6] R. D. Schlichting and F. B. Schneider, « Fail-stop processors : An approach to designing fault tolerant distributed computing systems » ACM Trans, on Computer Systems, Vol. 1, No. 3, pp. 222-238, Aug. 1983.

[7] Avizienis, A. & Kelly, J. P. J. « Fault Tolerance by Design Diversity: Concepts and Experiments ». IEEE Computer, 17(8):67.80. 1984

[8] Dhiraj K. Pradhan : «Fault Tolerant Computer System Design », Prentice Hall, 2000.

[9] F.C. Gärtner. « Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments ». ACM Computing Surveys, 31(1) :1–26, March 1999.

[10] A.S.Tanenbaum et M.Steen. « Distributed Systems: Principles and Paradigms ». Prentice Hall, 2002.

[11] F. B. Schneider. « Implementing fault-tolerant services using the state machine approach: a tutorial». ACM Comput. Surv., 22(4) :299–319, 1990.

[12] K. R. Mazouni. « Etude de l'invocation entre objets dupliqués dans un système tolérant aux fautes ». Technical report, 1996.

[13] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. « A survey of rollback recovery protocols in message-passing systems ». ACM Comput. Surv., 34(3) :375–408, 2002.

[14] K.M Chandy and Lamport distributed snapshots: determining global states of distributed systems. In ACM Transactions on Computer Systems, p 63-75,1985.

[15] Ivan Stojmenovic. Handbook of Wireless Networks And Mobil Computing. Qing-An Zeng and Dharma P. Agrawal. Handoff in Wirless Mobile Networks. John Wiley & Sons. 2002.

[16] R. Strom and S. Yemini. « Optimistic recovery in distributed systems ». ACM Trans. Comput. Syst., 3(3):204–226, 1985.

[17] B. Randell. « System Structure for Software Fault Tolerance ». IEEE Trans. On Software Engineering, 1(1) : 220{232, 1975.

[18] Arup Acharya and B. R. Badrinath. « Checkpointing Distributed Applications on Mobile Computers ». the Third Intl. Conf. on Parallel and Distributed Information Systems. Sep 1994.

[19] D. K. Pradhan, P. Krishna and N. H. Vaidya. « Recoverable Mobile Environment : Design and Trade-off Analysis ». In Proceedings of the 26 th international Symposium on Fault-Tolerant Computing, (Sendai, Japan, June 1996), IEEE, PP. 16-25.

[20] N. Neves and W. K. Fucks. « Adaptive Recovery for Mobile Environnements ». Communication of the ACM, vol 40, no 1, PP 68-74, Junuary 1997.

[21] D. L.Russel. « State restoration in systems of communicating processing ». IEEE Trans, software. Eng, vol. SE-6, n°2, PP.183-194? March 1980.

[22] M. Aliouet. « Recovery in distributed system from solid fault ». Université de Constantine, Algerie, SAFECOM 92.

[23] R. Koo, S. Toueg. « Checkpointing and rollback recovery for distributed systems ». IEEE Trans. Software Eng, vol.SE-13,n]1, pp.23-31, January 1987.

[24] L.Alvisi, E.N. Elnozahy, S. Rao, S. A. Husain and A. D. Mel : « An Analysis of Communication-Induced Checkpointing ». In Digest of Papers, FTCS-29, The Twenty Nineth Annual International Symposium on Fault-Tolerant Computing (Madison, Wisconsin), 242-249. 1999.

[25] R. Baldoni. « A communication-induced checkpointing protocol that ensures rollback dependency trackability ». In Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS ’97), page 68. IEEE Computer Society, 1997.

[26] M.Elnozahy, L.Alvisi, Y.M.Wang and D.B. Johnson « a survey of rollback_recovery protocols in message passing systems ». Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, October 1996.

[27] A.Agbaria and W. H. Sanders. « Distributed Snapshots for Mobile Computing Systems ». Procedings of the Second IEEE Annual Conference on Pervasive Computing and Computing and Communications (PERCOM’04) 0-7695-2090-1/04.2004.

[28] P. Kumar, L. Kumar, R. K. Chauhan and V. K. Gupta. « A non-intrusive minimum process synchronous checkpointing protocol for mobile distributed systems ». Personal Wireless Communications, 2005. ICPWC 2005. 2005 IEEE International Conference on 23-25 Jan. 2005 Page(s):491-495.

[29] Guohui Li, LihChyun Shu, « A Low-Latency Checkpointing Scheme for Mobile Computing Systems ». Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International.Volume 1, 26-28 July 2005 Page(s):491 - 496 Vol. 2 Digital Object Identifier 10.1109/COMPSAC.2005.26.

[30] Tantikul, T.; Manivannan, D. « A Communication-Induced Checkpointing and Asynchronous Recovery Protocol for Mobile Computing Systems ». Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on 05-08 Dec. 2005 Page(s):70 - 74 Digital Object Identifier 10.1109/PDCAT.2005.

[31] D. B. Johnson and W. Zwaenepoel : « Recovery in distributed systems using optimistic message logging and checkpointing ». J. Algorithms 11, 3, 462–491. 1990.

[32] R.H.B. Netzer and J. Xu. « Necessary and sufficient conditions for consistent global snapshots ». IEEE Transactions on Parallel and Distributed Systems, 6(2) :165, 1995.

[33] Y. M. Wang. « Reducing message logging overhead for log-based recovery ». In Proceedings of the IEEE International Symposium on Circuits and Systems, 1993.

[34] Elnozahy, E.N, Johnson D.B and w. Zwaenepoel , « the performance of consistent checkpointing ». Proc of the 11th symposium of reliable distributed systems, p39-47, Houston (TX), October 1992.

[35] D. Ely W.J. Bolosky, J.R. Douceur and M. Theimer. « Faisability of a serverless distributed file system deployed on an existing set of desktop pcs ». In ACM international conferenceon Measurement and modeling of computer systems, SIGMETRICS, pages 34–43, 2000.

[36] Linux Networx. http://www.linuxnetworx.com/.

[37] K. Marzullo L. Alvisi. Message logging : « Pessimistic, optimistic, causal, and optimal ». IEEE Transactions on Software Engineering, 24(2) :149–159, 1998.

[38] T.H. Lai and T.H. Yang. « On distributed snapshots ». Information Processing Letters, 25, May 1987.

[39] Samir Jafar, « Programmation des systèmes parallèles distribués : tolérance aux pannes, résilience et adaptabilité » Thèse de Doctorat de l'INPG, juin 2006.

[40] M. Satyanarayanan, « Fundamental Challenges in Mobile Computing », Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96), p.1.7,Philadelphia,Pennsylvania,USA,mai1996.
http://www.cs.cmu.edu/afs/cs/project/coda-www/ResearchWebPages/docdir/podc95.ps.gz.

[41] Y. Saygin, Ö. Ulusoy et A. K. Elmagarmid, « Association Rules for Supporting Hoarding in Mobile Computing Environments », Proceedings of the 10th International

[42] T. Imielinski et H. F. Korth (éd.), Mobile Computing, The Kluwer International Series in Engineering and Computer Science, vol. 353, Kluwer Academic Publishers,février 1996, http ://www.wkap.nl.

[43] Casio, Home Page, http ://www.casio.com.

[44] Dell, Home Page, http ://www.dell.com.

[45] IBM, Home Page, http ://www.ibm.com.

[46] Psion, home page, http://www.psion.com

[47] Infrared Data Association, Link Management Protocol, janvier 1996, statut : «Version 1.1 », http ://www.irda.org/standards/pubs/IrData.zip. 1.2.2

[48] Infrared Data Association, Serial Infrared Link Access Protocol (Ir-LAP), juin 1996, statut :«Version 1.1, complément et errata du 5 janvier 1999», http://www.irda.org/standards/pubs/IrData.zip. 1.2.2

[49] Infrared Data Association, IrDA Control Specification (Formerly IrBus) IrDA CIR (Control IR) Standard, juin 1998, statut : « Version 1.0, errata du 26 octobre 1999», http://www.irda.org/standards/pubs/Irda_ControlV1p0E.zip.

[50] Infrared Data Association, Serial Infrared Physical Layer Speci_cation, mai 2001, statut :« Version 1.4 », http ://www.irda.org/standards/pubs/IrData.zip

[51] Spectrix Corporation, Home Page, http ://www.spectrixcorp.com.

[52] C.E Perkins, « Ad hoc Networking », Addison Wesley, ISBN 0-201-30976-9, 2000.

[53] M.E. Streens Trup, « Routing in communications networks ». Prentice Hall International Editions, Annexes, 1995.

[54] Handspring, Home Page, http ://www.handspring.com

[55] Palm, Home Page, http ://www.palm.com.

[56] A. HARTER et A. HOPPER, « A Distributed Location System for the Active Office», IEEE Network,vol.8, n°1, janvier 1994, p. 62.70, ftp.orl.co.uk:/pub/docs/ORL/tr.94.1.ps.Z.

[57] A. S. Helal, B. Haskell, J. L. Carter, R. Brice, D. Woelk et M. Rusinkiewicz, « Any Time, Anywhere Computing », Kluwer Academic Publishers, 1999, http://www.wkap.nl

[58] G. Le Grand, « Qualité de service dans les environnements Internet mobiles », Thèse de doctorat de l'université de Paris VI, 2001.

[59] Mounir Achir , « technologies basse consommation pour les réseaux ad hoc » Thèse de doctorat de l'Institut National Polytechnique de Grenoble, 2005.

[60] P. Jacquet, A. Laouiti, P. Minet, and L. Viennot, « Performance of multipoint relaying in ad hoc mobile routing protocols ». in Networking 2002, Pise, Italie, 2002.

[61] C. E. Perkins, « Ad hoc Networking » Addison Wersley, ISBN 0201-30976-9, 2000.

[62] M. E. Streenstrup « Routing in communication networks » Prentice Hall International Editions Annexes 1995.

[63] Arup Acharya and B. R. Badrinath. « Checkpointing Distributed Applications on Mobile Computers ». the Third Intl. Conf. on Parallel and Distributed Information Systems. Sep 1994.

[64] P.Krishna, N. H. Vaiday and D. K. Pradhan. « Recovery in Distributed Mobile Environments ». IEEE Workshop on advances in Parallel and Distributed Systems. Oct 1993.

[65] R. Prakash and M. Singhal. « Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems ». IEEE Trans. Parallel and Distributed Systems, PP. 344-351, Oct 1996.

[66] G.Cao and M.Singhal. « On Coordinated Checkpointing in Distributed Computing Systems ». IEEE Trans. Parallel and Distributed. PP. 1213-1225. Dec 1998.

[67] N. Neves and W. K. Fucks. Adaptive Recovery for Mobile Environnements. Communication of the ACM, vol 40, no 1, PP 68-74, Junuary 1997.

[68] G. Cao and M. Singhal. « Low-Cost Checkpointing with Mutable Checkpoints in Mobile Computing Systems ». Proc 18 th Intl Conf. Distributed Computing Systems, PP 464-471, May 1998.

[69] G. Cao and M. Singhal. « On the Impossibility of Min-Process Non-Blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems ». Proc 27 th Intl Conf. on Parallel Processing, PP. 37-44, Aug 1998.

[70] H. Higaki and M. Takizawa. « Checkpoint-Recovery Protocol for Reliable Mobile Systems ». The 17 th International Symposium on Reliable Distributed Systems, pp. 93-99. 1998.

[71] Cheng-Min Lin et Chyi-Ren Dow. « Efficient Checkpoint-Based Failure Recovery Technique in Mobile Computing Systems ». Journal of International Science and Engineering, vol 17, N°4, p549-573,September 2001.

[72] Pushpendra Singh et Gilbert Cabillic. « Successive Checkpointing Approch for Mobile Computing Environment ». International Conference on Wireless Networks.2003.

[73] Sapna E. George, Ing-Ray et Chen Ying Jin. « Movement-Based Checkpointing and Logging for Recovery in Mobile Computing Systems » Proceedings of 5th ACM International Workshop on Data Engineerin for Wireless and Mobile Access, p51-58, ISBN1-59593-436-7,2006.

[74] Bidyut Gupta and Shahram Rahimi. « A Novel Roll-Back Mechanism for Performance Enhancement of Asynchronous Checkpointing and Recovery ». Informatica Journal, vol 31, N°1,2007.

[75] Sangho Yi, J. Heo, Y. Cho, and J. Hong, « Adaptive Mobile Checkpointing Facility for Wireless Sensor Networks ». 2006

[76] P. Lemarinier, A. Bouteiller, T. Herault, G. Krawezik, and F. Cappello. « Improved message logging versus improved coordinated checkpointing for fault tolerant mpi ». In Cluster '04 : Proceedings of the 2004 IEEE International Conference on Cluster Computing, pages 115–124, Washington, DC, USA, 2004. IEEE Computer Society.

Résumé

La large utilisation des ordinateurs portables et les progrès technologiques croissants dans les réseaux sans fil ont fait des applications mobiles une réalité qui confère à l'informatique ubiquitaire : *le calcul partout et à tout moment*. Ces dernières années ont vu le déploiement des infrastructures pervasives, à base de réseaux mobiles ad hoc (MANET). Ces derniers ne comportent ni nœud privilégié, ni infrastructure de contrôle centralisée, ils sont utilisés en cas de sinistre (tremblement de terre, incendie, sauvetage..) et peuvent être utiles pour des applications spécifiques aussi bien dans le domaine militaire que civil.

Cette flexibilité pose en revanche de nouveaux défis liés aux spécificités particulières de ce type de réseaux telles que : fréquentes déconnexions, mobilité, ressources limitées des terminaux, et la vulnérabilité face aux défaillances.

Assurer une continuité de service malgré l'occurrence d'une erreur, est un problème complexe vu la dynamicité et l'évolution rapide de la topologie. Pour cela on a recours aux techniques de tolérance aux fautes. Plus précisément : le recouvrement arrière (ou reprise).

Le travail présenté dans ce mémoire consiste à analyser l'état de l'art et proposer un algorithme de capture de points de reprise dans les applications réparties, s'exécutant sur un MANET et prenant en compte de manière objective leurs caractéristiques spécifiques.

Mots clés : Applications Parallèles, Environnement Mobile, Réseaux Mobiles Ad hoc, Checkpointing, Tolérance aux Fautes, Sûreté de Fonctionnement.

Abstract

The wide use of laptops and the technological progress in the wireless networks made mobile applications a true reality in the ubiquitous computing environment: indeed, *the calculation everywhere and at any time*. Over these last years, the deployment of pervasive infrastructures based on mobile ad hoc networks (MANET) has paid much more attention. These infrastructures comprised neither privileged node, nor centralized decision of control; they are used in situations of disaster (earthquake, fire, rescue.) and can be useful for specific applications as well in the military and civil field.

This flexibility poses on the other hand new challenges related to specificities of the mobile networks such as: frequent disconnections, mobility, restricted resources of the mobile terminals, and vulnerability faced with the failure occurrences.

Ensuring a continuity of service in spite of error occurrences is a complex problem when considering the dynamicity and fast change of topology. For facing this undesirable situation, we usually have recourse to the fault tolerance techniques. More precisely: rollback recovery mechanism.

The work presented in this thesis consists in a survey of state of the art and a checkpointing algorithm proposal in the distributed applications, being carried out on a MANET and taking objectively into account their specific characteristics.

Key-words: Parallel Applications, Mobile Environment, Mobile Ad hoc networks, Checkpointing, Fault Tolerance, Reliability.