REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère De l'Enseignement Supérieur

Université des **S**ciences et de la **T**echnologie **H**ouari **B**oumediene

Faculté Génie Electronique & Informatique

***D*épartement d'*I*nformatique**

***L*aboratoire des *S*ystèmes *I*nformatique**

*T*HESE DE *M*AGISTERE

Option: ***Programmation*** **et** ***Systèmes***

Thème:

Points de Contrôle dans les Systèmes Répartis en Environnement Mobile

Présenté par :

M. Haroun BENKAOUHA.

Directeur de Thèse :

D. Nadjib Badache.

Soutenue devant le jury :

Mme.	**M.** Boukala	**Maître de Conférences à l'USTHB**	**Présidente**
M.	**S.** LARABI	**Maître de Conférences à l'USTHB**	**Examinateur**

M.	**H.** Azoun	**Chargé de Cours à l'USTHB**	**Examinateur**
M.	**N.** Badache	**Maître de Conférences à l'USTHB**	**Rapporteur**

Juin 2003.

Résumé

Dans un *système réparti*, l'étude de l'état du système est primordiale dans le but de résoudre une classe très large de problèmes : *Détection des propriétés stables, Détermination des breakpoints, Recouvrement arrière lors des défaillances, Détection des propriétés instables ....*

Un *système de calcul mobile* est composé de nœuds statiques et mobiles interconnectés par un réseau de communication. La mobilité introduit d'autres contraintes : *fréquentes déconnexions, débit de transfert limité, la source d'énergie limitée, ressources utilisées restreintes* en plus du problème de *localisation du site.*

Le *checkpointing* est la détermination des *points de contrôle globaux cohérents* du système distribué. Il s'agit de l'ensemble de *points de contrôle locaux* (un par processus) qui représente l'*état local* d'un processus. La *cohérence* d'un point de contrôle global définit un état global que le calcul a atteint ou aurait pu atteindre.

Ce travail entre dans le domaine de la *Tolérance aux défaillances*. L'objectif est de concevoir *un algorithme de checkpointing* pour les systèmes répartis en tenant compte des *contraintes de mobilité* de certains processus.

Mots clés: Calcul Distribué, Calcul Mobile, Checkpointing, Cohérence, Mobilité, Point de Contrôle Global, Point de Reprise, Tolérance aux pannes.

Abstract

In a *distributed system*, a study of the system state is crucial in order to resolve a large class of problems : *stable properties detection, breakpoints determination, rollback recovery during default, nonstable properties detection …*

Mobile *computation system consists* of static and mobile nodes interconnected by a communication network. The mobility introduce to distributed system another constraints : *frequent disconnections, low bandwidth, weak energy source, restricted used resources*, added to this, the problem of a *site localization.*

The *Checkpointing* is the determination of a *consistent global snapshot* of the distributed system. It is question of the set of *local snapshots* (one by process) that represent a *local state* of a process. The *consistency* of a global snapshot describes a global state reached or can be reached by the computing.

This work is in the domain of *Fault-Tolerant*. The objective is to design *an algorithm of checkpointing* for distributed system taking into account *mobility constraints* of some processes.

Keywords: Distributed Computing, Mobile Computing, Checkpointing, Consistency, Mobility, Global Checkpoint, Snapshot, Fault Tolerant.

Remerciements

Grâce au bon Dieu tout puissant que ce modeste travail a été réalisé. Dieu Merci, qui m'a donné courage pour aller jusqu'au bout. Hamdoullah.

Je tiens à remercier Madame M. BOUKALA, Maître de Conférences à l'U.S.T.H.B. et Directrice du Département Informatique de m'avoir fait l'honneur de présider le jury de cette thèse.

Je remercie Messieurs S. LARABI et Maître de Conférences à l'U.S.T.H.B. et M. H. AZOUN chargé de cours à l'U.S.T.H.B. d'avoir accepté de participer au jury et d'examiner ce travails.

Je remercie mon Directeur de thèse, Docteur Nadjib BADACHE, Maître de Conférence à l'USTHB et directeur du Laboratoire des Systèmes Informatique, pour ses conseils, orientations et sa confiance.

Tous mes remerciements vont aussi à tous ceux qui m'ont aidé et soutenu dans tout le parcours.

Dédicaces

A la mémoire de mon père Mohamed BENKAOUHA assassiné le 22 Mars 1995 que je n'oublierai jamais ;

A ma mère qui a été tout le temps à mes cotés ;

A mes frères et sœur ;

A mes neveux et nièces ;

A tous ceux qui m'aiment…

Je dédie ce modeste travail.

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

INTRODUCTION

Un *système distribué* (*réparti*) est composé d'un ensemble fini de *sites* interconnectés par un *réseau* de *communication* implémentant des *canaux bipoints* [BM,1993][HMR,1998]. Un *système distribué* est dit *asynchrone*, s'il n'y a pas d'*hypothèses* particulières sur le *temps physique* [HMR,1998]. Ceci permet une modélisation plus fidèle de la réalité. La *synchronisation* est réalisée à l'aide de la communication, en échangeant uniquement des *messages* [HNR,1998][BHMR,1997] [HMR,1997]. Ce système est appelé aussi système à passage de messages (*message passing system*) [Lin,2001].

Le *calcul distribué* (ou *exécution distribuée asynchrone*) décrit l'exécution d'un *programme réparti* par une collection de processus [BM,1993]. Dans un souci de simplification et sans perte de généralité, nous considérons qu'à tout processeur est associé un et un seul processus [BM,1993][HMR,1998]. L'activité de chaque processus est vue comme l'*exécution séquentielle* d'évènements. Les trois principaux événements sont : l'*émission d'un message*, la *réception d'un message* et l'*exécution d'instructions internes* (*événements internes*). Un *algorithme distribué* permet d'implémenter les trois événements décrits ci-dessus.

Les *systèmes distribués asynchrones* sont caractérisés par une large classe de problèmes tels que la *détection des propriétés stables*, *la détermination des points d'arrêt (breakpoints)*, *le recouvrement arrière (rollback rocovery)* lors des défaillances, *la détection des propriétés-instables*… Pour résoudre ces problèmes, il est primordial de connaître l'*état du système*. Notre travail s'inscrit dans le domaine de la *tolérance aux pannes* (*défaillances*) ou en anglais *Fault Tolerant* et il s'agit plus particulièrement de la *détermination* des *points de contrôle globaux* (*global snapshots* / *global checkpoints*), appelé aussi le *checkpointing*.

Déterminer des *points de contrôle*, appelés dans notre cas *points de reprise* (*checkpoint*), est une *tâche très importante* pour le système. Ces points sont appelés, dans les applications au domaine de la détection de propriétés plus puissantes des exécutions, *états* ou *coupes* [HMR,1998]. Nous utiliserons dans ce document les termes *point de reprise*, *point de contrôle* ou *checkpoint* de manière interchangeable et qui auront le même sens que celui du domaine de la tolérance aux pannes.

Un *point de contrôle global* (*snapshot*) dans un système distribué est un *ensemble de points de contrôle locaux* (un par processus). Le point de contrôle local

d'un processus qui participe au calcul distribué est un de ses états locaux intermédiaires enregistrés.

Un *point de contrôle global* est dit *cohérent* ou *consistant* si aucun message dans l'état enregistré n'est *orphelin*. En d'autres termes, un point de reprise global n'est *cohérent* que si dans ses états locaux tous les *messages reçus* ont été *envoyés* dans d'autres points locaux faisant partie du point de contrôle global. La *cohérence* d'un *point de contrôle global* définit un *état global* par lequel le calcul est passé ou aurait pu passer.

Ainsi l'étude de la cohérence se fait sur une abstraction de l'exécution répartie qui ne prend en considération que les points de contrôle et leur relation de dépendance, tous les états locaux qui ne sont pas des points de contrôle sont ignorés.

Le *checkpointing* est une technique importante pour *minimiser la perte de calcul* dans un *environnement sujet aux défaillances*. Nous portons un intérêt à ces techniques dans un environnement fragile qui est l'*environnement mobile*. Un processus mobile dénote un processus s'exécutant sur une station en mouvement (véhiculé sur terre, navire ou avion) et utilisant un moyen de communication sans fil. Donc, il n'est pas astreint à une localisation fixe tout en restant connecté au réseau.

Cette mobilité rend le système plus sensible aux défaillances du fait qu'elle introduit de nouvelles contraintes qu'il faut prendre en charge lors de l'étude du système [BAI,1993a]. Les stations mobiles sont vulnérables aux pannes comme les pertes ou les problèmes physiques. Les déconnexions deviennent de plus en plus fréquentes à cause du changement de localisation (*handoff*) ou à d'autres problèmes. Ainsi, une station mobile peut être déconnectée pour une période arbitraire du réseau ou de façon permanente sans prévenir. La bande passante est, dans ce cas, limitée. Les ressources machine (mémoire, disque dur, vitesse…) sont restreintes. La localisation des stations mobiles est aussi un grand problème qui fait augmenter la complexité et les délais de transmission des messages. Pour ce, plusieurs protocoles de routage ont été proposés. La source d'énergie qui est dans notre cas, la batterie a une durée de vie limitée. Pour économiser l'énergie la station mobile peut mettre à l'arrêt certains composants durant la période de basse activité. Pour tout ceci, une station mobile est considérée comme *un support non sûr pour stocker les données de façon stable*. De plus l'environnement mobile offre une grande flexibilité dans le design du réseau, un déploiement facile et rapide. Par conséquent, les structures logiques (anneau, arbre, …), exploités traditionnellement par les algorithmes distribués ne peuvent plus être utilisées efficacement dans un environnement où les sites changent de place fréquemment [Bad,1995].

Un système distribué en environnement mobile sera défini comme étant un ensemble de processus communiquant par échange de messages (leur nombre n'est pas obligatoirement connu et peut changer). Certains de ces processus (la majorité) s'exécutent sur des stations mobiles (notées *MH* : *mobile hosts*) et d'autres sur des stations statiques. Les stations statiques, appelées aussi stations de support aux mobiles (notées *MSS* : *mobile support station*) seront des points d'accès pour les

sites mobiles à travers des lignes de communication sans fils. Toutes les autres caractéristiques et contraintes des systèmes répartis restent en vigueur.

Les *techniques de checkpointing* dans les systèmes répartis conventionnels sont classées en deux grandes catégories : le *checkpointing coordonné* (*coordinated checkpointing*) et le *checkpointing non coordonné* (*uncoordinated checkpointing*). Pour la première, deux grandes orientations dans ce domaine existent, celles qui oeuvrent pour faire participer un nombre minimum de processus à prendre leurs checkpoints et celles qui évitent de bloquer le calcul sous-jacent durant le déroulement de l'algorithme de checkpointing. Il a été prouvé qu'il est impossible de réaliser un algorithme qui tient compte ces deux paramètres [CS,1998]. Pour la seconde technique, les processus prennent indépendamment leurs points de contrôle locaux sans utiliser des messages de coordination.

Pour le cas mobile, de nouvelles techniques on été introduites comme les algorithmes *hybrides* [HT,1998], les algorithmes *quasi-synchrones* [MS,1996] ou la notion de *mutable checkpoint* [CS,2001].

Nous proposerons un algorithme de checkpointing qui rentre dans les techniques de checkpointing hybrides : Checkpointing coordonnées non bloquants (il a été montré que les algorithmes bloquants réduisent considérablement les performances du système ce qui est un très grand inconvénient surtout dans le cas mobile [PS,1996] et [CS,2001]) et le checkpointing indépendant basé index en envoyant certaines informations du checkpointing dans les messages de calcul pour permettre à certains processus de prendre des checkpoints additionnels et pour déranger le moins possible les stations mobiles en mode veille. La réalisation du checkpointing se fait en deux phases. La première consiste à enregistrer localement les checkpoints de façon indépendante (point de contrôle spontané) ou sur la base d'informations reçues avec un message de calcul (point de contrôle forcé). Nous utilisons dans cette phase une notion qui se rapproche des points de contrôle mutables définie dans [CS,2001] (n'est ni une tentative de point de contrôle ni un point de contrôle permanent et qui peut être sauvegardé n'importe où et pas nécessairement sur un support stable) dans le but d'éviter le transfert des informations vers les MSS et économiser la bande passante. La deuxième phase consiste à coordonner les processus pour enregistrer les points de contrôle sur un support de stockage stable. Cette phase est architecturée en deux niveaux, pour que la partie fixe du réseau supporte la grande partie du surcoût du checkpointing. Nous essayerons de limiter au maximum la coordination uniquement entre les stations fixes.

Ce document est organisé comme suit : Au début (chapitre `I`), nous commencerons par présenter les systèmes distribués et toutes les notions liées à cet environnement. Ensuite(chapitre `II`), nous parlerons du calcul réparti dans un environnement mobile, nous étudierons les contraintes de la mobilité et les conséquences de cette dernière sur le système. Dans le chapitre `III`, nous discuterons le checkpointing dans les systèmes distribués classiques (conventionnels), nous verrons leur classification et nous analyserons les principaux travaux réalisés dans ce domaine. Puis dans les chapitre `IV` et `V`, nous discuterons

les points de contrôle dans l'environnement mobile, nous présenterons et discuterons aussi les principaux algorithmes. Nous présenterons, dans le chapitre VI, notre contribution ainsi que les preuves nécessaires. Nous terminerons ce document par une conclusion générale. En annexe, une étude sur les différentes techniques de communication mobile et l'architecture des réseaux sans fil sont présentés.

CHAPITRE I :

LES SYSTEMES REPARTIS

1. INTRODUCTION

On appelle *système distribué* ou *réparti* (*distributed system*) ou aussi *système à passage de messages* (*message passing system* [Lin,2001]) tout ensemble de processus exécutant un calcul commun et qui ne communiquent que sur échange de messages à travers des canaux. L'objectif de ce chapitre est de donner les notions principales des systèmes répartis et du calcul distribué et son exécution.

2. DEFINITION

Un système distribué est un ensemble de sites S_1,…,S_n et un réseau de communication capable d'implémenter des canaux unidirectionnels entre pairs de processus pour leur permettre de communiquer par échange de messages. Les canaux de communication représentent le réseau de transport (figure 1.1).

Le système peut comporter certaines hypothèses comme par exemple sur la fiabilité des canaux de communication, l'ordre des messages (délivrés dans l'ordre –canaux FIFO- ou dans le désordre –canaux non FIFO-), l'altération, la perte ou le dé-séquencement des messages.

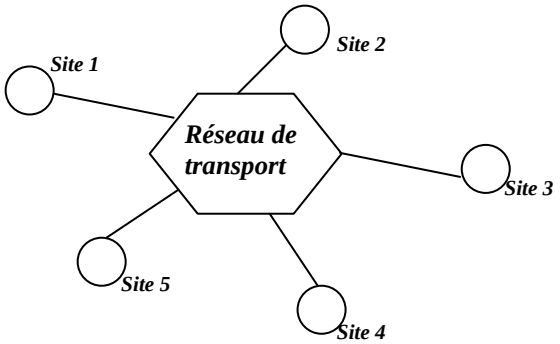


Figure 1.1 : Système réparti (Sites et Réseau de transport)

Dans un système réel, il peut y avoir plusieurs processus qui s'exécutent sur le même site. Comme le contrôle des processus s'exécutant sur la même machine (même site) relève du système centralisé, nous supposerons dans un but de simplification et sans perte de généralité, que chaque site est représenté par un seul processus. Donc un système réparti est vu comme *n* processus $P_1,\ldots,P_n$.

Chaque processus peut communiquer avec tout autre processus du système même à travers un autre processus (Système fortement connecté mais pas complètement). Le système ne comporte ni horloge globale, ni mémoire partagée.

3. SYSTEME REPARTI ASYNCHRONE

Un système distribué asynchrone est tout système distribué qui ne comporte aucune hypothèse particulière sur le temps physique [HMR,1998]. Toute idée d'un processus qui détient une horloge locale synchronisée ou un raisonnement basé sur le temps réel global sont éliminés [BM,1993]. Parmi ces hypothèses, nous citerons : Les vitesses relatives des processeurs et les durées de transfert des messages sont supposées quelconques.

L'absence d'hypothèses permet une modélisation plus fidèle de la réalité. Ceci marque la différence avec les autres systèmes comme les systèmes parallèles à mémoire partagée ou à horloge globale ou tout système synchrone où l'évolution est réglée par un dispositif qui produit des pulsations globales [Mos,1994].

Toute synchronisation se fait à travers l'échange de messages. Ainsi le système peut être vu comme un graphe quelconque où les sommets représentent les processus et les arcs les canaux de communication [Mos,1994]. Ce graphe est appelé graphe de communication (figure 1.2).

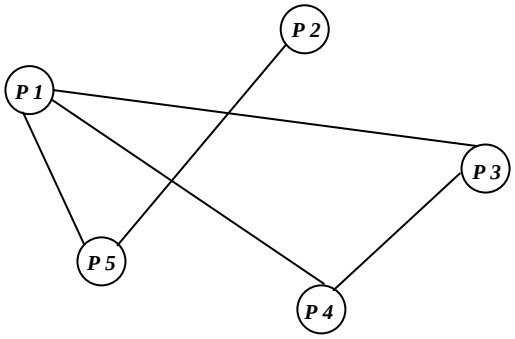


Figure 1.2 : Graphe de communication

Les systèmes distribués asynchrones représentent le plus complexe modèle possible [BM,1993]. L'intérêt de se mettre dans ces conditions pour étudier le système et résoudre certains problèmes est de minimiser le coût. En effet, la même solution sur un modèle moins contraignant revient moins chère.

4. STRUCTURES LOGIQUES

L'étude du système réparti nécessite à tout site d'adresser ou citer un processus sans ambiguïté. Ainsi chaque site ou processus doit avoir une identité unique. Le système réel d'identification des processus et des sites étant assez complexe et pour plus de clarté de présentation, les sites sont supposés être numérotés de *1* à *n* où *n* est le nombre de sites du système [Mos,1994] et donc désignés comme indiqué ci-dessus $P_1,\ldots,P_n$.

Il arrive qu'il soit nécessaire de construire une structure virtuelle au-dessus du graphe des communications, soit pour la facilité de conception soit pour des impératifs de contrôle et répartition. Plusieurs structurations différentes peuvent coexister à un même instant au sein d'un même système réparti pour supporter chacune une application ou un algorithme de contrôle donné. Une structuration définit pour un processus donné l'ensemble de processus avec lesquels il peut communiquer ainsi que la relation qui le lie à chacun d'eux. Parmi les structures logiques on peut citer : l'anneau, l'étoile, l'arbre, la grille …

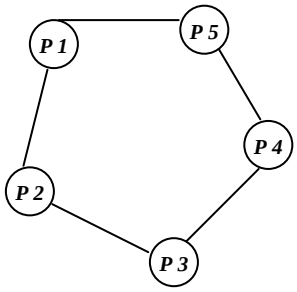


Figure 1.3 : Structure logique en Anneau

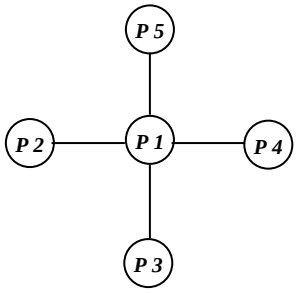


Figure 1.4 : Structure logique en Etoile

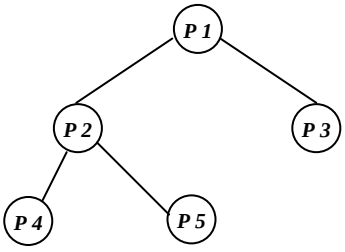


Figure 1.5 : Structure logique en Arbre

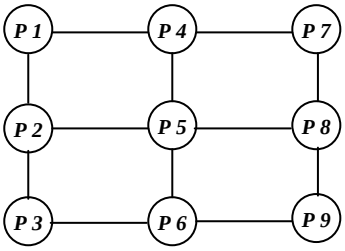


Figure 1.6 : Structure logique en Grille

5. LES DEFAILLANCES

Les composants d'un système réparti sont sujet à des défaillances (pannes). On dit qu'il y a défaillance d'un composant (physique ou logiciel) lorsque le comportement de celui-ci dévie de sa spécification. Dans la situation idéale, après défaillance, le système comme entité unique doit continuer à fonctionner le mieux possible (fonctionnement en mode dégradé). Les fonctions assurées par des composants "corrects" doivent continuer à être assurées de manière correcte.

Le caractère non prévisible des défaillances ainsi que la possibilité de défaillances multiples (une défaillance peut survenir pendant l'exécution de la procédure de recouvrement d'une autre défaillance) fait de la conception de systèmes répartis tolérants aux pannes un problème non trivial et souvent sans solution. [Mos,1994]

Les défaillances dans un système sont diverses et variées de par leur origine, leur persistance et leur nature. Une classification détaillée ainsi que la terminologie peut être trouvée dans [Lap,1991]. A notre niveau d'abstraction, on peut citer les pannes qui sont liées soit au canal de communication soit au site.

5.1 DEFAILLANCE D'UN CANAL

Un canal de communication non fiable peut perdre, altérer, dé-séquencer ou dupliquer les messages. Pour les deux derniers il suffit seulement de numéroter les messages. En cas de perte, il faut envoyer de nouveau le message si le canal n'est pas coupé.

5.2 FAUTES BYZANTINES

Terme issu du problème des généraux byzantins [LSP,1982]. Ce sont des fautes difficiles à traiter et dues à des comportements arbitraires de certains composants. Le site ou le canal peut exhiber un comportement incorrect à l'insu des autres composants comme l'altération non détectée d'un message ou l'émission de messages vers des sites qui ne sont pas destinataires. On distingue deux types de ces pannes :

- Fautes byzantines "*naturelles*" comme par exemple une erreur physique non détectée (sur une transmission de message, en mémoire, sur une instruction …) ou une erreur logicielle amenant une non vérification des spécifications.

- Fautes byzantines "*malicieuses*" comme par exemple tout comportement visant à faire échouer le système (sabotage, virus ....).

Il est prouvé qu'il est impossible de trouver le moindre consensus si le site ou le canal a ce genre de comportements [LSP,1982]. Pour cela, les protocoles font des suppositions comme par exemple sur le nombre maximum de composants pouvant avoir ce genre de comportement ou le nombre de messages qu'un canal défaillant peut perdre consécutivement.

5.3 PROCESSEURS A ARRET SUR DEFAILLANCE

Ce sont des machines (appelées aussi fail-stop processors) qui mettent les processeurs à l'arrêt dès qu'une panne (faute) est détectée. Ce genre de machines n'existent pas physiquement car il est impossible de les réaliser. Par contre, il est possible de mettre en œuvre une abstraction qui a, avec une probabilité élevée, d'avoir un comportement de *fail-stop process*.

La notion de fail-stop représente le tout ou rien, c'est complètement l'opposé du comportement byzantin du fait que les défaillances sont bien caractérisées.

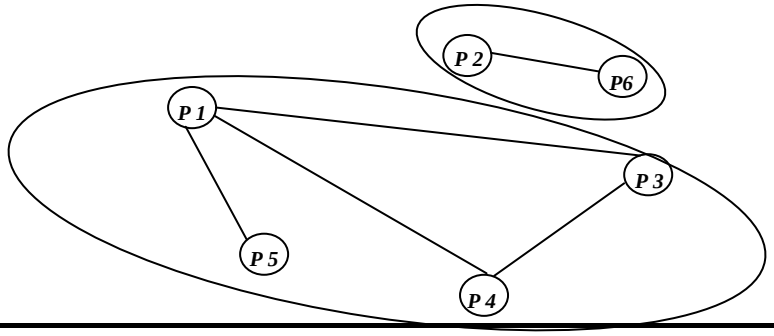
Lors de la défaillance, le fail-stop processor perd son état local et une partie de sa mémoire sauf la partie stable et ne fait jamais de transformations erronées de son état interne. La partie stable reste accessible pour les autres processus. Ainsi, les effets de la défaillance sont l'arrêt de l'exécution et la perte de l'état interne. Vous trouverez plus de détails sur ce genre de défaillance dans [SS,1983]. Ainsi, cette notion permet de simplifier la construction de la tolérance aux défaillances dans les systèmes de calcul distribués.

5.4 PARTITIONNEMENT

Suite à une panne (défaillance d'un canal de communication ou d'un site), le système peut se retrouver coupé en deux ou plusieurs parties n'ayant plus aucune possibilité de communiquer entre elles.

Toute partie, croyant être la seule à fonctionner, peut effectuer des traitements qui devraient se faire par exemple en exclusion mutuelle. Les différentes parties du système vont ainsi diverger et il en résultera un état incohérent après recouvrement du partitionnement. Ceci rendra ce problème parmi les plus durs à résoudre.

Figure 1.7 : Graphe de communication après Partitionnement



6. TECHNIQUES DE TOLERANCE

La sûreté de fonctionnement d'un système informatique est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre. Pour assurer la continuité du service il faut rendre le système tolérant aux pannes. Pour cela, il faut fournir au système les moyens de détection et de prise en compte des défaillances. Il existe des mécanismes et techniques simples comme : les points de reprise, le comptage, le time-out, la duplication, la diffusion atomique, le consensus et les quorums.

Il existe aussi d'autres mécanismes plus élaborés comme : la diffusion atomique, le consensus ainsi que les votes et quorums. Ci-dessous un bref aperçu sur ces différentes techniques.

6.1 POINTS DE REPRISES

Consiste à prendre régulièrement une image globale du système. Ces images, appelées aussi *points de contrôle*, *snapshot* ou *checkpoints*, permettent de reprendre l'exécution après l'arrêt causé par la défaillance. La reprise se fait à partir de la dernière image cohérente sauvegardé. Le grand problème est comment synchroniser tous les sites à prendre cette image alors que le système distribué est asynchrone. Donc, chaque site devra prendre sa propre image et la sauvegarder dans un endroit stable et accessible. Il faut trouver un moyen pour que l'image globale soit significative et représentative de l'un des états du système. Nous verrons par la suite en détail comment se fait cette prise et cette synchronisation.

Après la défaillance, chaque site reviendra en arrière pour reprendre son travail à partir de ce point là. Dans le cadre de notre travail, nous nous intéressons à cet aspect là et plus précisément pour le calcul de ces points et la façon de les enregistrer pour coordonner entre les différents sites dans le but d'obtenir un état global significatif. Nous allons voir comment trouver les checkpoints dans chaque processus, qui doivent représenter ensemble le plus récent état possible du système et au même temps doivent former un état par lequel est passé le système ou aurait pu passer sans influer sur le résultat final. Nous étudierons plus loin la notion de cohérence d'un checkpoint global.

6.2 COMPTAGE

Cette technique est utilisée pour tolérer certaines pannes ou problèmes des canaux de communication. La perte de messages et leur dé-séquencement peuvent être détectés très facilement en numérotant tous les messages émis sur chaque canal. Une rupture de séquence alerte le récepteur qui demandera alors la retransmission du message perdu.

Par exemple, un site P_i envoie deux message *m1* (numéroté 1) et ensuite *m2* (numéroté 2) vers un autre site P_j. Si P_j reçoit le message numéroté 2 (*m2*), il doit nécessairement avoir déjà reçu *m1* sinon il considère que le message *m1* comme perdu et demandera son émission de nouveau.

6.3 DUPLICATION

La duplication et la fragmentation des données sont utilisées pour pouvoir tolérer des défaillances dans un système. La lecture et l'écriture d'une donnée dupliquée restent possibles tant qu'il y a suffisamment de copies disponibles, ce qui préserve l'accessibilité.

Une autre approche, qui ressemble à celle là et qui est plus efficace, est basée sur la réplication de processus. Elle consiste à créer des copies multiples des processus sur des processeurs différents. La réplication peut être active, semi-active ou passive, des détails sur ces techniques peuvent être trouvés dans [Lap,1991].

6.4 DIFFUSION ATOMIQUE ET CONSENSUS

Pour qu'une diffusion soit atomique, il faut qu'au bout d'un temps fini, tous les sites reçoivent la même valeur (le message émis) ou personne ne la reçoit. Pour assurer la vivacité de la diffusion atomique, il faut utiliser un protocole de calcul de la liste de présence qui permet de restreindre le groupe aux sites corrects. Ceci limitera les attentes. Il peut donc arriver qu'un site soit exclu d'un groupe pour cause de lenteur et il devra exécuter une procédure de réadmission pour réintégrer le groupe.

Le consensus est un mécanisme qui permet de n'entreprendre que des actions préservant la cohérence du système. Il n'y a consensus entre les sites (d'un groupe donné) sur une valeur ou une action à accomplir que si tous les sites s'entendent sur la même valeur au bout d'un temps fini.

Les protocoles assurant la diffusion atomique permettent de résoudre le problème de la diffusion atomique et inversement. Pour les systèmes asynchrones ayant des comportements byzantins, ces problèmes n'ont pas de solution.

6.5 LES QUORUMS

Le mécanisme de quorums et son cas particulier le vote sont utilisés dans le problème d'exclusion mutuelle et la duplication. Il représente l'une des rares techniques résistant au problème du partitionnement du système.

Le mécanisme repose sur le nombre de permissions que le site doit obtenir pour accomplir son action. Un tel ensemble de permissions s'appelle un quorum. Ces ensembles sont définis de sorte qu'un site soit le seul à pouvoir exécuter son action. Par exemple, l'intersection de quorums associés à deux sites distincts est toujours non vide.

6.6 TIME-OUT

Dans certains modèles, cette technique est utilisée pour permettre de détecter les pannes sur le système distribué. Appelée aussi délai de garde, elle fixe une limite au temps d'attente d'une réponse ou d'un événement devant être réalisé par un autre site.

Par exemple, si les délais de transfert des messages sont bornés, le time-out est une bonne technique pour détecter les défaillances des sites et des canaux de communication.

Cette technique nécessite parfois de supposer que les horloges des processeurs sont synchronisées. Par exemple, si deux processus ne se mettent pas d'accord sur le fait qu'un troisième est à l'arrêt peut avoir des conséquences désastreuses.

7. CALCUL DISTRIBUE

De façon non formelle, nous pouvons définir un calcul distribué (réparti) comme une description de l'exécution d'un programme distribué par un ensemble de processus. L'activité de chaque processus est vue comme l'exécution d'une séquence d'évènements séquentiels. Si le système est asynchrone, le calcul réparti l'est aussi.

7.1 LES EVENEMENTS

Nous distinguons trois types d'évènements dans une exécution distribuée, des événements internes et deux types d'événements externes (l'émission et la réception de messages) permettant de synchroniser les processus :

- Les événements internes n'influent que sur le changement de l'état local.
- L'événement d'émission d'un message *m* : *send(m)* qui consiste à mettre le message *m* sur la file du canal de sortie vers le processus destinataire.
- L'événement de réception d'un message *m* : *receive(m)*. Le processus destinataire récupère le message à partir du canal d'entrée. Pour cet événement deux conditions sont nécessaires : tout d'abord il faut que le message arrive au processus destinataire et ce dernier doit déclarer son accord pour le recevoir.

Au niveau langage de programmation, la communication peut être accomplie à travers un certain nombre de paradigmes incluant [BM,1993] :

- Les appels de procédures à distance
- Les diffusions
- Les transactions distribuées
- Les objets distribués
- Mémoire commune distribuée

A notre niveau, tout ça se réduit à un simple échange de messages à travers les deux événements de *send* et *receive*.

7.2 HISTOIRE LOCALE D'UN PROCESSUS

L'histoire locale d'un processus P_i, notée h_i, durant le calcul est une séquence (qui peut être infinie) d'événements.

$h_i = e_i^1 e_i^2 \ldots e_i^n \ldots$

$e_{i,j}$ ou e_i^j : événement j du processus i.

$h_{i,k}$ ou h_i^k : représente les k premiers événements du processus i. h_i^0 est vide par définition.

L'histoire globale du calcul est l'ensemble $H=h_1 \cup h_2 \cup \ldots \cup h_n$. L'histoire globale ne spécifie aucune synchronisation ou ordre chronologique relatif entre les événements.

Dans un système distribué asynchrone, les événements de calcul peuvent être classés seulement dans un ordre basé sur la notion "cause – et – effet", appelée aussi relation de précédence causale notée par le symbole $\rightarrow$ et définie comme suit :

$$e_i^k, e_i^l \in h_i \;\; et \;\; k < l \;\; \Rightarrow \;\; e_i^k \rightarrow e_i^l$$
$$e_i^k = send(m) \;\; et \;\; e_j^l = receive(m) \;\; \Rightarrow \;\; e_i^k \rightarrow e_j^l$$
$$e \rightarrow e' \;\; et \;\; e' \rightarrow e'' \;\; \Rightarrow \;\; e \rightarrow e''$$

$e \rightarrow e' \Leftrightarrow$ *e* ***précède causalement*** *e'*.

Certains événements de l'histoire globale peuvent n'avoir aucun rapport de causalité. C'est à dire :

Ni $e \rightarrow e'$, ni $e' \rightarrow e \Rightarrow e$ et e' sont des événements concurrents (ou parallèles) et on écrit $e \,||\, e'$.

7.3 DEFINITION FORMELLE DU CALCUL DISTRIBUE

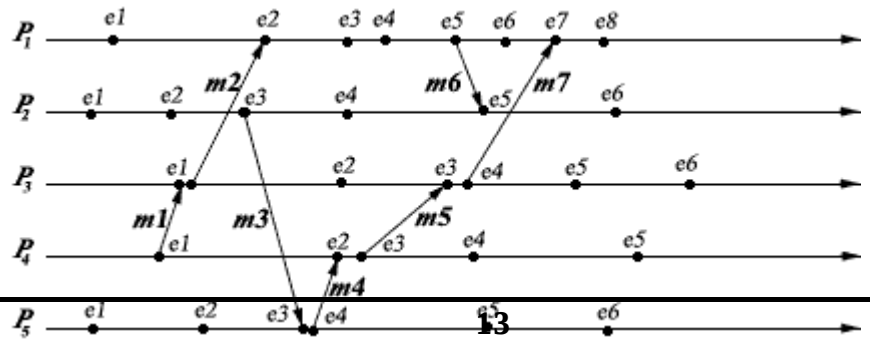
Le calcul distribué est un ensemble partiellement ordonné défini par le couple $(H,\rightarrow)$. Où H est l'histoire globale et $\rightarrow$ est la relation de précédence causale définie ci-dessus. La représentation graphique équivalente au calcul distribué est dite diagramme espace-temps.

7.4 DIAGRAMME ESPACE-TEMPS

Cette représentation graphique nous facilite la tâche pour observer l'évolution du calcul.

Chaque ligne horizontale représente l'exécution (l'évolution) d'un processus (la progression du temps se fait de gauche à droite). Une flèche d'un processus à un autre représente un message. Sa base est l'événement d'émission et le bout représente l'événement de réception du message (voir figure 1.8 ci-dessous).

Figure 1.8 : Exemple d'un diagramme espace-temps



Il est facile de savoir si deux événements sont liés causalement. Il suffit de tracer un chemin entre deux événements de gauche à droite horizontalement et/ou en suivant le sens des flèches pour conclure qu'ils ont un lien. Dans l'exemple ci dessus, on voit que e_1 de P_4 précède e_3 de P_1, par contre e_3 de P_1 et e_3 de P_5 sont indépendants.

7.5 OBSERVATION DU CALCUL DISTRIBUE

L'observation des processus d'un système est importante à plusieurs titres. Elle est nécessaire pour certains problèmes de contrôle. Elle permet de détecter des propriétés caractérisant le système :

- Passage dans un état particulier
- Vérification d'assertions
- La Terminaison, qui consiste à vérifier si le calcul distribué est terminé et aucun processus ne peut prendre cette décision seul.
- L'interblocage et ceci si dans un groupe de processus utilisant un même ensemble de ressources se constitue un circuit tel que chaque site attend un autre.

L'observation du calcul permet aussi, entre autres, d'effectuer des mesures de performances et de capter des états pouvant servir de points de reprise en cas de défaillance.

Dans le cas idéal, il faut que l'observation soit faite par un observateur externe au système [BM,1993], chose qui est impossible à réaliser dans un contexte de répartition car il est impossible d'effectuer une observation globale et instantanée du système. L'observation présente des difficultés qui rendent le problème non trivial.

De ce fait, le contrôle du système est fait par un processus appelé monitor. Ce processus fait partie du système mais peut être externe au calcul distribué.

7.6 ETAT GLOBAL

Le calcul de l'état global du système permet d'évaluer certains prédicats dans le but de détecter, vérifier ou déterminer certaines propriétés [BM,1993]. L'état global est défini comme l'ensemble des états locaux des différents processus du système.

Soient les notations suivantes :

σ_i^k ou $\sigma_{i,k}$: dénote l'état local du processus P_i juste après l'exécution de l'événement e_i^k ou $e_{i,k}$.

σ_i^0 ou $\sigma_{i,0}$: dénote l'état local initial du processus P_i (avant le début du calcul).

L'état local d'un processus contient généralement des informations telles que les valeurs des variables locales, la séquence des messages envoyés et reçus à travers les différents canaux.

De façon formelle, l'état global du calcul distribué est un n-tuple d'états locaux :

$\Sigma = (\sigma_1, \sigma_2, ..., \sigma_n)$, un état par processus.

Le problème est de s'assurer que l'état construit est significatif. Donc, on parle de cohérence de l'état global, qui revient à bien formaliser la notion du "raisonnement global avec des informations locales significatives". On reviendra sur le problème de cohérence plus loin dans ce chapitre.

7.7 LA COUPE

Une coupe du calcul distribué est un sous-ensemble *C* de l'histoire globale du calcul *H*.

$$C = h_1^{c_1} \cup h_2^{c_2} \cup ... \cup h_n^{c_n}$$

Où chaque c_i est un nombre entier correspondant à l'indice du dernier événement inclus dans la coupe pour le processus P_i.

L'ensemble des derniers événements $(e_1^{c_1}, e_1^{c_1}, ..., e_1^{c_1})$ inclus dans la coupe (c_1 , c_2 , … , c_n) est appelé frontière de la coupe.

Il est clair que pour chaque coupe définie par (c_1,c_2,…,c_n), on obtient un état global correspondant qui est $(\sigma_1^{c_1}, \sigma_1^{c_1}, ..., \sigma_1^{c_1})$.

Du point de vue graphique, la coupe est interprétée comme une division du diagramme espace-temps le long de l'axe temps. Dans l'exemple ci-dessous (figure 1.9), *C*1 représente la coupe correspondant à l'état global avant le début du calcul. La coupe *C*2 représente l'état du système après l'envoi du message *m5* par P_4 et avant sa réception par P_3. *C*3 est après l'envoi et réception de tous les messages.

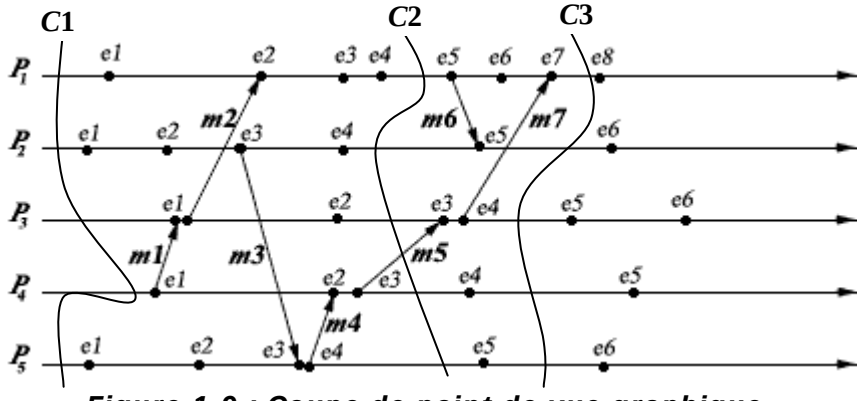


Figure 1.9 : Coupe de point de vue graphique

7.8 EXECUTION (RUN)

L'exécution *R* contient tous les événements de l'histoire globale *H* dans un certain ordre, de telle façon que les événements de chaque processus P_i apparaissent dans *R* dans le même ordre que dans h_i.

L'exécution représente une sorte d'ordre total alors que le calcul distribué est un ordre partiel. De ce fait, un seul calcul distribué peut avoir plusieurs exécutions différentes chacune correspondant à un déroulement différent du calcul.

7.9 LA COHERENCE

Une coupe *C* est dite cohérente (consistante) ***si*** pour chaque paire d'événements *e* et *e'* dans le système : $(e \in C)\ et\ (e' \rightarrow e)\ \Rightarrow\ e' \in C$.

La cohérence est liée directement à la relation de précédence causale.

A partir de la représentation graphique, il est facile de vérifier la cohérence de la coupe. Il suffit que pour toutes les flèches qui ont une intersection avec la coupe, leurs bases soient à gauche de la coupe et leurs bouts soient à droite.

Un état global cohérent est un état correspondant à une coupe cohérente.

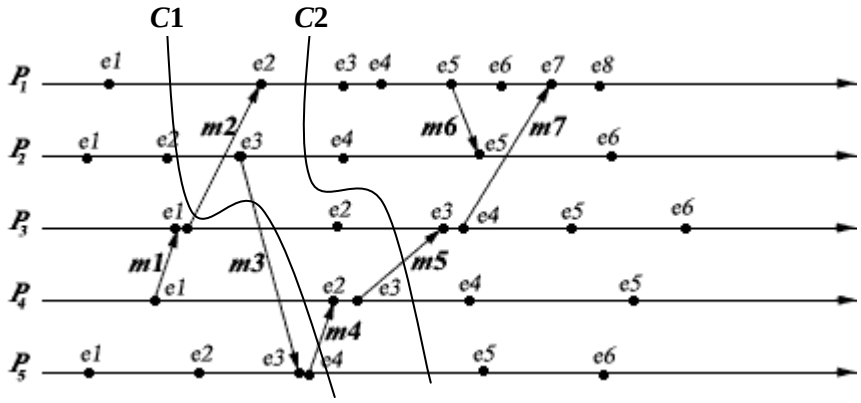


Figure 1.10 : Cohérence de la Coupe de point de vue graphique

Sur la figure ci-dessus, la coupe *C*1 n'est pas cohérente car l'événement de réception du message *m3* est à gauche de la coupe et l'événement d'émission est à droite. Par contre, la coupe *C*2 est cohérente.

8. LES MECANISMES D'ESTAMPILLAGE

A défaut d'une horloge à temps-réel globale, il faut trouver une technique simple permettant d'ordonner les événements et avoir une certaine forme de synchronisation globale entre les différents sites distants. Il existe des mécanismes d'estampillage qui permettent une certaine synchronisation entre les processus et de définir une relation d'ordre à travers les échanges de messages. Cet ordre est appelé l'ordre causal. Cette notion a été introduite par Lamport [Lam,1978].

8.1 HORLOGES LOGIQUES

Le mécanisme d'estampillage le plus simple (celui introduit par Lamport) est appelé *horloges logiques* (*logical clocks*) ou aussi *horloges scalaires*. Le principe est que chaque processus P_i gère un compteur implémenté par une variable locale de type entier

et notée Lc_i. Les valeurs de cette variable sont toujours positives et chaque processus P_i l'initialisé à 0 au lancement du calcul.

Il s'agit, en fait, d'associer un numéro (nombre entier positif) à chaque événement qui n'est autre que la valeur de l'horloge lors de la production de l'événement. Le déroulement des événements permettent d'incrémenter l'horloge. Cette valeur représente une date pour chaque événement.

On note $Lc(e_i)$ la valeur attribuée à l'événement e_i qui est aussi la valeur de l'horloge logique lorsque e_i est exécuté par P_i. On note Lc_i la valeur de la variable au niveau du processus P_i. Chaque message envoyé véhicule la valeur de l'horloge associée à l'événement *send* de l'émetteur de ce message et on la note *m.Lc*. On dit alors que l'horloge est embarquée (*piggybacking*) avec le message *m*. On dit que *m* est estampillé. Les événement locaux permettent d'incrémenter les horloges locales et les événements de réception permettent de synchroniser l'horloge du récepteur avec celle de l'émetteur. Avant le début du calcul, tout processus P_i du système initialise son compteur Lc_i à 0.

On peut résumer les règles de mise à jour à :

- $Lc(e_i) := Lc_i + 1;$ si e_i est un événement local ou une émission.
- $Lc(e_i) := max(Lc, m.Lc) + 1;$ si e_i est un événement de réception d'un message *m*.

Cette technique permet d'associer à chaque événement une date ou estampille. Ceci permettra d'ordonner les événements d'une façon qui n'est pas stricte (ordre partiel). Pour avoir un ordre total, il suffit, en cas d'égalité des horloges logiques, d'ordonner les événements selon les numéros de leurs processus par exemple.

De ce fait, les horloges logiques s'incrémentent en respectant la relation de précédence causale à condition que les canaux soient FIFO.

$\forall e \rightarrow e' \Rightarrow Lc(e) < Lc(e')$

Cette relation se démontre par récurrence sur la longueur de la chaîne d'événements reliant *e* à *e'*.

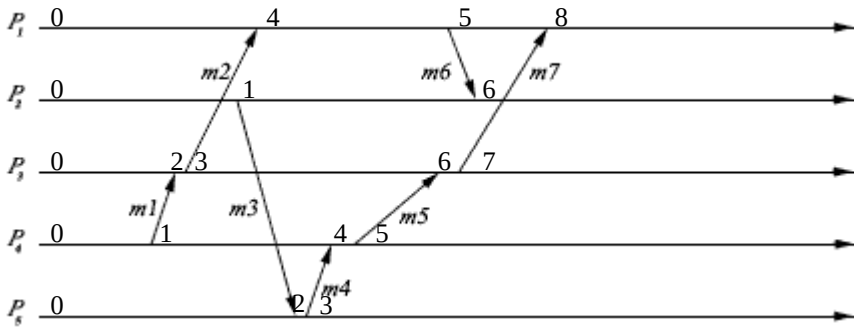


Figure 1.11 : Utilisation des horloges logiques

8.2 HORLOGES VECTORIELLES

Chaque site gère une horloge vectorielle constituée de *n* entiers (*n* étant le nombre de processus composant le système). Cette horloge est implémentée par un tableau de *n*

entiers positifs. On la note VC_i pour le processus P_i. Une telle horloge permet de dater les événements d'un site et est mise à jour lors de l'occurrence des événements. Comme pour les horloges scalaires, les messages envoyés par un site sont estampillés en utilisant la valeur courante de l'horloge vectorielle du site émetteur et la réception d'un message permet au site récepteur de synchroniser son horloge vectorielle avec celle du site émetteur du message. $VC_i[j]$ représente le numéro du dernier événement exécuté par P_j selon P_i. Chaque processus initialise son vecteur à 0 ($VC_i[j]:=0\ \forall j$) au début du calcul.

On note $VC_i(e)$ la valeur de l'horloge correspondant à l'événement e dans le processus P_i. On dit que $VC_i(e)$ est la date de l'événement e. VC_i représente la valeur actuelle de l'horloge au niveau du processus P_i. $m.VC$ est la valeur de l'horloge véhiculée par le message m. Les règles de mise à jour sont comme suit :

$VC_i(e)[i]:=VCi[i]+1;$ si e est un événement interne ou $e=send(m)$
ou
Pour $j:=1$ *à* $n\ VC_i(e)[j]:=max(VC_i[j],m.VC[j]);$
$VC_i(e)[i]:=VC[i]+1;$ si $e=receive(m)$

$\sum VC_i(e)[j]$ pour les $j \neq i$ représente le nombre d'événements dans le système qui précèdent l'événement e.

Pour deux valeurs de l'horloge vectorielle V et V', On dit que :

- $V=V' \Leftrightarrow \forall k : 1 \leq k \leq n : V[k] = V'[k]$
- $V < V' \Leftrightarrow (V \neq V')$ *et* $(\forall k : 1 \leq k \leq n : V[k] \leq V'[k])$

Les relations de dépendance ou indépendance causales entre deux événements e et e' produits respectivement par les processus P_i et P_j seront définies par :

- $e \rightarrow e' \Leftrightarrow VC(e) < VC(e')$
- $e \,||\, e' \Leftrightarrow (VC(e)[i] < VC(e')[i])$ *et* $(VC(e')[j] < VC(e)[j])$

Soient e et e' deux événements de la frontière de la coupe produits respectivement par les processus P_i et P_j. On dit que cette coupe est cohérente ***si et seulement si***,

- $\forall i,j : 1 \leq i \leq n$ *et* $1 \leq j \leq n\ VC_i(e)[i] \geq VC_j(e')[i]$

9. APPLICATION DISTRIBUEE

Une application répartie est mise en œuvre par un ensemble de processus communiquant formant un réseau de processus. La décomposition d'une application en processus est effectuée en fonction de la nature et l'utilisation de cette application, du système sous-jacent, des contraintes de conception et la localisation des ressources (une ressource peut être localisée sur plusieurs sites). L'application peut être vue comme un ensemble de services répartis [Mos,1994]. Tout service se divise en deux parties : les protocoles qui le mettent en œuvre et ceux qui réalisent l'interface.

10. ALGORITHME DISTRIBUE

Les algorithmes parallèles basés sur les échanges de messages entre processus sont appelés dans le domaine des réseaux protocoles. Dans le domaine des systèmes distribués, on les appelles algorithmes répartis.

Les algorithmes répartis réalisent des fonctions de base du système comme l'observation, la communication, la synchronisation … etc. Ces algorithmes ne s'intéressent pas au calcul exécuté par le processus mais contrôlent l'interaction du processus avec le reste du système (c'est à dire les autres processus).

Pour chaque processus P_i, il est possible de contrôler son état sans le changer. Par exemple, l'algorithme peut contrôler l'émission d'un message ou la réception sans se soucier du contenu du message.

Prouver la correction d'un algorithme réparti revient à montrer qu'il vérifie la propriété de sûreté et la propriété de vivacité. La première propriété implique qu'il n'arrive que ce qui est prévu par la spécification. La seconde propriété assure l'occurrence, au bout d'un temps fini, d'une propriété spécifiée. Elle écarte les problèmes d'interblocage et de famine (service refusé indéfiniment pour un processus et obtenu par d'autres).

Pour l'ordre causal, par exemple, la sûreté est assurée si aucune livraison de message ne viole l'ordre causal et la vivacité est assurée si tout message émis sera délivré à son destinataire au bout d'un temps fini.

Un certain nombre de critères permettent d'évaluer et de comparer différents algorithmes répartis qui réalisent une même fonction. Ces critères sont : la *symétrie* (si tous les processus exécutent le même texte), la *résistance aux défaillances* , les *hypothèses sur le système* (moins un algorithme fait d'hypothèses sur le support physique plus il est meilleur), le *trafic engendré* (la minimisation des messages augmentera l'efficacité), l'utilisation de *variables à domaine borné* et le *facteur d'échelle* (la taille du système, la mobilité, la zone géographique, la diversification des machine …).

11. CONCLUSION

Nous avons vu dans ce chapitre les différentes notions et terminologie nécessaires pour l'étude des systèmes répartis. Nous avons vu les différentes façons pour pallier à l'absence d'une horloge globale à temps réel. Nous avons aussi abordé la notion d'état global et le problème de la cohérence. Ceci nous permettra d'aborder dans les prochains chapitres les techniques *checkpointing*. Mais avant cela, dans le chapitre qui suivra, nous allons étendre le système pour parler des systèmes répartis dans un environnement plus sensible aux défaillances, c'est l'environnement mobile.

CHAPITRE II :

LES SYSTEMES REPARTIS MOBILES

1. INTRODUCTION

L'évolution rapide de la technologie dans les domaines de la communication et de la télécommunication (le cellulaire avec ses différentes générations, les systèmes basés satellite …) et des réseaux locaux sans fil, a rendu l'extension des systèmes répartis possible. Cette extension permet à l'usager d'accéder à l'information n'importe où et n'importe quand. Ces utilisateurs seront munis d'unités mobiles de diverses configurations et équipés d'interface de communication sans fil [Bad,1998].

Ce genre d'environnement, appelé environnement mobile, offre une liberté à l'utilisateur dans ses déplacements tout en restant connecté au réseau. Un système réparti en environnement mobile doit offrir aux usagers des services comparables à ceux habituellement offerts par les systèmes statiques sans compromettre leur aptitude de se déplacer. Ceci introduit des contraintes jusque là inconnues dans les systèmes distribués conventionnels. Ces nouvelles contraintes rendent le système encore plus vulnérable aux défaillances.

L'intérêt de l'étude des systèmes distribués mobiles est dicté par l'évolution de la technologie des communications qui a eu toujours une influence considérable sur l'informatique et plus précisément sur les réseaux informatiques.

L'objectif de ce chapitre est d'étudier cet environnement très contraignant et toutes ses caractéristiques et son impact sur le calcul distribué.

2. DESCRIPTION DU SYSTEME

Un système réparti avec des sites mobiles consiste en une infrastructure composée de :

- Sites fixes connectés à un réseau de communication filaire (wired network) qui représenteront un système distribué conventionnel.

- Sites mobiles connectés aux sites fixes à travers un réseau de communication sans fil (wireless network).

Certains sites fixes sont appelés Stations de Support pour Mobiles (Mobile Support Station) qu'on notera dans le reste de ce document MSS. Ces sites sont munis d'interfaces de communication sans fil. Les stations mobiles appelées aussi Mobile Hosts qu'on notera dans ce qui suivra MH vont se connecter au système à travers les MSS. Ainsi les MSS vont jouer le rôle de cellule et une MH sera connectée à une seule cellule à un instant donné et en se déplaçant d'une cellule à une autre elle change de MSS. Une cellule est une zone logique ou géographique couverte par une MSS. Dans un but de simplification de la présentation, nous considérons tous les sites fixes comme MSS [BAI,1994].

Le système (voir figure 2.1) peut être défini comme un ensemble de n processus s'exécutant sur n MSS : S_1, S_2, ..., S_n communiquant entre eux exclusivement par échange de messages à travers des canaux de communication filaires et m processus s'exécutant sur m MH : h_1, h_2, ..., h_m communiquant à travers un réseau sans fil en passant par la MSS à laquelle sont connectées. Généralement le nombre de MH est supérieur à celui des MSS ($m>>n$). Le nombre de MH n'est pas toujours fixe.

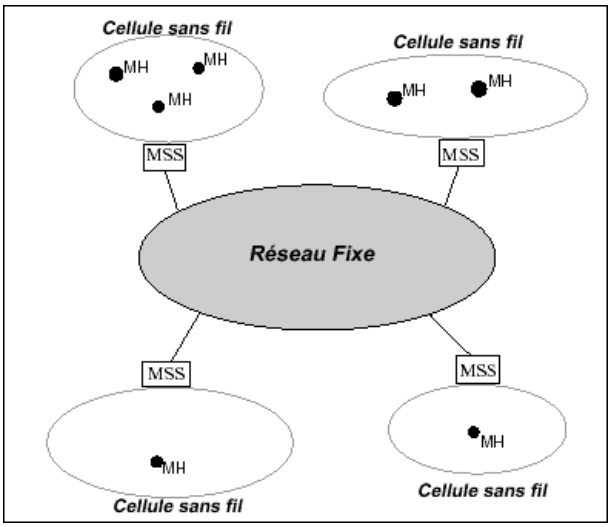


Figure 2.1 : Modèle de système distribué avec des unités mobiles

Le système garde toujours les caractéristiques d'un système distribué ordinaire. Il ne comporte ni horloge globale, ni mémoire partagée. Il est asynchrone, du fait de l'absence d'hypothèses sur le temps.

3. CARACTERISTIQUES DE L'ENVIRONNEMENT MOBILE

Les unités mobiles sont caractérisées par les déplacements, les déconnexions et re-connexions, la modestie de la capacité de calcul et de stockage. La bande passante du support de communication sans fil est faible en plus de la contrainte de l'énergie. Dans ce qui suit, nous donnerons plus de détails sur ces nouvelles contraintes.

3.1 LES RESSOURCES ET LA BANDE PASSANTE

Les unités mobiles peuvent être de diverses configurations. Elles peuvent avoir des capacités de stockage (disque), mémorisation et calcul plus ou moins modestes. Les *Ressources* machines des unités mobiles restent toujours *limitées* par rapport aux unités fixes malgré l'évolution de la technologie des ordinateurs portables.

Les unités mobiles utilisent le réseau de communication sans fil qui est loin d'être aussi fiable et rapide que le réseau filaire (réseau en fibre optique par exemple). Ceci rend la *bande passante* faible (voir tableau 2.1) et pose le problème des délais de transmissions et de réceptions qui seront plus grands. L'autre problème qui se pose est la livraison d'un message, un temps supplémentaire de recherche de la MH est nécessaire. Ceci peut ralentir considérablement le calcul ce qui nous pousse à toujours chercher un moyen pour minimiser le calcul sur les MH.

	Bâtiment	**Campus**	**Zone Large**	**Région**
Médium	LAN sans fil Infrarouge	Relais	Cellulaire	Satellite
Bande Passante	>1Mbps	>64Kbps	10-30Kbps	Asynchrone haut : <10Kbps bas : >1Mbps
Mobilité	Pédestre	Pédestre	Véhiculé Pédestre	Véhiculé Pédestre Stationnaire

Tableau 2.1 : Classification des réseaux mobiles

3.2 LES DEPLACEMENTS ET HANDOFF

La principale caractéristique des systèmes mobiles est le *changements de localisation* ce qui complique le routage des messages et augmente le temps de communication et la complexité des messages. Les messages envoyés d'un nœud à un autre peuvent être re-routés à cause de la déconnexion de l'unité mobile destinataire de la station statique actuelle vers la nouvelle station ce qui nécessite dans la majorité du temps une phase de recherche qui peut être très coûteuse en temps. Le changement de localisation et de ce fait de cellule, est appelé *Handoff* (figure 2.2). Il existe plusieurs protocoles qui permettent de gérer les handoffs.

Lorsqu'une MH h_i quitte sa cellule S_j et entre dans la cellule S_k ($j \neq k$), l'exécution du *protocole handoff* est déclenchée. Trois entités coopèrent dans ce protocole : la MH h_i et les deux MSS S_j et S_k. Le protocole est en général réalisé comme suit :

- h_i qui initialise la procédure lorsqu'elle rentre dans la cellule gérée par S_k en envoyant un message à cette dernière l'informant qu'elle vient de pénétrer sa cellule en quittant la cellule S_j.
- S_k envoie un message à S_j l'informant que h_i a changé de MSS.

- S_j fait quelques contrôles et calculs et transfert vers S_k les informations en sa possession concernant h_i

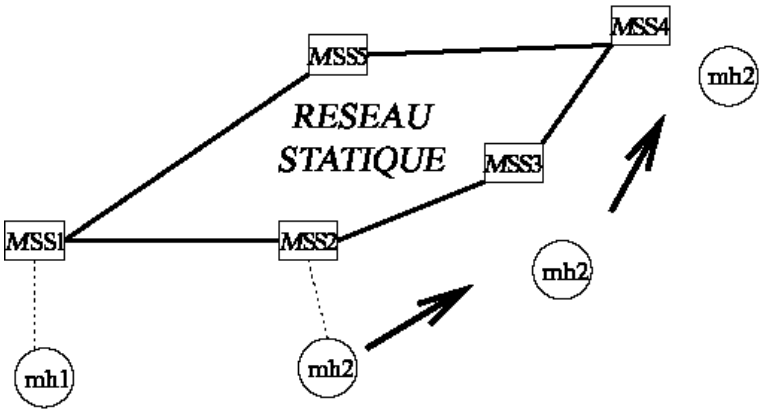


Figure 2.2 : Handoff

Des stratégie pour localiser une MH migrante sont nécessaires. Une MH peut demander un service au moment où elle est connectée à travers une cellule et en attendant la réponse elle peut se déplacer entre plusieurs cellules. Ainsi, pour délivrer un tel résultat, il faut connaître la nouvelle localisation de la MH. [BAI,1993a]

Le changement de localisation rend les structures logiques (arbre, étoile, graphe, anneau, grille,…), sur lesquels sont basés les résolutions de certains problèmes dans les systèmes répartis conventionnels, non adaptées [Bad,1995].

3.3 L'ENERGIE

Sur le plan *énergie*, les unités mobiles sont alimentées par une batterie qui se recharge (*source d'énergie autonome*). Les différents composants consomment de l'énergie ainsi que l'émission et réception de messages. Le système, durant la période de basse activité, éteint certains ou tous les composants. Ceci mettra à l'arrêt la majorité des fonctions de son système et reste seulement à l'écoute des messages entrants. Ce mode est appelé le mode veille (*doze-mode*) où le calcul ne reprendra que lors de la réception d'un message dans le seul but d'économiser de l'énergie.

Ceci nous pousse à toujours chercher des solutions pour le système distribué qui ne laissent pas la MH consommer trop d'énergie et traiter le cas où la machine est en mode de veille pour la déranger le moins possible durant cette période là.

3.4 LES DECONNEXIONS

Les systèmes mobiles sont aussi caractérisés par les *déconnexions fréquentes* et parfois sans avertir les autres processus. La déconnexion peut être aussi temporaire comme elle peut être permanente. Ainsi, la période de déconnexion est arbitraire. Ces déconnexions sont soit volontaires soit involontaires. Dans le premier cas (volontaires), elles peuvent être pour des raison propres à la MH, dues au déplacement de la station ce qui engendre un protocole de handoff, se trouvant dans une zone non couverte ou pour économiser de l'énergie (mode veille). Dans le deuxième cas (involontaires), la

déconnexion peut résulter d'une panne ou même de la vulnérabilité du réseau de communication sans fil.

Les conséquences de la déconnexion en mode veille diffère des autres déconnexions, car la MH reste joignable à partir du reste du système et toute autre station du système peut la remettre en mode normal si nécessaire.

Lors de la déconnexion, seuls les événements locaux seront exécutés sur la MH. C'est à dire, il n'y aura aucune émission ou réception de messages durant cette période. Dans le cadre du calcul mobile, il faut prendre en considération ce problème lors de la conception des protocoles et des applications.

3.5 VULNERABILITE AUX DEFAILLANCES

A cause des caractéristiques citées ci-dessus et en plus de la fragilité des unités mobiles (le risque de dégâts physiques est très élevé : la perte, le vol, la température, l'humidité, l'eau, l'exposition aux rayons …) rendent le système plus vulnérable aux défaillances (Parfois c'est toute la machine portable qui est remplacée). De ce fait, l'unité mobile n'est pas considérée comme endroit stable de stockage des informations importantes. Une des solutions consiste à sauvegarder les informations les plus importantes dans une MSS.

4. ALGORITHMES EN ENVIRONNEMENT MOBILE

Badrinath et son groupe dans [BAI,1993b] et [BAI,1994] ont tracé les grandes lignes de la conception des algorithmes pour les systèmes distribués en environnement mobile. Ils ont proposé le principe d'une architecture deux-tiers (à deux niveaux) pour structurer un algorithme dans cet environnement. Les MH et MSS ne seront pas traités de la même façon et ce dans le but de combler les lacunes de la disparité des ressources entre les MH et les MSS.

Ils est conseillé, de faire au mieux possible, pour que les coûts de calcul et de communication d'un algorithme distribué soient supportés par la partie statique du réseau. L'essentiel de l'algorithme doit être réalisé à travers l'exécution distribuée parmi les processus fixes pendant que les MH ne seront sollicitées que pour des opérations qui sont nécessaires pour la fonctionnalité globale voulue [BAI,1994].

L'algorithme doit prendre en charge les différentes contraintes de l'environnement mobile (citées ci-dessus) comme cela sera expliqué ci-dessous. Quelques recommandations pour la conceptions d'algorithmes distribués en environnement mobile seront présentées ci-dessous.

4.1 TRAITER LES HANDOFFS ET DECONNEXIONS

Dans un environnement mobile lors de la conception de l'algorithme, des traitements peuvent être rajoutés lors du protocole handoff ou la déconnexion. Ainsi une station mobile h_i, lorsqu'elle quitte sa cellule, elle envoie un message spécial appelé

leave à la MSS S_j qui couvre cette cellule. En rejoignant une nouvelle cellule couverte par S_k, elle envoie un message *join* à S_k. Lorsque h_i veut se déconnecter du réseau, elle envoie un autre message spécial *disconnect* à S_j. Pour se reconnecter de nouveau au réseau, elle envoie un message *reconnect* à S_k qui n'est pas nécessairement la même MSS que S_j qui a reçu *disconnect*.

De son coté la MSS S_j qui gère un ensemble de sites MH locaux qui lui sont connectés, en recevant un message *leave* de la part de h_i, supprime ce site de son ensemble et en recevant un message *disconnect*, S_j marque h_i en tant que déconnecté. En recevant le message *join* ou *reconnect*, S_j rajoute h_i à son ensemble. Pour le message *reconnect*, S_j envoie un message à S_k pour ne plus marquer h_i comme étant déconnecté.

Lors de l'envoi ou la réception de ce genre de message l'algorithme doit prévoir des taches qui permettront de prendre en charge ces deux aspects.

4.2 MISE EN ŒUVRE DU CANAL FIFO

Pour illustrer l'abstraction du canal FIFO sur un système distribué conventionnel, il suffit de numéroter séquentiellement les messages transmis à travers le canal. Toute rupture de séquence à l'arrivée des messages à l'autre bout du canal implique une perte de message ou un dé-séquencement dû à l'asynchronisme du système [Mos,1994].

Dans un environnement mobile, ce mécanisme peut être mis en œuvre sur la partie fixe du réseau et sur les canaux sans fil si les MH ne changent pas de localisation. Avec un tel mécanisme on peut poser les hypothèses que les canaux sont fiables et assurent une livraison FIFO des messages à condition que les MH se déplacent sans quitter leurs cellules. Dans [BAI,1994], une technique a été présenté pour que l'abstraction canal FIFO reste valide même lorsque la MH quitte sa cellule ou se déconnecte du réseau. La technique consiste à ce que la MH envoie le numéro de séquence avec les messages *leave* et *disconnect*.

4.3 COUT D'UN CALCUL MOBILE

La mesure typique de l'efficacité d'un algorithme distribué pour un réseau fixe est la complexité de communication de l'algorithme relativement au nombre de messages échangés dans une exécution de l'algorithme. Avec l'introduction des unités mobile et les contraintes associées, la complexité de communication doit aussi inclure un coût de recherche. Ce coût représente le nombre de messages échangés pour localiser une MH [BAI,1993b]. Vu que le coût d'un message envoyé sur un réseau filaire et un message envoyé sur réseau sans fil ne sont pas identiques, la complexité de communication doit considérer un troisième coût qui est le nombre de messages sans fil.

- C_{fixed} : Coût d'envoi d'un message point à point entre deux MSS.
- $C_{wireless}$: Coût d'envoi d'un message d'une MH vers sa MSS locale à travers un canal sans fil.
- C_{search} : cout dû à la localisation de la MH et le transferts du message à la MSS local courante.

En se basant sur ces coûts, un message envoyé d'une MH à une autre coûtera $2*C_{wireless}+C_{search}$. De la même façon, un message envoyé d'une MSS à une MH non locale coûtera $C_{wireless}+C_{search}$.

4.4 STRUCTURATION A DEUX-NIVEAUX (TWO-TIER)

Les algorithmes doivent être structurés en suivant l'architecture deux-tiers (two-tier structure) :

- Les demandes de calcul et communication de l'algorithme doivent être satisfaites sur le segment statique du système au mieux possible.
- Les structures de données encapsulant l'état de l'exécution de l'algorithme doivent résider dans une station fixe.

La communication nécessaire pour l'exécution de l'algorithme peut être découpée en trois composants : *global*, *local* et *recherche*. Le premier consiste en les messages où l'émetteur et le destinataires sont des MSS (communication nécessaire pour l'évolution de l'algorithme comme par exemple la mise à jour des structures de données appropriées). Le deuxième consiste à la communication dans une cellule sans fil entre une MH et sa MSS locale (par exemple initialisation de l'algorithme ou communication du résultat final de l'exécution). Le troisième composant consiste en les messages qu'échange une MSS pour localiser une MH. Il est suggéré que le composant global domine la communication globale.

4.5 PRISE EN CHARGE DES MH EN MODE VEILLE ET DECONNECTE

L'algorithme distribué conçu pour un calcul en environnement mobile ne doit pas obliger chaque MH de participer durant toute l'exécution de l'algorithme. Autrement dit, il faut éviter de déranger une MH en mode veille. L'algorithme nécessite de prendre en considération la possibilité que un ou plusieurs participants peuvent se déconnecter pendant qu'une exécution de l'algorithme poursuit sa progression. Ainsi, il doit prendre en charge la variabilité du nombre de processus participants au calcul et le surcoût de recherche si le reste des participants demandent à être informés de la déconnexion (ou même de la re-connexion) de la MH. Certains algorithmes de tolérance aux défaillances conçus pour les systèmes statiques considèrent le changement du nombre de processus participants comme un cas de défaillance. Or, les déconnexions des MH ne peuvent pas être considérées comme une défaillance vu que c'est une opération volontaire et ainsi ce genre d'algorithmes ne seront plus efficaces en environnement mobile.

Il est donc recommandé (selon le principe de la structuration deux-tiers) à ce que les MSS soient responsables de la progression de l'exécution de l'algorithme. Les déconnexions de une ou plusieurs MH ne doit pas altérer le nombre de participants dans l'algorithme. Ceci peut être réalisé par les MH qui doivent transférer (upload), vers la MSS, toute donnée nécessaire pour la progression de l'algorithme avant de se déconnecter. [BAI,1994]

4.6 LES STRUCTURES LOGIQUES

Plusieurs algorithmes distribués se basent sur des structures logiques sous-jacentes pour effectuer la communication nécessaire et ce dans le but de fournir un certain degré d'ordre et de prévisibilité à la communication entre processus. Les messages échangés vont suivre uniquement les chemins logiques décrits par la structure.

La déconnexion d'un nœud mobile peut nécessiter la reconfiguration de la structure logique et ceci résultera en un certain nombre de messages additionnels et un surcoût de recherche. La structure logique prédéfinit une séquence de nœuds que le message traversera pour atteindre sa destination à partir du processus émetteur. Si l'un des nœuds intermédiaires travaille en mode veille, il sera forcé à revenir au mode normal pour transférer le message au nœud suivant.

Ainsi, le coût de maintien d'une structure logique composées de MH outrepassera les avantages d'une pareille structure sous-jacente dans la conception de l'algorithmes.

Des exemples d'algorithme conçus pour les systèmes distribués adaptés pour l'environnement mobile seront trouvés dans [BAI,1994],[Bad,1995] et [Bad,1998].

5. CONCLUSION

Nous avons abordé dans ce chapitre l'introduction de la notion de mobilité à un système distribué et ce qui en résulte comme contraintes. Ainsi, l'étude d'un système distribué dans un environnement mobile devient plus complexe. Dans les deux chapitres qui vont suivre, nous allons voir les techniques de checkpointing. Dans le chapitre III, dans un environnement conventionnel puis nous utiliserons les notions abordées dans ce chapitre pour voir comment étendre les techniques existantes à un environnement mobile.

CHAPITRE III :

CHECKPOINTING CONVENTIONNEL

1. INTRODUCTION

Un *point de contrôle* distribué ou *snapshot* distribué est la combinaison des état locaux des processus. On dit qu'une image des état locaux des processus individuels est prise. Un point de contrôle global d'un calcul distribué asynchrone est une abstraction de ce qui est appelé usuellement état global. La détermination du checkpoint global (***le checkpointing***) est importante dans plusieurs applications. Par exemple, déterminer des points de reprises cohérents (consistants) est un problème assez connu dans les systèmes de gestion de base de données réparties.

Le *checkpointing* est une technique importante pour *minimiser la perte de calcul* dans un *environnement sujet aux défaillances*. Un point de reprise (checkpoint) est une copie de l'état d'une application sur un support de stockage stable. Une application sauvegarde périodiquement des checkpoints qui lui permettent de recouvrir après une défaillance en revenant en arrière vers le checkpoint le plus récent. Parmi les applications des checkpoints, sont inclus le debugging distribué, la tolérance aux pannes, la migration de processus, les systèmes distribués et parallèles, les systèmes basés objets, réseau de stations de travail, applications distribuées à mémoire partagée, micro-architecture, taches temps réel, calcul mobile, agents mobiles … [Lin,2001].

L'objectif de ce chapitre est de présenter les différentes techniques de checkpointing dans les système distribués conventionnels ainsi que les principaux travaux réalisés.

2. DEFINITIONS

Ci-dessous, nous allons voir quelques généralités et définitions de certaines notions qui seront nécessaires pour la suite comme le point de contrôle global ou la consistance d'un point de contrôle que ce soit de point de vue non formelle ou formelle (résultats de Netzer et Xu).

2.1 POINT DE CONTROLE GLOBAL

Un point de contrôle global (global checkpoint) est un ensemble de points de contrôle locaux, un par processus. Un point de contrôle local est un état enregistré par un processus. L'ensemble de points de contrôle locaux et un sous ensemble des états locaux d'un processus.

On note $C_{i,j}$ le i ème point de contrôle du processus P_j. On suppose que chaque processus P_i prend un point de contrôle local initial $C_{0,i}$ juste avant le début du calcul qui correspond à son état à ce moment là (l'état initial du processus) et un point de contrôle final fictif sera considéré à la fin du calcul. D'autres points de contrôle seront pris après certains événements. [HMR,1998].

2.2 CONSISTANCE D'UN POINT DE CONTROLE

La cohérence (consistance) d'un point de contrôle global dépend du flux de messages échangés durant le calcul du fait qu'un checkpoint global n'est cohérent que s'il ne comporte pas de message orphelin.

Un message M envoyé par le processus P_i au processus P_j est appelé orphelin respectivement au couple ordonné des enregistrements locaux $C_{x,i}$ et $C_{y,j}$ ***si*** M est livré dans $C_{y,j}$ ***alors*** que l'événement d'émission n'appartient pas à $C_{x,i}$.

Définition 1

Un couple ordonné de points de contrôle locaux est consistant si et seulement s'il n'y a pas de messages orphelins respectivement à ces couples.

Définition 2

Un point de contrôle global est consistant si tous ces couples de points de contrôle locaux sont consistants

2.3 RESULTATS DE NETZER ET XU

Le fait que deux points locaux ne soient pas liés causalement ne constitue pas une condition suffisante pour dire que le point global auquel appartiennent ces deux points est cohérent. Certaines dépendances cachées, non captables par un mécanisme d'estampillage, peuvent exister entre les points locaux. Ceci les empêche de participer dans le même point de contrôle global.

Netzer et Xu ont publié dans [NX,1995] des résultats qui s'avèrent être fondamentaux pour l'étude des points de contrôle. Ils ont notamment énoncé une condition nécessaire et suffisante qui permet de décider si un ensemble quelconque de points de contrôle locaux peut être étendu pour former un point de contrôle global cohérent. Ils ont introduit la notion de Z-Chemin. L'existence d'un tel chemin entre deux points de contrôle locaux révèle une dépendance qui leur interdit d'appartenir à un même point de contrôle global cohérent.

2.3.1 NOTION DE Z-CHEMIN (Z-PATH OU ZIGZAG PATH)

Dans cette partie, nous rappelons ces résultats. La notion importante est celle de Z-chemin, elle généralise la notion de chemin causal et la précédence de Lamport.

Il existe un Z-chemin de $c_{i,x}$ à $c_{j,y}$ si :

1) $i=j$ et $x<y$ (les deux points de contrôle sont relatifs au même processus et $c_{i,x}$ est le premier des deux),

ou

2) il existe une séquence de messages $[m_1, m_2, \ldots, m_q]$ $(q \geq 1)$ telle que :

(2.1) l'événement $send(m_1)$ est produit par P_i après $c_{i,x}$ (c'est à dire : $send(m_1) \notin I_{i,x'} \wedge x' < x$).

et

(2.2) $\forall\, m_l : 1 \leq l \leq q :$
soit $I_{k,z}$ l'intervalle dans lequel est produit $receive(m_l)$. l'événement $send(m_{l+1})$ est produit dans l'intervalle $I_{k,z'}$ avec $z' \geq z$ (c'est à dire : $receive(m_l) \in I_{k,z} \wedge send(m_{l+1}) \in I_{k,z'} \wedge z' \geq z$).

et

(2.3) L'événement $receive(m_q)$ est produit par P_j avant $c_{j,y}$ (en d'autres termes : $receive(m_q) \in I_{j,y'} \wedge y' < y$).

2.3.2 LA RELATION *Z*

La notion de Z-chemin due à Netzer et Xu est liée, de la manière suivante, à la relation $\rightsquigarrow$. Ainsi nous définissons cette relation comme suit :

$c_{i,x} \rightsquigarrow c_{j,y}$ si, et seulement si, il existe un *Z-chemin* de $c_{i,x}$ à $c_{j,y}$. Les Z-chemins donnent ainsi une vision opérationnelle de la relation $\rightsquigarrow$.

Le théorème suivant, énoncé et prouvé pour la première fois dans [NX,1995], puis étendu dans [BHR,1997] à des modèles de calcul plus généraux incluant notamment la communication par variables partagées, constitue le résultat fondamental associé aux points de contrôle.

2.3.3 THEOREME

Nous présentons dans cette section le théorème de Netzer et Xu [NX,1995].

Un ensemble de points de contrôle *S*, où chacun des points est issu d'un processus différent peuvent appartenir au même checkpoint global cohérent si et seulement si aucun point de contrôle de *S* n'a de Z-Chemin avec les autres éléments de *S* y compris lui même.

S peut être complété pour former un point de contrôle global cohérent si, et seulement si, $\forall\ a, b \in S : \neg(a \rightsquigarrow b)$.

Un cas particulier important est celui où *S* est réduit à un seul élément : *S={a}*. Dans ce cas le point de contrôle local *a* fait partie d'au moins un point de contrôle global cohérent si, et seulement si, $\neg (a \rightsquigarrow a)$. En d'autres termes *a* n'est pas impliqué dans un Z-Cycle.

3. CLASSIFICATION DES ALGORITHMES DE CHECKPOINTING

On se place dans le contexte général suivant : Tout processus, pour des raisons qui lui sont propres (par exemple, l'occurrence d'une échéance éventuellement périodique, la réception d'un signal, le passage à vrai d'un prédicat local, etc.) définit certains de ses états locaux comme des points de contrôle. Si les processus définissent leurs points de contrôle indépendamment les uns des autres, il est possible que la cohérence ne soit pas satisfaite. Pour pallier cette situation indésirable, certains protocoles optent pour une forme de coordination entre les processus qui s'avère nécessaire. Ainsi, pour certains un processus jouera le rôle de coordinateur et pour d'autres ils embarquent sur les messages des informations sur le checkpointing ou utilisent des configurations de communication.

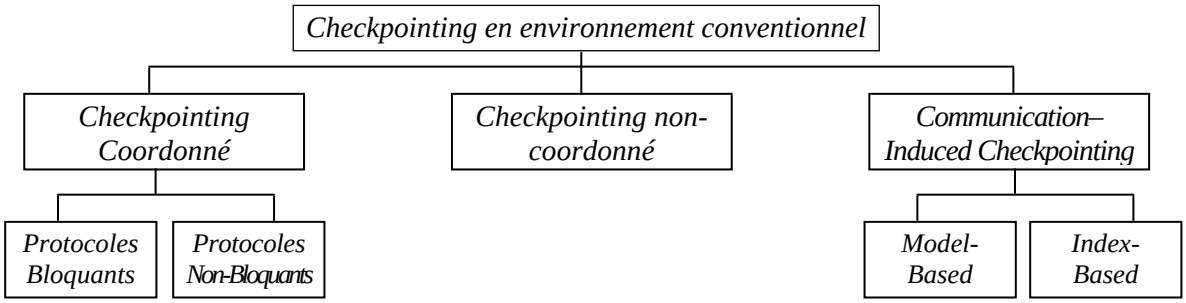


Figure 3.1 : Classification des techniques de checkpointing

De ce fait, nous distinguons deux approches dans la conception de protocoles de checkpointing : La classe des *protocoles de checkpointing non-coordonnée* (*uncoordinated checkpointing*) et la classe des *protocoles de checkpointing coordonnée* (*coordinated checkpointing*). Une troisième classe appelée *communication-induced checkpointing* (*checkpointing induit communication*) qui est plutôt proche des algorithmes non coordonnés. La figure 3.1 schématise la classification complète. Ci-dessous un peu plus de détails sur les différentes classes.

3.1 CHECKPOINTING COORDONNE

Les protocoles de définition de checkpoints se distinguent les uns des autres par les mécanismes de synchronisation qu'ils utilisent pour définir certains points de contrôle. Cette coordination constitue le problème majeur du *checkpointing*. Ainsi, des messages de contrôle (qui viennent s'ajouter à ceux de l'application) sont utilisés pour réaliser la coordination. On parle aussi de protocoles à synchronisation explicite [HMR,1998]. Un processus se chargera pour orchestrer l'application de checkpointing afin d'obtenir un état global cohérent de telle façon que l'ensemble des derniers points de contrôle pris par

les processus forment cet état. Ainsi, après défaillance, le recouvrement se fait à partir du dernier checkpoint pris (le plus récent).

De ce fait, les processus nécessitent de garder un seul point de contrôle permanent dans un support stable, ce qui réduit le surcoût de stockage et élimine le besoin de la fonction de ramasse-miettes (garbage collector).

Le principal inconvénient des protocoles de cette classe est la latence. Il existe deux grandes orientations dans la conception des algorithmes de checkpointing coordonnés. La première consiste à bloquer le calcul distribué pour engager un nombre minimum de processus dans le checkpointing. Pour la seconde, elle consiste à ne jamais bloquer le calcul sous-jacent.

3.1.1 ALGORITHMES BLOQUANTS

Les algorithmes coordonnés bloquants se font généralement en deux phases comme décrit ci-dessous. Un des processus du système prendra l'initiative pour déclencher le checkpointing.

Lors de la première phase, le coordinateur qui est appelé dans ce cas aussi l'initiateur prend son point de contrôle et diffuse à tous les processus une requête leur demandant d'entrer dans le processus de checkpointing. Tout processus qui reçoit cette requête sera engagés dans le checkpointing et arrêtera son calcul pour prendre son point de contrôle local. Ainsi, les processeurs seront gelés jusqu'à ce que tous les messages en transit arrivent à destination puis prennent leurs checkpoints pour continuer par la suite le calcul normal. Il met ce point de contrôle comme tentative de checkpoint puis envoie un message de reconnaissance à l'initiateur en retour.

L'initiateur en recevant un tel message de la part de tous les processus qu'il a engagé dans l'algorithme de checkpointing, entamera la deuxième phase du protocole en diffusant un message spécial appelé *commit*. En recevant ce message, chaque processus mettra la tentative de checkpoint comme permanente. Après, chaque processus sera libre de continuer son calcul normal.

Ces algorithmes ne sont pas toujours mauvais mais consomment beaucoup de temps à cause du blocage du calcul. Le principal avantage est le fait qu'ils peuvent engager un nombre minimum de processus dans le checkpointing. Ceci est réalisé à travers une communication directe ou indirecte des différents processus avec l'initiateur depuis le dernier checkpoint pour savoir s'il est nécessaire de prendre un nouveau checkpoint ou non. Dans ce cadre nous citerons le travail de Koo et Toueg dans [KT,1987] qui ont réalisé un protocole bloquant qui engage un nombre minimum de processus.

3.1.2 ALGORITHMES NON-BLOQUANTS

Pour ne pas bloquer l'envoi et réception des messages durant le checkpointing, le problème fondamental est de prévenir un processus lors de la réception de messages d'applications qui risquent de rendre le checkpoint global incohérent. Prenons l'exemple de la figure 3.2, P_0 envoie un message *m* après avoir reçu la requête

de checkpoint (*checkpoint request*) de la part du coordinateur (*initiator*). Si on suppose que *m* arrivera à P_1 avant la requête de checkpoint, cette situation va donner une incohérence car $c_{1,x}$ a enregistré la réception de *m* alors que $c_{0,x}$ n'a pas enregistré l'évènement d'émission.

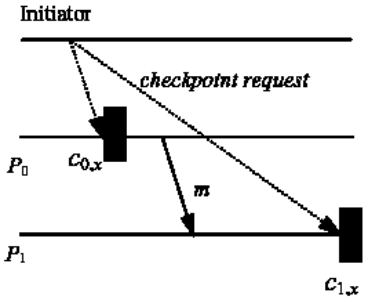


Figure 3.2 : inconsistance d'un checkpoint

Si les canaux sont FIFO, ce problème peut être évité comme l'illustre la figure 3.3. Ceci est réalisable, en envoyant juste avant l'émission d'un message sur un canal après le checkpoint de la requête sur le même canal.

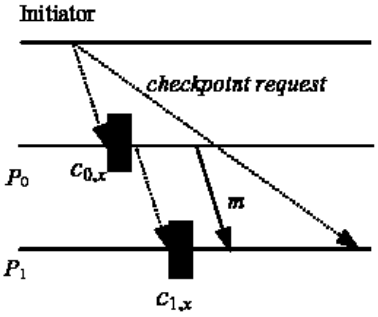


Figure 3.3 : checkpointing coordonné avec canaux FIFO

C'est l'idée développée par Chandy et Lamport [CL,1985] qui étaient les premiers à trouver une technique qui ne bloquera pas le calcul sous-jacent. Le processus coordinateur initie le checkpointing en diffusant un message spécial à tous les processus. Les détails des travaux de Chandy et Lamport seront abordés plus loin dans ce chapitre.

Si les canaux ne sont pas FIFO, la requête peut être embarquée sur les messages de calcul [LY,1987]. On a aussi parlé d'une technique dans le chapitre précédent pour la mise en œuvre des canaux FIFO.

Les algorithmes non-bloquants sont intéressants et très adaptés pour le recouvrement après défaillance. En effet, chaque processus revient au dernier checkpoint enregistré (le plus récent).

Prakash et Singhal [PS,1996] ont essayé de rallier la notion de protocole non-bloquant à l'avantage des algorithmes bloquants qui est l'engagement d'un nombre minimum de processus dans le processus de checkpointing. Nous parlerons en détail de ce travail dans le chapitre suivant.

3.2 CHECKPOINTING NON COORDONNE

Les algorithmes adoptant cette technique sont simples avec un coût d'exécution très bas. Les protocoles de cette classe (appelés aussi dans la littérature *Checkpointing conventionnel indépendant*) ne nécessitent pas de processus coordinateur. Chaque processus sauvegarde périodiquement ses états de façon indépendante et aucun message supplémentaire n'est nécessaire.

Cette méthode offre aux processus plus d'autonomie dans la décision de prise de checkpoint. Le principal avantage de cette autonomie est que chaque processus peut prendre un checkpoint au meilleur moment qui lui convient (par exemple quand la quantité d'information sauvegardée est petite pour réduire le surcoût du checkpointing local). Mais cette technique a aussi plusieurs inconvénient.

Le premier est la possibilité que l'effet domino peut survenir ce qui génère, lors du recouvrement, une perte d'une importante quantité de travail réalisé et peut même faire un retour arrière jusqu'au début du programme. Le système doit donc prévenir l'effet domino. Il faut donc forcer certains processus à prendre d'autres checkpoints pour garder la cohérence.

Le deuxième inconvénient est que certains points de reprise pris ne seront d'aucune utilité, qui ne peuvent appartenir à aucun checkpoint global cohérent. Le troisième est lié à la capacité de stockage car chaque processus est forcé de maintenir plusieurs points de contrôle ce qui nécessite une fonction de ramasse-miettes (garbage collector) pour récupérer les points de contrôle qui ne seront d'aucune utilité. Le dernier inconvénient est cité dans [EAWJ,1999] qui montre que ce genre de protocoles ne sont pas adéquats aux systèmes ayants de fréquentes sorties.

Dans le but de déterminer un checkpoint global cohérent pour le système réparti lors du recouvrement, les processus enregistrent les dépendances parmi leurs points de contrôle comme suit : Soient les notations $c_{i,x}$ le *x*ème checkpoint du processus P_i où x est appelé l'index du checkpoint et $I_{i,x}$ l'intervalle entre les checkpoints d'index x-1 et x (figure 3.4 illustre les notions d'index et intervalle). Si P_i envoie un message m durant l'intervalle $I_{i,x}$ à P_j, il doit embarquer la pair (i,x) dans m. Quand P_j reçoit m durant l'intervalle $I_{j,y}$, il enregistre la dépendance de $I_{i,x}$ à $I_{j,y}$ qui est sauvegardée sur un support de stockage stable quand P_j prend son point de contrôle $c_{j,y}$.

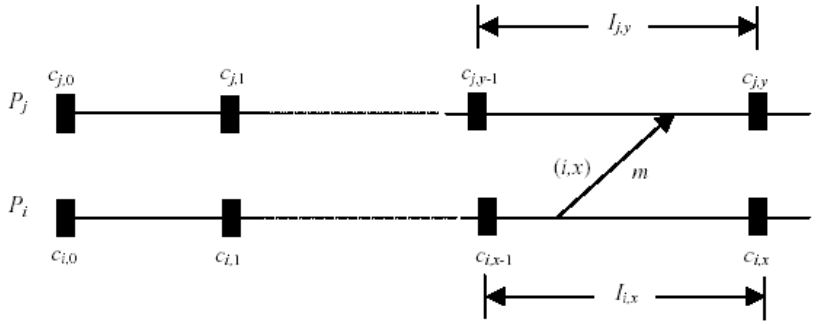


Figure 3.4 : Index et Intervalle de checkpoint

En cas de défaillance, le processus de recouvrement initie le retour arrière en diffusant un message *dependency request* qui est une requête pour collecter toutes les dépendances maintenues par chaque processus. Lors de la réception d'un tel message le processus arrête son exécution et répondra avec les informations de dépendance sauvegardées. L'initiateur du processus de recouvrement calculera par la suite la ligne de recouvrement en se basant sur les informations de dépendance globale et la diffusion de requête de retour arrière (*rollback request*). Un processus qui reçoit une telle requête, vérifie si son état appartient à la ligne de recouvrement alors il continue normalement son exécution sinon il doit revenir jusqu'au dernier point de contrôle indiqué dans la ligne de recouvrement. Il existe deux approches pour déterminer les lignes de recouvrement. La première repose sur les graphes de dépendance-retour arrière (rollback-dependency graph) et la seconde sur le graphe de checkpoint. Vous trouverez plus de détails dans [EAWJ,1999].

3.3 CHECKPOINTING INDUIT COMMUNICATION

Cette technique (communication-induced checkpointnig) est utilisée pour éviter l'effet domino survenu dans la première technique (Checkpointing non coordonné) lorsque les processus prennent leurs point de contrôle indépendamment. Aucun message de contrôle supplémentaire n'est utilisé et donc aucun coût supplémentaire de synchronisation n'est ajouté au calcul distribué. Les algorithmes non-coordonnés utilisent des configurations de communication ou de l'information embarquée (*piggybacked information*) pour obtenir la cohérence.

Toutefois, l'indépendance des processus est contrainte afin de garantir l'éventuelle progression de la ligne de recouvrement. Par conséquent, les processus peuvent être forcés à prendre des checkpoints supplémentaires.

Pour cette technique, nous distinguons deux types de *checkpoints locaux* : Un *point de contrôle local spontané* (*basic checkpoint*) qui est défini par le processus de sa propre initiative et un *point de contrôle local forcé* (*forced checkpoint*) appelé aussi *checkpoint adapté*. L'objectif est de déterminer quand est-ce un point de contrôle local individuel peut être combiné avec d'autres dans la perspective de former un *point de reprise global consistant* (*cohérent*).

Pour cette classe aussi, il existe deux grandes orientations : *Checkpointing basé index* (*Index-based checkpointing*) et *checkpointing basé modèle* (*model-based checkpointing*).

Index-based checkpointing: Les messages de l'application sont utilisés pour véhiculer des informations de contrôle; celles-ci sont utilisées par le destinataire pour décider s'il doit ou non prendre un point de contrôle forcé. On parle aussi dans la littérature de protocoles à synchronisation implicite [HMR,1998]. Dans cette technique un index de checkpoint permet d'archiver la consistance (cohérence). L'incohérence entre deux checkpoints de même index peut être évitée au vol si l'index du checkpoint est embarqué dans chaque message de calcul. En cas de réception d'un message avec un index supérieur à celui sauvegardé localement, le processus est forcé à prendre un point de contrôle avant de traiter le message. Ceci évite une incohérence au vol. Le coût d'un tel algorithme est élevé à cause du nombre de checkpoints qui seront pris. Des stratégies

ont été proposées pour diminuer le nombre de ces checkpoints. En se basant sur les travaux de Netzer et Xu (notion de *z-chemin* développée plus haut), des techniques de diminution de checkpoints forcés ont été proposées. Parmi les plus récents travaux nous citerons ceux de Baldoni et son équipe [BHR,1997][BHMR,1997] dont l'algorithme sera donné vers la fin de ce chapitre.

Model-based checkpointing : Le système conserve le point de contrôle et les structures de communication qui vont prévenir l'effet domino ou effectue des propriétés fortes. Parmi les techniques de model-based checkpointing nous citerons [Lin,2001]:

- CASBR (Checkpoint After send Before Receive) : consiste à ce que chaque processus prend un checkpoint à chaque fois après avoir envoyé un message et avant de recevoir un message.
- CAS (Checkpoint After Send) : consiste à ce que chaque processus prend un checkpoint à chaque fois après l'envoi d'un message d'un message.
- CBR (Checkpoint Before Receive) : consiste à ce que chaque processus prend un checkpoint à chaque fois qu'il reçoit message et avant de le traiter.
- NRAS (No Receive After Send) : Cette technique assure qu'il n'y ait pas de reception de messages après l'envoi d'un message. Ainsi, le processus doit prendre un checkpoint avant de délivrer un message reçu si au moins un message a été reçu depuis le dernier checkpoint pris.
- FDI (Fixed Dependency Interval) : Ce modèle combine les techniques de piggybacking avec les autres modèles dans le but d'éviter les points de contrôle inutiles et ainsi réduire le nombre de checkpoints.
- FDAS (Fixed Dependency After Send) : Les autres modèles sont intégrés dans cette famille. Il rajoute une variable booléenne au modèle FDI.

Les tableau récapitulatif ci-dessous dresse une comparaison des performances des trois grandes classes de checkpointing

	Checkpointing coordonné	**Checkpointing non-coordonné**	**Communication-induced checkpointing**
Checkpoints par processus	Un seul	Plusieurs	Plusieurs
Effet domino ?	Non	Possible	Non
Garbage collection	Simple	Complexe	Complexe
Recouvrement complexe ?	Non	Oui	Oui
Retour arrière	Dernier checkpoint	Non borné	Possible plusieurs checkpoints

Tableau 3.1 : Comparaison de classes de checkpointing

4. TRAVAUX ANTERIEURS

Le premier algorithme de *checkpointing* coordonné a été présenté par Barigazzi et Strigini dans [BS,1983]. Il est très restrictif car il suppose que toutes les communications sont atomiques. Le travail de Koo et Toueg [KT,1987] a enlevé cette restriction et réduit le nombre de messages de synchronisation et ainsi le nombre de *checkpoints*. Leu et Bhargava [LB,1988] ont réglé le problème des canaux FIFO du précédent algorithme. Leur algorithme embarque un drapeau (flag) avec les messages qu'il envoyés sur chaque canal. A partir de là, le récepteur vérifie s'il a besoin de prendre un point de contrôle avant de traiter le message. Deng et Park, dans [DP,1994], ont apporté des améliorations en proposant un algorithme qui adresse en même temps les messages orphelins et les messages perdus. Cristian et Jahanian dans [CJ,1991] et Ramanathan et Shin dans [RS,1993] ont proposé des *horloges de synchronisation* (*loosely synchronous clocks*) qui permettent de déclencher en local l'action de checkpointing pour tous les processus approximativement au même moment sans qu'il y ait d'initiateur. Le problème est qu'une panne risque de ne pas être détectée à temps et en même temps ça va restreindre le système qui est naturellement asynchrone.

Tous ces travaux nécessitent de bloquer le calcul durant le checkpointing. Le premier algorithme non bloquant est celui de Chandy et Lamport [CL,1985]. Ils ont proposé un algorithme où des marqueurs sont envoyés à tous les autres processus durant le checkpointing. Leur travail suppose que les canaux sont FIFO et génère une grande complexité des messages. Lai et Yang [LY,1987] l'ont amélioré en enlevant la supposition que les canaux sont FIFO. Ils embarquent sur chaque message envoyé sur chaque canal la requête de checkpointing. Leur algorithme nécessite beaucoup d'espace pour sauvegarder l'histoire entière des messages pour chaque canal.

L'algorithme proposé par Al Nozahy et al. dans [EJZ,1992] utilise une numérotation séquentielle des points de contrôle (checkpoint sequence number) pour identifier les messages orphelins et évite aux processus de bloquer leur calcul durant le checkpointing. Toutefois, cette approche nécessite que l'initiateur communique avec tous les autres processus en calcul.

Dans les deux sections suivantes, nous allons voir avec un peu plus de détails deux algorithmes de checkpointing, l'un est celui proposé par Chandy et Lamport, le premier protocole coordonné non bloquant et le second (induit-communication) est basé sur les conclusions des travaux Netzer et Xu.

5. TRAVAUX DE CHANDY ET LAMPORT

Le protocole assume que les canaux sont fiables et FIFO. Le protocole dû à Chandy et Lamport [CL,85] s'appuie sur des messages spéciaux dits de contrôle appelés *marqueurs*. Ces marqueurs ne doivent pas influer sur le calcul sous-jacent. Tout marqueur *mk* est doté de la propriété suivante : sur le canal où il est émis, tous les messages émis avant *mk* sont délivrés et tous les messages émis après *mk* sont délivrés

après *mk*. Il n'y a pas de contraintes particulières sur la livraison des messages de l'application à part celles qui les lient aux marqueurs.

Lors de l'envoi de marqueur par P

```
For each channel c, incident on, and directed away from P:
        record P's state;
        Send(mk) along c;
```

```
        if P has not recorded its state then
        record P's state; state{c} = {};
        else state{c} = {messages received along c after q's
state was recorded and before P received mk along c };
```

Lors de la réception de mk par P à travers le canal c

Figure 3.5 : Algorithme de Checkpointing [CL,1985]

L'algorithme se déroule comme illustré par la figure 3.5 ci-dessus. Ainsi, un processus *P* initie le checkpointing en envoyant *mk* à tous les processus qui ont un canal d'entrée les liant à *P* pour leur demander d'enregistrer leurs états locaux. Après avoir envoyé *mk*, *P* enregistre son état et après il pourra continuer son calcul normalement et envoyer des messages de calcul.

Un processus en recevant *mk*, soit il a déjà enregistré son état local soit il l'a pas encore fait. Pour le premier cas, il mettra la séquence de messages reçus à travers le canal par lequel il a reçu *mk* (après l'enregistrement de son état et avant la réception de *mk*) comme étant l'état du canal. Pour le deuxième cas, il enregistre son état local et considère l'état du canal comme vide.

Ce protocole mène à une complexité de messages $O(n^2)$. Car le marqueur circulera à travers tous les canaux. Le checkpointing dans ce cas est très coûteux.

6. PROTOCOLES BASES SUR LES Z-CHEMINS

La relation binaire $(C, \rightsquigarrow)$, définie plus haut, constitue une abstraction d'une exécution répartie. La technique de checkpointing peut prendre diverses formes mais obéit dans tous les cas au schéma suivant. Les points de contrôle locaux définis comme tels à l'initiative des processus sont appelés checkpoints spontanés. Afin que $(C, \rightsquigarrow)$ satisfasse le critère de cohérence recherché, le mécanisme de coordination oblige les processus, lorsque certaines conditions sont réalisées, à définir d'autres états locaux comme points de contrôle. On parle alors de checkpoints forcés. En résumé, les états locaux d'un processus se décomposent en trois classes :

- Ceux qui sont définis comme points de contrôle à la seule initiative du processus (checkpoints spontanés).
- Ceux qui sont définis comme points de contrôle par le mécanisme de coordination (checkpoints forcés).
- Ceux qui ne sont pas des points de contrôle.

Les critères de cohérence sont définit par les deux points suivants :

- (P1) Tout point de contrôle local appartient à au moins un checkpoint global cohérent.
- (P2)Tout point de contrôle local est associé, lors même de sa définition, à un des checkpoints globaux cohérents auxquels il appartient.

Alors que P1 signifie qu'il n'existe pas de point de contrôle local inutile, P2 associe au vol une identité globale à chaque point de contrôle local. Ceci permet une reconstruction et une exploitation aisées de points de contrôle globaux cohérents (ceci est particulièrement utile lorsque les points de contrôle sont des points de reprise utilisés pour le recouvrement arrière).

Au niveau du modèle décrit précédemment, le point P1 est satisfait si, et seulement si, il n'y a pas de Z-cycles. Dans cette partie, nous commençons par énoncer une propriété opérationnelle (fondée sur des points de contrôle locaux) qui s'avère être une condition nécessaire et suffisante pour prévenir les Z-cycles. Il est montré que tous les protocoles qui garantissent les points P1 et P2 énoncés précédemment s'appuient, de façon explicite ou implicite, sur cette propriété.

6.1 PROTOCOLES A SYNCHRONISATION EXPLICITE

Le protocole est décrit par les deux règles suivantes (lc_i représente l'estampille du dernier point de contrôle local de P_i) :

- ***(R1) Point de contrôle spontané :*** chaque fois que P_i prend un tel point de contrôle $c_{i,x}$, il incrémente lc_i de 1, attribue la valeur de $lc_{i,x}$ à $c_{i,x}.t$ et envoie des marqueurs $mk(lc_i)$ à chacun des autres processus.
- ***(R2) Point de contrôle forcé :*** lorsque P_i reçoit $mk(lc)$, si $lc_i < lc$ il prend un point de contrôle forcé $c_{i,x}$, met à jour lc_i ($lc_i := lc$) et attribue la valeur de lc à $c_{i,x}.t$; de plus, il diffuse à son tour des marqueurs comme dans le cas *R1*.

Il est facile de voir, d'une part, que ce protocole (simple) satisfait le théorème (il n'y a donc pas de point de contrôle local inutile : *P1*) et, d'autre part, que tous les points de contrôle locaux de même estampille définissent un point de contrôle global cohérent (*P2*), il est facile de voir que le coût de ce protocole en nombre de massages est proportionnel au nombre de canaux de communication.

Avec ce protocole, lorsque de façon spontanée, un (ou plusieurs) processus définit son *x-ième* point de contrôle local, tous les autres processus définiront également leur *x-ième* point de contrôle local. Cette synchronisation globale n'est pas nécessaire pour obtenir les conditions du théorème.

6.2 PROTOCOLES A SYNCHRONISATION IMPLICITE

Le principe est très simple : il consiste à faire transporter les estampilles par les messages de l'application. Les deux règles qui lui sont associées sont les suivantes.

- ***(R1) Point de contrôle spontané :*** lorsqu'un processus P_i définit un tel point de contrôle $c_{i,x}$, il incrémente lc_i de 1 et attribue à $c_{i,x}.t$ la nouvelle valeur de lc_i

- ***(R2) Point de contrôle forcé :***
  - Tout message m transporte la valeur courante lc_j de son émetteur P_j (soit $m.lc$ cette valeur)
  - Lors de la réception d'un message m, P_i teste si $lc_i < m.lc$. Si tel est le cas, P_i recale lc_i à la valeur de $m.lc$ et définit l'état courant comme point de contrôle $c_{i,x}$ (avec $c_{i,x}.t = lc_i$).

En considérant des exécutions réparties dotées de points de contrôle spontanés, le nombre de points de contrôle forcés pris par un protocole constitue une métrique simple pour évaluer son coût. Plusieurs protocoles visant à réduire ce nombre ont été proposés. Nous présentons ici quelques unes de ces améliorations. La première porte sur la condition associée à la définition d'un point de contrôle forcé, la seconde porte sur la gestion des estampilles.

La stratégie de diminution des points forcés peut facilement être mise en œuvre en associant deux tableaux à tout processus P_i. $sent\text{-}to_i[k]$ a la valeur vrai si, et seulement si, a envoyé des messages à P_k depuis son dernier point de contrôle ; $min\text{-}to_i[k]$ désigne alors la plus petite estampille de ces messages. La règle *R2* devient alors (la règle *R1* restant inchangée) :

- ***(R2) Point de contrôle forcé :*** lorsque P_i reçoit un message m, il effectue $lc_i = max(lc_i,m.lc)$ et prend un point de contrôle forcé si, et seulement si : $(\exists\ k : sent\text{-}to_i[k] \wedge m.lc < min\text{-}to_i[k])$

Deux points de contrôle locaux consécutifs $c_{i,x}$ et $c_{i,x+1}$ ne sont pas équivalents si, entre $c_{i,x}$ et $c_{i,x+1}$, P_i a reçu un message m tel que : $m.lc \geq c_{i,x}.t$

Cette notion de points de contrôle locaux équivalents est proposée dans le protocole introduit dans [QCB,1998]. La règle *R1* devient dans ce protocole (la règle *R2* reste inchangée) :

- ***(R1) Point de contrôle spontané :*** lorsque P_i prend le point de contrôle spontané $c_{i,x}$, il n'incrémente lc_i de 1 que si $c_{i,x}$ n'est pas équivalent à $c_{i,x-1}$. La valeur lc_i est attribuée pour estampille à $c_{i,x}$.

6.3 UN PROTOCOLE [HMR,1997]

Nous allons voir dans cette section, un protocole qui mettra en œuvre les résultats qu'on a vu ci-dessus. Le protocole, comme le détaille la figure 3.6 ci-dessus, consiste en trois procédures, une (*checkpoint*) exécutée lors de la prise d'un point de contrôle local, une (*send*) lors de l'envoi d'un message et la dernière (*delivery*) lors de la réception d'un message. C'est un protocole à synchronisation explicite car il n'utilise pas de messages systèmes.

```
procedure checkpoint
  begin
    etat_local := Calculer l'état local;
    C_{i,x} := etat_local;
    C_{i,x}.t := lc_i; x :=x+1;
    MIN_SENT_i := +∞; MIN_REC_NR_i := +∞;
    MX_REC_i := -∞; MX_SENT_NR_i := -∞;
    lc_i := lc_i + 1;
  end
```

```
procedure send
  begin
    m.t := lc_i + valeur;  // *** valeur>=0 pour incrémenter lc_i
    case   m.t = lc_i : m.recorded := false;
           m.t > lc_i :
              case   m.t > min(MIN_SENT_i,MIN_REC_NR_i): record m; m.recorded:=true;
                     m.t <= min (MIN_SENT_i, MIN_REC_NR_i): lc_i := m.t; m.recorded := false;
              endcase
    endcase;
    if (not(m.recorded)) then MX_SENT_NR_i:= max(MX_SENT_NR_i ,m.t); endif;
    MIN_SENT_i := min(MIN_SENT_i ,m.t); Send_message m;
  end
```

```
procedure delivery
  begin
    case   m.t < lc_i : if (not(m.recorded)) then record m; m.recorded:=true endif;
           m.t = lc_i : skip;
           m.t >lc_i :
             case   m.t <= min (MIN_SENT_i, MIN_REC_NR_i) : skip;
                    m.t > MIN_SENT_i : checkpoint();
                    MIN_REC_NR_i<m.t<MIN_SENT_i : DO
                       EITHER checkpoint
                       OR S :={m'∈REC_Nr_i | m'.t<m.t}; ∀m'∈S : record m'; REC_Nr_i := REC_Nr_i -
                          S;
                          if (REC_Nr_i = Ø)
                            then MIN_REC_Nr_i := +∞
                            else MIN_REC_Nr_i := min(m'.t / m'∈REC_Nr_i); endif ;
             endcase;
             lc_i := m.t;
    endcase;
    if (not(m.recorded)) then REC_Nr_i := REC_Nr_i ∪{m}; MIN_REC_Nr_i := min(MIN_REC_Nr_i,m.t);
                              endif;
    MX_REC := max(MX_REC,m.t); Deliver_message m;
```

Figure 3.6 : Algorithme de Checkpointing [HMR,1997]

7. CONCLUSION

Nous avons passé en revue dans ce chapitre les différentes techniques de checkpointing dans les systèmes répartis conventionnels. Nous avons abordé la

classification des algorithmes de checkpointing et le détail de deux protocole, l'un coordonné non-bloquant et l'autre communication-induced checkpointing. Ceci nous permettra d'entamer dans le chapitre suivant, les techniques de checkpointing en environnement mobile et voir la possibilité d'étendre les techniques abordées ci-dessus pour cet environnement.

CHAPITRE IV :

CHECKPOINTING POUR CALCUL MOBILE

1. INTRODUCTION

Les techniques de checkpointing abordées au chapitre III et appliquées directement dans un environnement mobile deviennent inefficaces et même inapplicable. Même certaines notions de défaillances ou de tolérances aux défaillances deviennent insensées. Par exemple dans un système distribué classiques les handoffs, mise en veille ou les déconnexions des MH peuvent être considérées comme des pannes et déclencheront un recouvrement arrière.

L'algorithme de checkpointing doit prendre en considération les contraintes de mobilités des MH (discutés dans le chapitre II) pour trouver un point de contrôle global cohérent.

Dans ce chapitre, nous allons voir comment réaliser des protocoles de checkpointing pour les systèmes répartis dans un environnement mobile.

2. SUPPORT DE STOCKAGE STABLE

Les points de contrôle doivent être sauvegardés dans un endroit sûr et qui leur permettra de rester accessibles même après défaillance. Il ne sert à rien si un checkpoint est perdu lors de la défaillance car son utilité est pour permettre au système de reprendre à cet état là. Par conséquent, il faut toujours l'enregistrer sur un support appelé *stockage stable* (*stable storage*).

Ceci peut être réalisé comme suit. Chaque point de contrôle sera enregistré sous forme de fichier sur un serveur de fichiers partagé. Le serveur doit être structuré de telle façon à optimiser les opérations d'écriture ce qui réduirait le temps de prise de checkpoint local. Dans le but de protéger le serveur des défaillances, il faut utiliser un serveur de sauvegarde (server backup) qui sera mis à jour régulièrement.

Le gestionnaire de fichier doit assurer un accès tolérant aux pannes aux fichiers des checkpoints. Il doit donc assurer la continuité de l'accès que ce soit en lecture ou écriture. Chaque checkpoint pourra être dupliqué sur des disques séparés et probablement

même sur des sites différents. Ceci nécessite une gestion des copies et des versions de backup. Le nombre de copies identiques de checkpoints dépendra de l'environnement (probabilité de crash).

A cause des fréquentes déconnexions, du mode mise en veille, de la limite de capacité de stockage, et la vulnérabilité du matériel portable, une station mobile ne peut pas être considérée comme un support stable pour enregistrer des points de contrôle. Une MH ne peut pas assurer l'accès continuelle aux fichiers de checkpoint à cause des multiples déconnexions. Une MH ne peut pas se permettre de sauvegarder plusieurs copies de checkpoint et des versions de backup en raison de la limite de son espace disque. En plus une unité mobile peut tomber en panne à tout moment et pourra perdre les points de contrôle enregistrés définitivement et rendre la reprise impossible.

3. PERFORMANCE DE CHECKPOINTING ET MOBILITE

Les techniques de checkpointing courantes ne peuvent pas être appliquées directement et avec facilité car les systèmes de calcul mobiles ont un ensemble différent de critères que les systèmes de calcul distribués classiques (voir chapitre II).

Ainsi la réalisation d'un bon algorithme de checkpointing pour un environnement mobile doit traiter les principaux problèmes de mobilité. Il doit prendre en considération certains aspects qui vont influer directement sur sa performance. Ci-dessous quelques points importants pour un checkpointing performant et efficace.

3.1 SURCOUT DU AU CHECKPOINT

Le principal critère de tout programme est le temps d'exécution. Pour un programme distribué, le surcoût du checkpoint est l'incrémentation du temps d'exécution du programme à cause de l'application de checkpointing. Ce surcoût est causé, en général, par le temps de prise du checkpoint, sa sauvegarde sur un support de stockage non volatile. Il est aussi du aux messages et données en relation avec le checkpointing. L'envoi et la réception de messages de synchronisation et les informations embarquées dans les messages de calcul qui vont ralentir la transmission. A cause de la faiblesse de la bande passante du réseau sans fil le temps de transmission est toujours plus grand sans oublier le temps de recherche d'une station mobile (Le message pour atteindre sa destination peut traverser le réseau sans fil et le réseau filaire, ce qui augmente le temps d'acheminement du message).

Pour la sauvegarde, qui consiste en le chargement (upload) des points de contrôle vers les MSS, il est possible d'utiliser les checkpoint incrémentaux qui consistent à transférer juste les informations qui ont changé depuis le dernier point de contrôle enregistré sur la MSS. Pour ce qui est des messages, il faut réduire au maximum les messages de coordination et la taille des données embarquées dans les messages de calcul. *Plus le temps d'exécution du protocole de checkpointing est réduit, meilleure sera la performance de l'algorithme.*

3.2 NOMBRES DE CHECKPOINTS

Parmi les critères d'un programme est l'espace nécessaire pour son exécution. D'autant que la ressource espace de stockage est limitée dans les MH. Pour obtenir un point de contrôle global cohérent dans un système de calcul mobile, il est nécessaire parfois pour une MH d'avoir à maintenir de multiples checkpoints. Ces copies doivent être sauvegardées sur des supports stables, c'est à dire transférés sur des MSS. Par conséquent, il faut diminuer leur nombre et ne garder que ceux qui sont vraiment nécessaires pour que l'état global reste cohérent. Sans oublier de réduire aussi la taille des structures de données utilisées. *Plus le nombre de checkpoints est réduit, meilleure sera la performance de l'algorithme.*

3.3 ECONOMISER L'ENERGIE

La consommation d'énergie est maximale lors de l'envoi et la réception de messages. La diminution de messages va permettre à la MH de sauvegarder de l'énergie. Les deux points précédents sont liés directement à celui là, car la diminution du temps de checkpointing et le nombre de checkpoints va permettre à la MH de gagner un temps pour ne pas consommer durant son énergie.

En général, une MH se met en mode veille si elle n'a pas d'activité locale et se réveillera à la réception de messages. L'envoi de messages (autres que ceux du calcul) vont déranger la MH en mode veille et déclenchera une consommation de son énergie. Il faut alors réduire les messages de contrôle à destination des MH pour éviter de les déranger et économiser de l'énergie

3.4 COUT DU RECOUVREMENT

Lors de l'occurrence d'une défaillance dans le système, une application de retour arrière (rollback) vers le dernier état cohérent enregistré est déclenchée. Cette application va se baser sur les résultats du checkpointing et dépendra de la technique du checkpointing. Elle est par exemple très coûteuse pour la classe de protocoles non coordonnés. Dans un environnement mobile, il conviendrait que l'application de retour arrière soit la plus simple possible et ce en utilisant aussi peu que possible de messages.

Un algorithme de checkpointing qui rend le recouvrement de faible coût est meilleur pour un environnement mobile vu qu'il permet d'économiser de l'énergie, la bande passante et remettra en marche le système le plus rapidement possible.

3.5 LATENCE DE LA COLLECTE DU CHECKPOINT GLOBAL

Les connexions et déconnexions risquent d'augmenter de façon significative le temps d'achèvement de la construction du checkpoint global cohérent. Cette application de collecte peut être lancée pour des raisons spécifiques à tout moment.

Puisque une MH ne peut fournir son point de contrôle en étant déconnectée, ceci oblige qu'un checkpoint doit être pris au moment de chaque déconnexion.

Ainsi, le point de contrôle pris avant la déconnexion va appartenir à n'importe quel état global cohérent collecté durant la période de déconnexion de cette MH. De cette façon, un algorithme ne retardera pas le temps de réponse du checkpointing.

3.6 NOMBRE DE PROCESSUS

L'algorithme de checkpointing doit être conçu de telle façon à s'exécuter quelque soit le nombre de processus participant au calcul distribué. Si ce nombre augmente, le checkpointing doit s'exécuter avec les mêmes performance et efficacité. Parfois, le nombre de MH n'est pas connu à l'avance. Il arrive aussi, durant le calcul, que ce nombre varie.

4. CLASSIFICATION DES TECHNIQUES DE CHECKPOINTING MOBILE

Les algorithmes de la classe des protocoles de checkpointing coordonnés peuvent assurer un point de contrôle global cohérent et nécessitent de maintenir la taille de deux états pour chaque processus. Toutefois cette technique force chaque processus dans le système de calcul mobile à prendre un nouveau checkpoint ou bloquer le calcul sous-jacent durant le checkpointing. Par ailleurs, il est possible que plusieurs messages soient envoyés pour que l'état global soit cohérent. Par conséquent, le surcoût de la coordination est important dans l'environnement mobile.

Les techniques de communication-induced checkpointing et non-coordonnés demandent à chaque processus à prendre un checkpoint indépendamment. Toutefois, chaque processus devra enregistrer plusieurs points de contrôle locaux dans un support de stockage stable. Sachant que les MH ne sont pas considérées comme des supports de stockage stables, le coût de ce genre d'algorithmes est trop élevé dans un environnement mobile. Les protocoles non-coordonnés ont en plus le désavantage que le coût de recouvrement est trop élevé car ça nécessite un grand échange de messages pour tracer une ligne de recouvrement (avec risque d'effet domino). Ceci rend cette technique impossible dans un environnement mobile [QCB,1998].

A partir de là, de nouvelles techniques sont nécessaires pour réaliser un checkpointing en 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 dans l'environnement mobile. Nous citerons ici les trois principales orientations.

4.1 CHECKPOINTING HYBRIDE

Higaki et Takizawa dans [HT,1998] ont proposé une nouvelle technique pour l'environnement mobile. L'algorithme intègre les avantages des protocoles de checkpointing coordonnés et non coordonnés conventionnels. L'algorithme est structuré en deux niveaux. Chaque MSS prend un point de contrôle local en utilisant un protocole synchrone. L'approche de collection de checkpoints par les MSS est considérée comme protocole coordonné. Les MH prennent leurs points de contrôle par un protocole asynchrone.

Cette technique est très intéressante, vu qu'elle prend en considération les problèmes de la mobilité et les MH sont traitée différemment. Parmi les défauts de cette technique est que le protocole peut causer l'effet domino et que les protocoles sur les MH et MSS ne négocient pas entre eux. Ainsi, il est difficile d'obtenir un état global cohérent [Lin,2001].

4.2 QUASI-SYNCHRONOUS CHECKPOINTING

Mannivan et Singhal on proposé une nouvelle technique qu'ils ont appelé checkpointing quasi-synchrone (quasi-synchronous checkpointing) [MS,1996].

L'algorithme présenté est simple et il a les mérites des techniques de checkpointing asynchrones (surcoût bas) et le mérite des protocoles de checkpointing synchrones (temps de recouvrement bas). Chaque processus prend des points de contrôle indépendamment. D'autres checkpoints sont pris à la suite de la réception de messages. L'index est incrémenté après chaque checkpoint local ou checkpoint forcé. Les points de contrôle forcés sont pris dans le but de faire avancer la ligne de recouvrement et ce lors de la réception d'un message contenant un index supérieur au sien.

Malgré que l'algorithme à un coût très faible, il nécessite d'enregistrer plusieurs points de contrôle.

Les algorithmes quasi-synchrones sont répartis en trois classes différentes : SZPF (Strictly Z-path Free) strictement sans Z-Chemin, ZPF (Z-Path free) sans Z-Chemin, ZCF (Z-Cycle free) sans Z-Cycle. Des détails sur cette classe de protocoles ainsi que l'analyse de performance seront trouvés dans [MS,1999].

4.3 MUTABLE CHECKPOINTING

Cao et Singhal, dans [CS,2001], ont proposé une nouvelle technique dans les algorithmes de checkpointing coordonnés qui permet de sauvegarder les points de contrôle sur n'importe quel support (pas nécessairement stable) comme la mémoire principale ou le disque dur de la MH. Ce genre de checkpoints, appelés points de contrôle mutables, ne sont ni une tentative de checkpoint ni un checkpoint permanent. Ceci permettra de diminuer le surcoût du transfert de données sur un support stable de stockage (sur une MSS). Ce genre de checkpoints est pris à la réception des informations embarquées dans les messages de calcul. Nous verrons ce protocole dans le détail un peu plus tard dans le prochain chapitre. Ce genre de protocole ne fait pas de différence entre

une MH et une MSS qui sont traitées de la même façon et risque de déranger les MH en mode veille par la réception de messages de contrôle.

5. CONCLUSION

Après avoir vu les techniques de checkpointing en environnement conventionnel en chapitre III, nous avons étudié dans ce chapitre les techniques de checkpointing en environnement mobile. Nous avons remarqué que les techniques conventionnelles ne deviennent plus efficaces. Nous avons alors essayé de classifier les protocoles de checkpointing destinés à l'environnement mobile. Ceci nous permettra d'aborder dans le prochain chapitre quelques travaux en détail. Nous essayerons de voir les avantages et les inconvénients de chaque protocole.

CHAPITRE V :

TRAVAUX ANTERIEURS

1. INTRODUCTION

Parmi les travaux les plus récents réalisés dans ce domaine nous citerons la notion de *mutable checkpoints* introduite par *Cao* et *Singhal* dans [CS,2001] et l'algorithme de *Manabe* dans [Man,2001]. Mais jusqu'à présent aucun algorithme n'a été implémenté dans la réalité.

Acharya et son groupe dans [ABI,1994] ont été les premiers à présenter un algorithme de checkpointing pour les systèmes de calcul mobile. Dans leur algorithme, il n'y a pas de coordination entre les processus. Une MH prend un checkpoint local quand une réception de message est précédée par un envoi de message dans la même MH. Le nombre de checkpoint augmentera si chaque réception sera suivie d'une émission de message. Ceci peut dégrader les performances du calcul. D'autres protocoles ont été donné pour réduire le nombre de points de contrôle à sauvegarder sur des supports stables de stockage comme celui de Mannavan et Singhal [MS,1996]. Dans ce document une nouvelle orientation a été introduite, il s'agit d'algorithmes quasi-synchrone. Dans le même sens, un algorithme basé index a été présenté dans [QBC,1997] destiné aux unités mobiles. Il permet de réduire l'évolution de l'index du checkpointing, ce qui réduit le nombre de checkpoints forcés.

Dans le cadre des checkpointing coordonnés, Prakash et Singhal ont essayé d'associer deux notions qui étaient orthogonales auparavant [PS,1996] : associer un nombre minimum de processus et ne pas bloquer le calcul durant le checkpointing. Dans [CS,1998], il a été prouvé qu'il est impossible d'associer ces deux notions et ils ont présenté par la suite une nouvelle notion.

Une nouvelle orientation a été proposée par Cao et Singhal dans [CS,2001] qui est la notion de checkpoint mutable en se basant sur la technique de checkpointing coordonné. Par contre, Manabe dans [Man,2001] a proposé une technique structurée en deux-tiers basée sur index-based checkpointing. Une autre technique a été présentée par Lin dans [Lin,2001], qui consiste en un algorithme hybride qui se déroule en trois phases, la première coordonnée entre MSS et la deuxième basée index entre MSS et MH

et la dernière pour finaliser le point de contrôle globale entre MSS. Dans la suite de ce chapitre nous présenterons les détails des principaux travaux.

2. PREMIER PROTOCOLE DANS UN ENVIRONNEMENT MOBILE

Acharya, Badrinath et Imielinski ont proposé le premier protocole de checkpointing destiné à un système distribué avec des unités mobiles dans [ABI,1994]. L'algorithme est conçu de telle façon à prendre en charge les déplacements des MH et leurs déconnexions.

L'algorithme de checkpointing prend en considération les trois points suivant :

- Un dépôt stable pour les checkpoints locaux des MH.
- La déconnexion de une ou plusieurs MH ne doit pas prévenir le checkpointing global d'une applications s'exécutant sur une MH
- L'algorithme de checkpointing doit éviter de subir un surcoût de recherche ou d'information.

Le mécanisme pour les MH est asynchrone. L'idée principale est que chaque MH prend de façon indépendante ses états locaux comme checkpoints ensuite elle va les transférer vers la MSS locale. Quand la MH se déconnecte, son checkpoint local sera disponible dans une MSS.

La MH prend un point de contrôle à la suite de réception de message de calcul. Le point de contrôle local doit inclure chaque message *m* envoyé vers une autre MH si l'événement *send(m)* est inclus dans l'état global. Ce message est sauvegardé dans une structure appelée *log*. L'algorithme utilise deux phase (*SEND* et *RECV*). Une station MH notée h_i en phase *SEND*, à la réception d'un message *m*, elle prend un checkpoint et passe à la phase *RECV* avant de délivrer le message à l'application de calcul. Si h_i est en phase *RECV*, la priorité sera donnée pour l'envoi du message *m* à une autre MH, et passera à la phase *SEND*. Ces règles de deux phases assurent qu'aucune dépendance n'est créée entre deux points de contrôle locaux de la même MH. Les MH utilisent deux tableaux. *LOC* et *CKPT*. $LOC_i[j]=k$ représente que la MH h_i voit que la MH h_j est dans la cellule de la MSS s_k. $CKPT_i[j]=k$ représente que la MH h_i voit que la MH h_j a pris le point de contrôle numéro *k*.

La MSS sera chargée de collecter un ensemble de points de contrôle locaux mais ne participe pas au calcul. Il faut s'assurer par la suite de construire un checkpoint global cohérent correspondant à une MH h_i dans une de ses cellules. La MSS initie la calcul de l'état global en envoyant à toutes les cellules demandant pour chaque MH h_j le checkpoint local ayant pour numéro de séquence $CKPT_i[j]$. Si le point de contrôle de h_j est disponible à la MSS locale ($LOC_i[j]$), il répondra immédiatement à l'initiateur avec le point de contrôle local en question, sinon, il attendra que h_j prenne son prochain checkpoint.

Actions Lorsque MH envoie un message M

```
If phase=RECV then phase=SEND;
send(M,CKPT[],LOC[])
append M to log
```

Actions Lorsque MH reçoit un message M d'une autre MH

```
If phase=SEND then checkpoint ;
phase=RECV;
for j<=|MH| and j>=1 do
      CKPT[j]:=M.CKPT[j]
      LOC[j]=M.LOC[j]
Deliver message M
```

```
checkpoint ;
Changer de MSS vers MSS p ;
LOC[i] := p ;
```

Lors du handoff pour une MH
Avant une déconnexion volontaire d'une MH
Procédure checkpoint

Figure 5.1 : Algorithme de checkpointing [ABI,1994]

```
Record local_state ;
Transfer log, CKPT[], LOC[] to the local MSS;
CKPT[i] := CKPT[i]+1;
```

Dans ce protocole, deux vecteurs d'entiers doivent être embarqués sur chaque message de calcul. La taille des vecteurs dépend du nombre de MH. Ceci peut ralentir l'envoi des messages de calcul, à cause de la bande passante des canaux sans fil. En même temps, comme on l'a indiqué plus haut si les événements d'émission et de réception se succèdent l'un après l'autre, il y aura plusieurs points de contrôle qui seront pris et sauvegardés sur un support de stockage stable. Le coût deviendra trop élevé à ce moment là.

3. PROTOCOLE QUASI-SYNCHRONE DE MANNIVAN ET SINGHAL

Mannivan et Singhal ont présenté dans [MS,1996] un nouveau protocole qui peut bien être adapté aux systèmes distribués en environnement mobile. Ce protocole faisant partie de la classe des protocoles quasi-synchrone est un mélange entre le checkpointing indépendant non coordonné et le checkpointing basé index.

Chaque processus P_i a deux variables entières qui sont sn_i et $next_i$.

- sn_i représente le numéro séquentiel du dernier checkpoint pris par P_i.
- $next_i$ représente le numéro séquentiel du prochain checkpoint local spontané qui sera pris par P_i. Cette variable est incrémenté par le processus chaque x unités de temps. L'intérêt de la variable $next_i$ est de garder les numéros séquentiels des

derniers checkpoints des processus proches entre eux, ce qui permettra lors du recouvrement d'aider à la progression de la ligne de recouvrement.

Les stations mobiles vont stocker leurs points de contrôle sur les stations statiques, puisque leurs disques durs ne sont pas stables. Mannavan et Singhal proposent pour cela d'utiliser une structure maintenue au niveau de chaque MH afin de localiser la MSS où est sauvegardé chaque point de contrôle. Cette structure appelée *location_info* est un ensemble où chaque entrée est un triplet de la forme (*pid*, *checknum*, *location*) où *pid* est l'identificateur du processus, *checknum* est l'index du point de contrôle et *location* est l'identificateur de la MSS où est sauvegardé le point de contrôle numéro *checknum*.

Le détail du protocole est donné ci-dessous (figure 5.2).

Actions Lors de l'initialisation pour P_i

```
next_i. :=1 ;
sn_i := 0 ;
```

Actions Lorsque P_j reçoit un message M de P_i

```
If M.sn > sn_j then ;
      Take Checkpoint C;
      C.sn := M.sn;
      sn_j :=C.sn
Process M
```

Actions Lorsque P_i envoie un message M

```
M.sn := sn_i ;
Send(M);
```

Actions Lorsqu'il est temps pour P_i d'incrémenter $next_i$

```
next_i := next_i +1;
```

Actions Lorsqu'il est temps pour P_i de prendre un checkpoint local spontané

```
if next_i > sn_i  then
      Take Checkpoint C;
      C.sn := next_i;
      sn_j :=C.sn;
```

Figure 5.2 : Algorithme de checkpointing [MS,1996]

Le grand problème de ce protocole est la sauvegarde d'un grand nombre de points de contrôle et surtout pour les MH qui doivent transférer à chaque fois leur nouveau point de contrôle vers la MSS ce qui augmentera le trafic sur le réseau sans fil. Ceci nécessitera aussi une application de ramasse-miettes (garbage collector). Ce protocole ne fait pas de différence entre processus s'exécutant sur MH ou sur MSS et de ce fait, il ne prend pas en considération les cas de déconnexion et handoff. L'avantage de ce protocole est que le recouvrement proposé dans [MS,1996] est simple et sans effet domino. L'autre avantage est l'absence de messages de contrôle qui peuvent déranger les MH en mode veille.

4. PROTOCOLE BASE INDEX POUR MH DE QUAGLIA ET AL.

Quaglia et son groupe ont proposé un protocole qui optimise un autre protocole basé index en réduisant les différences entre les numéros de séquence des points de contrôle. Ceci permettra alors de réduire le nombre de points de contrôle forcés.

Un point de contrôle forcé est pris quand un message reçu contient un numéro de séquence supérieur à celui enregistré localement. A chaque déconnexion ou handoff un point de contrôle est pris ce qui fait avancer la numérotation des points de contrôle.

La procédure d'initialisation

```
Procedure init
        sn_i := 0;
        rn_i := -1;
        checkpointing; % basic checkpoint %
```

Actions Lorsque une MH h_i envoie un message m à une MH h_j

```
m.sn := sn_i;
send(m) to h_j ;
```

Actions Lorsque une MH h_i reçoit un message m

```
rn_i := max(m.sn; rn_i);
If m.sn > sn_i then
    begin
      sn_i := m.sn;
      checkpointing; % forced checkpoint %
    end;
```

```
If rn_i = sn_i then sn_i := sn_i + 1;
checkpointing; % basic checkpoint %
```

Actions Lorsque une MH h_i change de cellule

Actions Lorsque une MH h_i se déconnecte du réseau

La procédure de checkpointng

Figure 5.3 : Algorithme de checkpointing [QCB,1998]

```
If rni = sni then sni := sni + 1;
```

```
Procedure checkpointing
        take a checkpoint C_{i;sni} ;
```

Pour une MH h_i, en plus de la variable locale de numérotation séquentielle des points de contrôle locaux sn_i, une variable rn_i est utilisée pour contrôler l'évolution de la numérotation. Elle permet à la station mobile de savoir lequel de ses points de contrôle locaux peut remplacer son prédécesseur dans la ligne de recouvrement. Ainsi, ils ont montré que si $rn_i < sn_i$ lors de la prise d'un point de contrôle spontané *C*, alors C ne dépend d'aucun point de contrôle dans la ligne de recouvrement avec un numéro de séquence sn_i. Dans ce cas, le numéro de ce point de contrôle est mis à sn_i. Ce protocole mène à une augmentation lente des sn_i.

Ce protocole souffre des problèmes des protocole de checkpointing induits communication. Le recouvrement est très complexe et peut même mener à un effet

domino. Le protocole aussi oblige la station mobile à chaque prise de point de contrôle (spontané ou forcé) à l'envoyer vers un support de stockage stable. Sachant qu'à chaque déconnexion et handoff, un point de contrôle forcé peut être pris, ceci augmentera le trafic sur le réseau sans fil surtout si la mobilité des stations mobiles est importante. De plus, le protocole ne prend pas en considération le calcul qui se déroule sur les nœuds statiques et les considère juste comme de simples relais. L'avantage de ce protocole est l'absence de messages de contrôle pour la synchronisation qui peuvent déranger les MH mises en mode veille.

5. PROTOCOLE COORDONNE DE PRAKASH ET SINGHAL

Nous présentons dans cette section l'algorithme de checkpointing proposé par Prakash et Singhal dans [PS,1996] qui a pour objectif d'impliquer le minimum possible de processus dans le calcul de checkpoint global et sans bloquer le calcul sous-jacent. Ce protocole peut être classé comme protocole coordonné.

A cause de la faiblesse de la source d'énergie et la bande passante, il faut diminuer les messages de contrôle et ainsi, le protocole doit forcer un nombre minimum de processus à prendre leurs points de contrôle locaux. L'échange de messages implique une grande consommation d'énergie. Pour arriver à cet objectif, le protocole utilise des messages de contrôle pour coordonner les processus et utilise aussi l'information de contrôle embarquée sur les messages de calcul pour diminuer le nombre de checkpoints forcés.

Afin de prendre en charge la mobilité, tout processus mobile P_i doit calculer son *état local* avant de se déconnecter d'un site statique SS_j et l'enregistrer sur ce site avec les variables utilisées dans le calcul du point de reprise global. Au moment de la *déconnexion*, le site SS_j stockera à son niveau les messages reçus par P_i. Si l'un de ces messages est une requête pour le calcul de son état, le site SS_j doit le faire pour P_i en envoyant l'état enregistré avant la déconnexion et mettre à jour les variables locales.

Lorsque P_i se *reconnecte* sur un autre site statique SS_k, il doit envoyer une *requête* à travers ce dernier pour SS_j. SS_j envoie les messages destinés à P_i et en cas de calcul du point de reprise global durant la déconnexion, SS_j envoie aussi l'état et les variables. Ceci permettra d'exécuter le calcul des points de reprise globaux sans faire de distinction entre *processus mobiles* et *processus statiques*.

Nous pourrons mieux connaître le fonctionnement de l'algorithme (illustré par la figure 5.4) à travers le fonctionnement des différentes structures de données utilisées dans l'algorithme et décrites ci-dessous.

- ***interval_number*** : Variable *entière* stockée *localement* par *chaque processus* et véhiculée dans tous les *messages de calcul* et dans les *messages de requêtes de calcul du point de reprise*. Cette variable est initialisée à 1 et incrémentée à chaque calcul du point de reprise localement.
- ***Send_weight*** et ***recv_weight*** : Variables *réelles* stockées *localement* par *chaque processus* et véhiculées dans les *messages de requête de calcul du point de reprise* et dans le *message de réponse* envoyé à l'initiateur de l'algorithme. Sa

valeur varie entre 0 et 1, elle est initialisée à 0 et mise à 1 par le processus initiateur.

- ***weight*** : Variable *réelle* stockée localement par le *processus initiateur* du calcul du point de reprise global. initialisée à 0 et mise à jour par l'addition des valeurs de terminaison reçus dans les messages de réponse. Si cette variable est égale à 1 alors le programme se termine.
- $\boldsymbol{R_i}$: *Tableau* de *n bouléen* ou *n* représente le nombre de processus du système participant au calcul distribué. Il est stocké *localement* par *chaque processus* et véhiculé dans les *messages de calcul* et les *messages de requêtes*. Il permet de détecter les dépendances causales entre les processus. Il est de taille de *n bits* et nécessite des opérations logiques qui sont assez rapides et moins coûteuses. Ce tableau est initialisé comme suit :
  $R_i[i] := 1$; et $R_i[j] := 0$ pour tout $j := 1$ à n et $j <> i$
  Il est mis à jour à la réception d'un message m comme suit : $R_i[j]:=m.R_i[j]$ *OR* $R_i[j]$
  où *m.R* est le tableau envoyé dans le message, *OR* est l'opérateur logique de bits (*ou*). Ce tableau est réinitialisé après chaque calcul du point de reprise localement. Le fait qu'une valeur $R_i[j]$ soit égale à 1, veut dire que P_i a une dépendance causale avec P_j. Les processus ayant une dépendance causale uniquement qui participeront dans le calcul du nouveau point de reprise global. Pour les autres, c'est les anciens états qui seront pris en considération. Ceci diminuera le nombre de processus participant dans le calcul de l'état global.
- ***Interval_vector*** : *Tableau* de *n entiers* stocké *localement* par *chaque processus*. Ce tableau est initialisé à 1.
  Interval_vector*[i] := Interval_number* et ***Interval_vector****[j]* correspond à la variable ***interval_number*** reçu par P_i dans le dernier message de P_j.
- ***first*** : *Tableau* de *n bouléens* stocké *localement* par *chaque processus*. initialisé à 0 à chaque calcul de point de reprise et *first[j] = 1* si P_i envoie un message de calcul à P_j. Il permet à P_i de savoir s'il a contacté les autres processus par un message de calcul depuis le dernier point de reprise.
- ***trigger*** : *enregistrement* composé de *deux champs*. L'*identificateur du processus initiateur* et la *valeur de la variable* ***Interval_number*** *du processus initiateur* du calcul de l'état global.
- ***propagate*** : *Tableau* de *n bouléens* stocké *localement* par *chaque processus*, lui permettant de savoir à quel processus, il va faire suivre la requête de calcul du point de reprise.
  propagate[j] := $R_i[j]$ *AND NOT (m.R[j])*

La requête sera envoyée aux processus ayant une dépendance causale avec P_i et pas avec le processus ayant envoyé la requête.

L'algorithme considère que les canaux de communication sont FIFO.

Actions Lorsque P_i envoie un message de calcul à P_j

```
    if first[j]=0 then { first[j]:=1; send(P_i , message, R_i , interval_number, own_trigger);
}
```

```
    clear first; rfirst:=0;        take_local_snapshot;        weight:=1.0;
    own_trigger.pid:=own identifier(P j );     increment(interval_number);
    own_trigger.inum:=interval_number;     increment(interval_vector[j]);
    to all nodes P_i ,such that R[i]=1 {
        weight:=weight/2; send_weight:=weight;
        send(initiator_id, REQ, R_j , interval_number, own_trigger, send_weight);}
    reset all bits, except own bit, in R_j;        resume normal computation;
```

Action pour initier le checkpointing par P_j
A la réception par les processus P_i de la requête de la part de P_j
Actions de P_i à la réception d'un message de calcul de la part de P_j
La procédure de propagation

Figure 5.4 : Algorithme de checkpointing [PS,1996]

Il a été prouvé par Cao et Singhal que cet algorithme peut mener à une situation d'incohérence. Ils ont montré qu'il est impossible de réaliser un algorithme coordonné de checkpointing qui ne bloque pas le calcul sous-jacent tout en forçant seulement un

```
    receive(P_j , REQ, m.R, interval_number', msg_trigger, recv_weight);
    if msg_trigger = own_trigger then {
        to all nodes P_k ,such that Propagate[k]=1
            {  recv_weight:=recv_weight/2; send_weight :=recv_weight;
               send(P_i , REQUEST, R_i , interval_number, own_trigger, send_weight);}
```

```
    receive(P_j , message, m.R, interval_number' , msg_trigger);
    if interval_number' <= interval_vector[j] then process the message and exit;
    else { interval_vector[j]:= interval_number' ;
          if msg_trigger.pid = own_trigger.pid then
           {   if msg_trigger.inum = own_trigger.inum then process the message;
               else{  propagate snapshot(R_i , m.R, P_i , interval_number, msg_trigger,0);
                      process the message; rfirst:=1;} }
          else{ if rfirst = 0 then
                  { propagate snapshot(R_i , m.R, P_i , interval_number, msg_trigger,0);
                    process the message; rfirst:=1;}
          else process the message;} };
```

nombre minimal de processus à prendre leurs points de contrôle. Les détails de cette preuve sont donnés dans [CS,1998]. De plus cet algorithme souffre des défauts des techniques de checkpointing coordonné dans l'environnement mobile et du fait que les MH et MSS sont traitées de la même façon ce qui engendre un surcoût de recherche des MH.

6. POINTS DE CONTROLE MUTABLES DE CAO ET SINGHAL

Cao et Singhal ont présenté une nouvelle orientation [CS,2001] qui fait suite au travail de [PS,1996] détaillé dans la section précédente et suite au travaux de [CS,1998]

qui ont montré l'impossibilité de checkpointing dans ce cas là. Le travail est considéré donc comme une amélioration du protocole de [PS,1996].

L'idée est de réaliser un checkpointing coordonné tout en exploitant les messages de calcul pour transférer certaines informations liées au checkpointing. Ainsi une tentative de checkpoint peut être prise à la suite d'une requête transmise par un message de contrôle. D'autres points de contrôle peuvent être forcés à la suite après comparaison des informations reçue et les données locales.

Les points de contrôle pris à la suite de la coordination et ceux pris à la suite de la réception de messages de calcul ne sont pas semblables. Certains points de contrôle pris à la suite des messages de calcul ne sont pas toujours utiles.

De là vient la notion des points de contrôle mutables (mutable checkpoints) qui ne sont ni une tentative de checkpoint, ni un checkpoint permanent. Ils permettent de réaliser un algorithme de checkpointing coordonné plus efficace pour les systèmes de calcul distribués mobiles. Ces points, créés à la suite de la réception de messages de calcul, ne sont pas sauvegardés sur un support de stockage stable. Ils peuvent être enregistrés par exemple sur la mémoire principale ou le disque dur de la MH. Il est ainsi possible d'éviter les dépassements dus aux transferts de grandes quantités d'informations de la MH vers un support de stockage stable sur une MSS en utilisant le réseau de communication sans fil. Le checkpoint mutable peut changer en une tentative de checkpoint. Par exemple, un processus P_i qui a pris un checkpoint et quand il reçoit une requête de checkpointing il fait le transfert du mutable checkpoint vers un support de stockage stable et force tous les processus dépendants à prendre des tentatives de checkpoint. Si P_i ne reçoit pas la requête après la fin du checkpointing, il se débarrasse du point de contrôle mutable.

Le traitement de la mobilité se fait de la même façon que dans [PS,1996] et le calcul des points de reprise est réalisé sans faire de distinction entre *processus mobiles* et *processus statiques*.

L'algorithme (détails dans figure 5.5) est non bloquant basé sur le concept des points de contrôle mutables. Pour mieux comprendre le fonctionnement de l'algorithme, ci dessous sont définies des structures de données utilisées. Chaque processus garde à son niveau deux tableaux et plusieurs variables.

- ***R_i*** : *Tableau* de *n bits*. Il est stocké *localement* par *chaque processus* et véhiculé dans les *messages de calcul* et les *messages de requêtes*.
  $R_i[j] := 1 \Rightarrow P_i$ reçoit un message de calcul de la part de P_j.
  Ce tableau permettra au processus de savoir si un processus dépend de lui ou non.
- ***csn_i*** : *Checkpoint sequence number*. *Tableau* de *n entiers* stocké *localement* par *chaque processus* P_i. $csn_i[j]$ représente le numéro de séquence du checkpoint de P_j selon la vision de P_i. P_i le met à jour à chaque réception d'un message de P_j. Ce tableau est initialisé à *0* partout.
- ***weight*** : Variable *réelle* positive initialisée à *.0* et ne dépassant pas 1. Elle permet de détecter la terminaison du checkpointing
- ***trigger$_i$*** : *enregistrement* composé de *deux champs*.
  L'*identificateur du processus initiateur(pid).*

Le champs *inum qui représente le numéro de séquence du checkpoint* chez *pid* lorsqu'il a pris son propre point de contrôle.

- ***sent*$_i$:** variable booléenne, ne prendra la valeur 1 que si un message est reçu par P_i durant l'intervalle de checkpoint courante.
- ***cp_state*$_i$** : variable booléenne, mise à 1 si P_i est dans le processus de checkpointing. Cette varaiable est initialisée à 0.
- ***old_csn*** : variable utilisée pour sauvegarder le *csn* du checkpoint (tentative ou permanent) courant.
- ***CP*$_i$** : Enregistrement à 4 champs au niveau de chaque P_i :
  mutable: le *mutable checkpoint* de P_i.
  R : le vecteur booléen proper à P_i avant de prendre son *mutable checkpoint* courant.
  trigger : la variable trigger associée au *mutable checkpoint* courant.
  sent : valeur de la variable *sent* de P_i avant de prendre son *mutable checkpoint* courant.
- ***msg_trigger*** : représente $trigger_j$ reçu par P_i dans la requête envoyée par P_j.
- ***own_trigger*** : représente $trigger_i$ de P_i.
- ***req_csn*** : est le *csn* qui est attaché à la requête.
- ***MR*** : Est une structure où sont sauvegardés *R* est *csn* pour les envoyer attachés à la requête.

L'algorithme considère que les canaux de communication sont FIFO.

Le problème majeur est que l'algorithme ne fait pas de distinction entre les processus s'exécutant sur des MH et ceux s'exécutant sur des MSS. Le nombre total de processus doit être connu dès le départ. L'utilisation de messages de coordination peut diminuer considérablement de la performance du checkpointing. Une MH peut recevoir un message de contrôle alors qu'elle est en mode veille ou déconnectée. Une MH doit envoyer à son tour la requête à plusieurs processus dont certains qui ne sont pas nécessairement dans sa cellule ce qui augmente les délais de transmission.

Action pour initier le checkpointing par P_j

increment($csn_j[j]$); $own\_trigger:=(P_j ; csn_j[j])$; $Cp\_state_j:=1$; $weignt_j:=0$;
for $k := 0$ to N ***do*** $MR[k].csn := 0$; $MR[k].R := 0$;
$MR[j].csn:=csn_j[j]$; $MR[j].R := 1$; $prop\_cp(R_j ; MR;P_j ; own\_trigger, 1.0)$;
take a local checkpoint (on stable storage); $old\_csn_j:= csn_j[j]$; $sent_j :=0$; reset R_j;

Actions chez P_i à la réception de la requête de la part de P_j

receive(P_j ; request; MR; recv_csn; msg_trigger; req_csn; recv_weight);
$csn_i[j]:=recv\_csn$;
if $old\_csn_i > req\_csn$ ***then send***(P_i ; reply; recv_weight) ***to*** the initiator; return;
$Cp\_state_i:=1$;
if $msg\_trigger=own\_trigger$
then if $CP_i.trigger = msg\_trigger$
then $prop\_cp(CP_i.R; MR;P_i ; msg\_trigger; recv\_weight)$;
save CP_i.mutable on stable_storage; $old\_csn_i:=csn_i[i]$; $CP_i:=NULL$;
send(P_i ; reply; $weight_i$) ***to*** the initiator;
else send(P_i ; reply; recv_weight) ***to*** the initiator;
else $increment(csn_i[i])$; $own\_trigger:=msg\_trigger$;
$prop\_cp(R_i;MR;P_i;msg\_trigger;recv\_weight)$;
take a local checkpoint (on stable storage); $old\_csn_i:= csn_i[i]$;

Actions Lorsque P_i envoie un message de calcul à P_j

if $Cp\_state_i = 1$ ***then*** ***send***(P_i , message, $csn_i[i]$; own_trigger); $sent_i := 1$;
else ***send***(P_i , message, $csn_i[i]$; NULL); $sent_i := 1$;

Actions chez P_i à la réception d'un message de calcul de la part de P_j

receive(P_j ; m; recv_csn; msg_trigger);
if $recv\_csn <= csn_i[j]$ ***then*** $R_i[j]:=1$; process the message;
else if $csn_i[msg\_trigger.pid]=msg\_trigger.inum$
then $csn_i[j]:=recv\_csn$; $R_i[j]:=1$; ***process the message***;
else $csn_i[j]:=recv\_csn$;
if $msg\_trigger<>NULL$ ***and*** $sent_i=1$ ***and*** $msg\_trigger<>own\_trigger$
then ***take a local checkpoint, save*** it in CP_i.mutable; $CP_i.R:=R_i$;
$CP_i.trigger:=msg\_trigger$; $CP_i.sent:= sent_i$; $sent_i:=0$; reset R_i;
if $msg\_trigger <>NULL$ ***and*** $Cp\_state_i=0$
then $Cp\_state_i:=1$; ***increment***($csn_i[i]$); $own\_trigger:=msg\_trigger$;
$R_i[j]:=1$; ***process the message***;

Actions dans la seconde phase pour l'initiateur P_i

receive(P_j; reply; recv_weight); $weight_i := weight_i + recv\_weight$;
if $weight_i=1$ ***then*** $Cp\_state_i := 0$; ***broadcast***(commit; msg_trigger);

Actions chez processus P_j en recevant un message broadcast

receive(commit; msg_trigger); $csn_j[msg\_trigger.pid]=msg\_trigger.inum$; $Cp\_state_j:=0$;
if $CP_j.trigger=msg\_trigger$ ***and*** $CP_j<>NULL$ ***then*** $sent_j:=sent_j \cup CP_j.sent$; $R_j:=R_j \cup CP_j.R$; $CP_j:=NULL$;
if there is a tentative checkpoint associated with msg_trigger, ***make it permanent***;

La procédure de propagation

prop_cp(R_i; MR;P_i; msg_trigger; recv_weight)
$weight_i := recv\_weight_i$;
for $k:=0$ to N ***do*** $temp[k].csn:=max(MR[k].csn; csn_i[k])$; $temp[k].R := max(MR[k].R;R_i[k])$;
for any P_k, such that ($R_i[k] = 1$) ***and*** $(max(MR[k].csn; csn_i[k]) <> MR[k].csn)$
$weight_i := weight_i =2$; ***send***(P_i; request, temp, $csn_i[i]$; msg_trigger; $csn_i[k]$; $weight_i$);

Figure 5.5 : Mutable checkpoints [CS,2001]

L'algorithme aussi ne traite pas le cas des multiples initiatives de checkpointing concurrentes. Selon [CS,2001], lorsqu'un processus reçoit une requête de checkpoint (alors qu'il est en checkpointing) soit il l'ignore tout simplement ce qui bloquera cette initiative ou il la diffère après la fin de l'algorithme sans donner les détails de mise en oeuvre.

7. PROTOCOLE HYBRIDE DE LIN

Lin a proposé un protocole hybride qui s'exécute en trois phases qu'il a appelé PQP Checkpointing. La première phase appelé pré-synchronisation consiste à une coordination entre MSS. La MSS initialise le protocole en diffusant la requête à toutes les autres MSS. La deuxième phase appelée Quasi-synchrone se déroule entre MSS et les MH locales. Cette phase consiste à ne pas envoyer directement la requête aux MH. Elle déclenche un timer T_{quasi} et durant ce temps là, la MSS profitera pour transférer la requête à chaque MH qui recevra un message de calcul. La requête est transférée grâce au numéro de séquence du checkpoint. La troisième phase appelée post-synchronisation, elle commence lors de la fin du temps T_{quasi}. Durant cette phase, la MSS enverra la requête à toutes les MH se trouvant dans sa cellule et n'ayant pas reçu un message de calcul durant l'intervalle de temps T_{quasi}.

Procédure de checkpointing

```
Procedure checkpoint_here(i)
        Take checkpoint C_a,i;
        Send checkpoint C_a,i to MSS_p ;
```

Actions lorsque MH_a reçoit un message de calcul

```
Receive(MH_b, Mh_a, im, m);
If im > i then
        checkpoint_here(im);
        i := im;
Process message m;
```

Actions lorsque MH_a reçoit la requête

```
Receive(MSS_p, Mh_a, im, request);
If im > i then checkpoint  here(im) and i := im;
```

Actions lorsque MH_a envoie un message de calcul m à MH_b

```
Send computation(MHa, MHb, im, m) to MSSp;
```

Actions lorsque MH_a se déconnecte de MSS_p

```
checkpoint_here(i + 1);
Send disconnect(MH_a, MSS_p) to MSS_p;
```

Actions lorsque MH_a se reconnecte MSS_q

```
Send reconnect(MH_a, MSS_q, MSS_p) to MSS_q;
```

```
Send leave(MH_a , MSS_p) to MSS_p;
Send join(MH_a , MSS_q, MSS_p) to MSS_q;
```

Actions lors du handoff pour MH_a (déplacement de MSS_p vers MSS_q)

Figure 5-6 : Protocole PQPCkpt pour MH [Lin,2001]

Actions d'initialisation pour MSS_p
Actions lorsque MSS_p initialise le checkpointing ($MSS_p = MSS_c$)
Actions lorsque MSS_p reçoit la requête de MSS_c
Actions lorsque T_{quasi} arrive à terme pour MSS_p
Actions lorsque MSS_p reçoit un message de calcul
Actions lorsque MSS_p reçoit un point de contrôle $C_{a,i}$ de MH_a
Actions lorsque MSS_p reçoit une réponse à la requête de MSS_q
Actions lorsque MSS_p reçoit un message commit du coordinateur MSS_q=MSS_c
Actions lorsque MH_a quitte MSS_p
Actions lorsque MH_a rejoint MSS_p après avoir quitté MSS_q
Actions lorsque MSS_p reçoit un message de handoff de la part de MSS_q
Actions lorsque MH_a se déconnecte de MSS_p

i, RC : integer (: = 0);
MQ, RQ, DQ, TC, GC, DB : Queue (: = ∅);

i := i + 1; ∀q, send (MSS_p, MSS_q, i, request) to MSS_q, where 1≤q≤Ns - 1;

receive (MSS_c, MSS_p, im, request)
if im > i then Set a coordinator timer T_{quasi};
i := im; RQ:=MQ;

Send (MH_a, MSS_p, request) to MH_a, ∀MH_a∈RQ;

receive (MH_a, MH_b, im, m);
if im > i then i := im; else im := i;
if MH_b∈RQ then delete it from RQ;
if MH_b in the cell of MSS_p then deliver the message to MH_b;
else search MH_b and deliver the message to MH_b's MSS;
if MH_b ∈DQ then put the message into DB;

Actions lorsque MH_a se reconnecte à MSS_p après avoir déconnecté de MSS_q

receive (MH_a, MSS_p, $C_{a,i}$);
if $C_{a,i}$ ∉TC then Add $C_{a,i}$ into TC else replace it;
if ∀b, $C_{b,i}$ ∈TC and MH_b ∈MQ then send reply(MSS_p, MSS_c) to coordinator;

receive (MSS_q, MSS_p, reply); RC = RC + 1;
if RC = Ns - 1 then ∀q, send committed(MSS_p, MSS_q) to MSS_q, where 1≤q≤Ns - 1; RC

receive (MSS_q, MSS_p, commit);
GC := TC; TC = ∅;

receive (MH_a, MSS_p, leave); Delete MH_a from MQ;

receive (MH_a, MSS_p, MSS_q, join); Add MH_a into MQ;
if MSS_q ≠ Null then Send(MSS_p, MSS_q, MH_a, handoff) to MSS_q;

receive (MSS_q, MSS_p, MH_a, handoff);
Send a temporary checkpoint $C_{a,i}$, $C_{a,i}$ ∈ TC, to MSS_q;
Send a permanent checkpoint $C_{a,i}$, $C_{a,i}$ ∈GC, to MSS_q;

receive (MH_a, MSS_p, disconnect); Add a member MH_a into DQ;

receive (MH_a, MSS_p, MSS_q, reconnect);
if p ≠q then Send a temporary checkpoint $C_{a,i}$, $C_{a,i}$∈TC, from MSS_q;
Send a permanent checkpoint $C_{a,i}$, $C_{a,i}$ ∈GC, from MSS_q;
Deliver all messages of MH_a in the DB of MSS_q to MSS_p;

Figure 5-7 : Protocole PQPCkpt pour MSS [Lin,2001]

Le protocole sera finalisé après la réception par chaque MSS des points de contrôle de toutes les MH locales en les enregistrant sur un support de stockage stable en tant que checkpoints permanents. Par la suite, chaque MSS doit envoyer un message de réponse favorable à la requête à la MSS qui joue le rôle de coordinateur. Cette dernière va envoyer un message *commit* pour annoncer la sortie du protocole de checkpointing et en recevant ce message les MSS doivent rendre les tentatives de checkpoints comme points de contrôle permanents.

Les MH utilisent une variable entière notée *i* qui représente l'index du dernier point de contrôle pris. Les MSS utilisent six files :

- *MQ* : liste des MH locales à la MSS.

- *RQ* : liste des MH locales n'ayant pas reçu un message de calcul depuis le lancement du timer.
- *DQ* : liste des MH déconnectées.
- *DB* : file des messages envoyés à des MH déconnectées.
- *TC* : les tentatives de checkpoints des MH locales.
- *GC* : les checkpoints permanents des MH locales.

Cet algorithme ne tient pas compte du calcul qui se déroule sur les MSS et ne calcule que le point de contrôle global incluant les processus s'exécutant sur les MH. Un autre problème peut surgir lorsque une MH se déconnecte durant la deuxième ou la troisième phase du protocole et qui peut mener à une inconsistance. Parmi les insuffisances est le non traitement des initiatives de checkpointing concurrentes et seules les MSS qui ont le droit d'avoir cette initiative malgré le fait qu'elles ne participent pas au calcul distribué. Par contre, ce protocole a réussi à joindre l'avantage des protocoles coordonnés et les protocoles basés index grâce aux trois phases. Parmi ces avantages, une station mobile n'est pas tout le temps dérangée par les messages de contrôle et le nombre de points de contrôle sauvegardés sur le stable storage.

8. PROTOCOLE BASE INDEX A DEUX NIVEAUX DE MANABE

Nous présentons dans cette section l'algorithme de checkpointing proposé par Y. Manabe dans [Man,2001] qui a conçu un protocole basé index architecturé en deux-tiers. En d'autres termes, il différencie dans le traitement les MH des MSS pour faire face aux contraintes dues à la mobilité.

procédure mhcheckpoint

take a checkpoint; ck := ck + 1; see:=sck;

ck := 0; gcn := 0; sck := -1; see := -1;

procédure d'initialisation

Actions lorsque h prend un checkpoint spontanné

Actions lorsque h envoie un message m à une MSS s_j

mhcheckpoint; gcn := gcn + 1; cgc(gcn) := ck;
send(m, gcn, ck, see) to s_j ;

Actions lorsque h reçoit un message m d'une MSS s_j

```
receive (m, mgcn, mck, mhsee)
if mhsee = ck then mhcheckpoint;
for each y (gcn<y ≤mgcn) do cgc(y) := ck;
sck := max(sck,mck);  gcn := max(gcn,mgcn);
process m;
```

Actions lorsque h se connecte à une MSS s_j

```
set current MSS be sj;   send (connect, gcn) to sj;
```

Actions lorsque h se déconnecte d'une MSS s_j

```
set current MSS be null; send (disconnect, gcn) to sj;
mhcheckpoint; /* handoff checkpoint */
see := -1;  sck := -1;
```

Figure 5-8 : Protocole basé Index pour MH [Man,2001]

procédure d'initialisation

```
ck[i] := 0;          for each k(≠i) do ck[k] := -1;
for each k do        lck[k] := -1;        gcn[k] := 0;        see[k] :=false;     st[k] :=false;
for each h ∈ Hi do           hgcn[h] := 0;      hck[h] := -1;       hsee[h] := -1;       hst[h] :=false;
for each h ∈ Hi and k do     hlck[h,k] := -1;   hlgcn[h,k] := -1;
```

```
take a checkpoint; ck[i] := ck[i]+1;
see[i] :=false;      for each k(≠i) do see[k]:=true;
for each k do st[k] :=false;
for each h ∈ Hi do hst[h] :=false; hsee[h] := hck[h];
```

procédure checkpoint

procédure testcondition(ngcn)

Actions lorsque s_i prend un checkpoint spontanné

Actions lorsque s_i envoie un message m à une MSS s_j

Actions lorsque s_i envoie un message m à une MH h $\in H_i$

```
send(m, gcn, ck, see) to h ;    st[j] :=true;
```

```
send(m, gcn[i], ck[i], hsee[h]) to h;  hst[h] :=true;
for each k do hlck[h,k] := ck[k]; hlgcn[h,k] := gcn[k];
```

```
   for each y (gcn[i]<y ≤ngcn) do cgc[y]:=ck[i];
gcn[i] := max(gcn[i], ngcn);
```

Actions lorsque s_i reçoit un message m d'une MSS s_j

```
receive (m, mgcn, mck, msee) from MSS s_j
for each k do
   if ck[k] = mck[k] then     see[k] := see[k] OR msee[k]
        else if ck[k] < mck[k] then     see[k] := msee[k];
   ck[k] := max(ck[k], mck[k]);       gcn[k] := max(gcn[k], mgcn[k]);
testcondition(mgcn[j]);                process m;
```

Actions lorsque s_i reçoit un message m d'une MH h

```
receive(m, mgcn, mck, msee) from MH h;
gcn[h] := max(gcn[h], mgcn[h]);
if gcn[i]<mgcn[h] then
   if msee=ck[i] or (∃k, st[k] and gcn[k]<mgcn[h]) or (∃h' ∈Hi, hst[h'] and hgcn[h']<mgcn[h']) then checkpoint;
   for each y (gcn[i]<y ≤mgcn[h]) do cgc[y]:=ck[i];
hck[h]:= max(hck[h], mck);          gcn[i]:= max(gcn[i], mgcn[h]);          process m;
```

```
receive(disconnect, mgcn) from MH h;
hgcn[h]:= max(hgcn[h], mgcn);       testcondition(hgcn[h]);
Hi = Hi – {h};       HI[h] := {hlck[h],       hlgcn[h]};
```

Actions lorsque h se déconnecte de s_i

Actions lorsque h se connecte à s_i

Figure 5-9 : Protocole basé Index pour MSS [Man,2001]

Ce protocole est intéressent du fait qu'il dérange le moins possible les stations

```
receive(connect, mgcn) from MH h; Hi = Hi ∪{h};
receive (hlck[h], hlgcn[h]) from h's previous MSS;
hsee[h] := -1;     hst[h] :=false;
for each k do      if ck[k] ≤hlck[h,k] then   see[k] :=true;
for each k do  ck[k]:= max(ck[k],   hlck[h,k]); hlck[h,k] := -1;  gcn[k] := max(gcn[k], hlgcn[h,k]);
hgcn[h] := mgcn;                   testcondition(hgcn[h]);
```

mobiles. L'inconvénient majeur de ce protocole est que à chaque déconnexion ou handoff un point de contrôle est pris dans un support de stockage stable. Ceci peut gêner la mobilité des MH qui doivent à chaque fois transférer une quantité importante d'information vers la MSS. De plus le protocole en traitant chaque handoff ou déconnexion, la MSS doit bloquer tout calcul en attendant de récupérer certaines information de la nouvelle MSS. Ceci va limiter les performances du calcul sur les stations fixes.

9. CONCLUSION

Nous avons passer en détail quelques nouvelles orientations dans le checkpointing en environnement mobile. Les principaux protocoles réalisés jusqu'à présent pour ce domaine ont été expliqués. Nous avons vu les approches de chacun d'eux et analysé chaque travail. Chaque nouvelle classe a ses avantages et aussi ses inconvénients. Nous remarquons par exemple que le protocole de [CS,2001] sera plus coûteux si le nombre de processus augmente. Donc, il est plus adapté à des systèmes avec un nombre de processus réduit. Le travaux de [QCB,1998] ou [Lin,2001] traitent le système comme si les nœuds statiques ne participent pas au calcul. Maintenant que nous connaissons, les critères de performances des protocoles de checkpointing en environnement mobile et les points qu'il faut prendre en charge nous allons proposer dans le chapitre suivant un protocole de checkpointing.

CHAPITRE VI :

PROPOSITION D'UN PROTOCOLE

1. INTRODUCTION

Dans ce chapitre, nous présenterons notre contribution qui consiste en un nouveau protocole hybride de checkpointing. Mais avant de présenter la solution, nous décrivons le système de calcul. Nous considérons le système comme une collection de processus qui communiquent uniquement par échange de messages. La majorité de ces processus s'exécutent sur des stations mobiles (MH) dont le nombre n'est pas connu au préalable et qui peut changer durant l'exécution du calcul distribué. Les stations mobiles, pour communiquer, passent par les stations fixes qui joueront le rôle de supports pour les MH. Le système obéit aux caractéristiques des systèmes distribués et de l'environnement mobile décrits aux chapitres I et II.

2. PREMIERE APPROCHE

En résumé de ce qu'on a vu jusqu'à présent, un protocole de checkpointing doit :

(1) Réduire le nombre de messages de synchronisation avec les MH.

(2) Réduire les informations transmises via les messages de calcul.

(3) Déranger le moins possible une MH car elle peut être en mode veille pour économiser l'énergie.

(4) Ne pas dépendre du nombre de processus participant au calcul et plus précisément du nombre de MH.

(5) Prendre en considération le fait qu'une MH peut se déconnecter à tout moment volontairement ou involontairement.

(6) Prendre en charge les handoffs.

(7) Sauvegarder le minimum possible de points de contrôle par MH.

(8) Assurer un recouvrement facile et sans coût élevé et sans effet domino.

(9) Décharger la MH de la tâche de coordination, permettra en plus d'économiser de l'énergie et la bande passante.

(10) Gérer les initiatives concurrentes multiples.

(11) Diminuer le nombre de points de contrôle envoyés vers le support de stockage stable.

Il est très difficile, voire impossible, d'arriver à joindre tous les points dans un même protocole. Il faut donc trouver un consensus pour approcher au maximum chacun des points cités ci-dessus.

A partir de là, la meilleure façon de réaliser un bon checkpointing efficace pour les systèmes distribués en environnement mobile est de décharger complètement la partie mobile du réseau de cette application.

Tous les messages envoyés de et vers une MH passent obligatoirement par la cellule à laquelle elle appartient. La MSS qui gère la cellule, pourra se charger de cette opération à la place de la MH. Elle aura la possibilité, à partir des messages reçus et envoyés (par la MH), de calculer toutes les dépendances et trouver un checkpoint global cohérent contenant les états des MH sans que ces dernières participent au calcul. Ceci répondra à tous les onze points.

Mais l'état local d'un processus n'est pas seulement les événements d'émission et de réception, mais consiste aussi à l'état des variables locales liées à l'évolution localement de l'application distribuées sous-jacente. Par conséquent, il faut toujours impliquer les MH dans le checkpointing tout en laissant le soin aux MSS de se charger du calcul des dépendances pour impliquer le minimum possible de processus dans le checkpointing. Ceci réduirait considérablement le surcoût du checkpointing du fait que la partie fixe du réseau supportera une bonne partie du protocole de checkpointing. Ceci contribuera pour se rapprocher des points (3), (4), (5), (6) et (9).

A partir de là, la tâche de synchronisation doit être donnée aux MSS ce qui fera passer la complexité du coût de synchronisation de $O(|MH|)$ à $O(|MSS|)$, sachant que généralement $|MH| >> |MSS|$. Une MH voulant réaliser la synchronisation doit envoyer sa requête à sa MSS locale qui héritera de la requête et se chargera du reste. Ceci règlera définitivement le point (9) et contribuera à atteindre les points (3), (5) et (6).

Pour réaliser le point (1), on peut utiliser la notion de points de contrôle mutables pour pouvoir enregistrer certains checkpoints localement. Pour transférer ces points de contrôle vers une MSS pour les sauvegarder sur un support de stockage stable, il faut profiter des messages de calcul pour que la MSS envoie la requête.

Une MSS initiatrice du protocole doit envoyer la requête à toutes les autres MSS et ensuite chaque MSS doit envoyer la requête aux MH locales (connectées à la MSS). Ceci permettra d'économiser de l'énergie et faire avancer rapidement le checkpointing vu que la bande passante du réseau filaire est nettement supérieure à celle de la MH. Il permettra aussi de réaliser particulièrement le point (4) et toucher les points (1), (2) et (5).

Pour le problème des initiatives de checkpoints concurrentes (10), le fait de donner uniquement la possibilité de diffuser la requête de checkpointing aux MSS contribuerait à réduire le nombre de processus concurrents. Mais pour régler ce problème définitivement, au lieu que les initiatives soient concurrentes, elles peuvent contribuer pour former un point de contrôle global cohérent. Il a été montré dans [Mat,1989] que

l'union de points de contrôle cohérents formeront un point de contrôle cohérent. Pour cela les points de contrôle pris localement doivent être numérotés séquentiellement (à l'aide d'une variable qu'on nommera *csn*) de telle façon que deux checkpoints ayant le même *csn* appartiendront au même point de contrôle global cohérent.

L'utilisation de la technique de checkpointing coordonné permettra de réaliser le point (8). Pour le point (11), il faut utiliser la technique de checkpointing incrémental. Ce point peut être approché en n'envoyant pas à chaque événement de déconnexion ou handoff le point de contrôle vers la MSS comme c'est le cas dans [Man,2001] ou [Lin,2001].

Utiliser la technique de checkpointing indépendant chez les MH donnera plus de liberté à la MH pour choisir le moment de prise de son point de contrôle.

L'algorithme doit se dérouler en deux phase l'une en utilisant la technique de checkpointing coordonné et l'autre la technique de checkpointing indépendant ou induit communication.

3. PROTOCOLE A DEUX-PHASES

A partir de l'analyse faite ci-dessus, nous pouvons conclure que l'algorithme doit être structuré en deux niveaux (two-tier structured), c'est à dire qu'il y aura un protocole pour les MSS et un autre pour les MH. Ceci nous aidera à faire supporter la plus grande charge du protocole de checkpointing à la partie fixe du réseau. L'algorithme doit aussi se dérouler en deux phases (algorithme hybride). Une phase se déroule sans la coordination et qui consiste à enregistrer les points de contrôle localement (indépendamment ou suite aux informations de contrôle reçues) et la seconde pour coordonner et sauvegarder un point de contrôle sur un support de stockage stable.

3.1 PHASE NORMALE

Cette phase consiste à ce que chaque processus prenne à sa propre initiative des points de contrôle spontanés comme dans le cas du checkpointing indépendant. Ceci permettra une simplicité du protocole et une certaine liberté pour les MH en choisissant le bon moment pour prendre leurs checkpoints. Durant cette phase, pour assurer la cohérence, certains points de contrôle forcés sont pris suite à la réception de certaines informations. de contrôle transmises via les messages de calcul. Cette synchronisation est obligatoire.

Pour éviter de transférer à chaque fois une grande quantité d'informations vers le support de stockage stable se trouvant sur la MSS, nous proposons que les points de contrôle pris lors de cette phase soient sauvegardé localement (c'est à dire dans la mémoire ou le disque dur de l'unité mobile). Ces points de contrôle sont semblables aux checkpoints mutables [CS,2001], ils vont permettre d'éviter de gaspiller de la bande passante du réseau sans fil.

La combinaison dans cette du checkpointing indépendant et le checkpointing induit communication nous fait rappeler l'approche des protocoles quasi-synchrones [MS,1996]

[MS,1999]. Ainsi, à chaque point de contrôle pris localement est assigné un numéro séquentiel à travers la variable local *csn*.

La variable *csn* sera incrémentée à la pise d'un point de contrôle spontané et si lke processus a reçu un message depuis le dernier checkpoint pris. Un processus, en recevant un message de calcul comportant une valeur de *csn* supérieure à la sienne, il prendra un checkpoint forcé et mettra à jour sa variable *csn*.

Ceci nécessitera de sauvegarder plusieurs points de contrôle localement et pour optimiser l'espace de stockage, nous proposons d'utiliser la technique des checkpoints incrémentaux. C'est à dire entre deux checkpoints on n'enregistrera que les changements intervenus.

3.2 PHASE COORDINATION

Le principal avantage des techniques de checkpointing coordonné est le nombre de points de contrôle sauvegardés localement qui est au maximum deux (tentative et permanent). Ceci simplifierai le protocole de recouvrement et éviterai l'effet domino. Nous voulons exploiter ces avantages dans notre solution.

Cette phase consiste donc à synchroniser les différents processus pour récolter un état global cohérent dans un support de stockage stable à partir des points de contrôle enregistrés localement. Pour diminuer le nombre de messages de coordination vers les unités mobiles, cette phase va se dérouler en trois étapes. La première consiste à réaliser la coordination entre MSS. La deuxième étape se déroule au niveau de chaque MSS. Pour éviter le surcoût du à la recherche d'une station mobile, chaque MSS synchronisera uniquement les MH qui sont dans sa cellule. La dernière étape consiste à ce que chaque MSS informe le processus coordinateur de la fin de la sauvegarde des tentatives de points de contrôle sur un support de stockage stable et le coordinateur finalisera le protocole par l'envoi d'un message de type *commit* pour rendre toute tentative de point de contrôle permanente.

Ceci sera bénéfique si la coordination se limiterai aux MSS seulement. De là, le processus coordinateur doit être sur une MSS. Si une MH veut initier la coordination, elle chargera sa MSS locale du rôle de coordinateur. Une MSS coordinatrice diffuse la requête à toutes les autres MSS à travers les canaux de communication filaires. Chaque MSS doit prendre son checkpoint local et doit informer les MH. Pour éviter que les MH soient dérangées, nous pouvons nous inspirer de la technique utilisée dans [Lin,2001] et qui consiste à attendre un délai donné *T* avant de transmettre aux processus mobiles dans les cellules de la MSS. Durant ce temps, certaines MH pourront recevoir un message de calcul et la MSS pourra profiter pour transmettre la requête de checkpointing à travers ce message. Ce délai passé, il faut obliger les processus mobiles n'ayant pas reçu la requête à envoyer leurs checkpoints.

4. AMELIORATION

La première phase (phase normale) utilise une technique de checkpointing qui nécessite beaucoup d'espace de stockage même en utilisant le checkpointing local incrémental. Il faut donc trouver le moyen pour réduire leur nombre en développant une fonction de ramasse-miettes (garbage-collector) simple.

On sait que les protocoles quasi-synchrones donnent un point de contrôle global cohérent en faisant l'union de tous les points de contrôle locaux ayant le même numéro. Ainsi, si le plus petit numéro du dernier checkpoint local (*csn* local) pris dans le système est *x*, on peut alors construire un point de contrôle global cohérent correspondant *x*. Pour les autres points de contrôle *y* avec $y < x$ sont plus anciens et ne servent plus à rien et il est possible de libérer leurs places.

L'information du plus petit nombre *x* peut être déduite par les MSS à travers les messages de calculs échangés dans le système et qui transitent par ces mêmes MSS.

De même, après la coordination et la récolte du checkpoint *x* dans un support de stockage stable, on peut déduire que tous les points de contrôle de numéro inférieur à *x* ne sont plus nécessaires et ils peuvent être supprimés.

Après avoir réglé le problème de stockage des points de contrôle, il nous reste à fixer le temps *T*. Nous proposons qu'il soit fixé selon le système. Il ne doit pas être assez grand (tend vers l'infini) car le checkpointing deviendra quasi-synchrone et pas très court (tend vers zéro) car le checkpointing deviendra coordonné. Il est possible de prendre *T* selon des statistiques du système. Il correspondra, par exemple, au temps moyen pour qu'au moins 50% des MH dans la cellule reçoivent au moins un message de calcul.

Pour le problème des checkpoints concurrents, il est facile à régler. On sait que les checkpoints ayant même numéro de séquence forment le même checkpoint global cohérent. Supposant que deux processus initient le checkpointing en parallèle. Deux cas se présentent, soit ils vont initier avec le même numéro *csn* (la variable *csn* sera définie plus loin) ou l'un a un *csn* supérieur à l'autre. Pour le premier cas, ça ne pose pas de problème. Pour le second un processus sur MSS en recevant deux requêtes, il abandonnera celle qui a le plus petit *csn* pour construire un point de contrôle le plus récent possible. Pour diminuer le nombre d'initiatives concurrentes, tous processus se trouvant en processus de checkpointing, après avoir reçu une requête du coordinateur, ne pourra pas lancer une nouvelle coordination qu'après la fin du processus de coordination.

5. DEROULEMENT

Pour mieux comprendre le fonctionnement du protocole, voyons le déroulement dans un cas général.

1- Initialisation lors du début du calcul

2- Un processus qu'il soit mobile ou fixe prend des points de contrôle selon son besoin local. Chaque point de contrôle doit comporter un numéro *csn* (*checkpoint sequence*

number). Deux points de contrôle qui se succèdent sans qu'il y ait de réception de messages auront le même numéro.

3- Un processus prend un point de contrôle additionnel (forcé) s'il reçoit un message comportant un *csn* supérieur au sien.

4- Tout processus (s'exécutant sur MH ou MSS) peut lancer la coordination. S'il s'agit d'une MH, elle enverra un message à sa MSS qui s'occupera du reste.

5- Ainsi seules les MSS seront chargées de la coordination. Une MSS enverra à toutes les autres MSS la requête de checkpointing avec son *csn* avec une portion de sa variable locale réelle *weight* (Pour assurer qu'à tout moment la somme de toutes les variables *weight* est 1). Une MSS ne fera la coordination que si elle n'est pas en processus de checkpointing.

6- Une MSS en recevant la requête, elle doit vérifier si elle est en checkpointing avec un *csn* supérieur à celui reçu. Dans ce cas, elle enverra un rejet à l'initiateur.
Si les deux *csn* sont égaux, elle sauvegardera dans une liste l'adresse de cet initiateur qui est un concurrent.
Dans le dernier cas (son *csn* est inférieur à celui reçu), elle enverra un rejet à l'ancien et entamera de nouveau le checkpointing pour le nouveau *csn*.
Si la MSS n'est pas en processus de checkpointing elle rentre alors en processus de checkpointing.

7- La MSS qui commencera le checkpointing pour un *csn* donné *x* reçu avec l'initiative prendra son checkpoint enregistré localement de numéro *x*, s'il n'est pas disponible, elle prendra le premier qui a un numéro supérieur à *x*. Ce checkpoint sera enregistré sur un support de stockage stable comme tentative. Mais juste avant, elle lance le compte à rebours avec le timer *T*. Tout message de calcul reçu durant cet intervalle et destiné à un MH dans sa cellule sera utilisé pour transmettre la requête, ainsi les MH en mode de veille ne seront réveillés que par les messages de calcul. A la fin de l'intervalle *T* les processus n'ayant pas reçu de messages recevront un message de contrôle leur demandant d'envoyer le checkpoint.

8- Une MH en recevant la requête enverra le checkpoint de numéro *x* s'il est disponible sinon elle prendra le premier qui a un numéro supérieur à *x* et l'enverra au MSS pour être sauvegarder sur un support de stockage stable. Si le *csn* local est inférieur à *x*, la MH prendra un checkpoint à la réception de la requête et enverra ce checkpoint à la MSS. La MH vide les checkpoints enregistrés localement jusqu'au checkpoint envoyé.

9- Une MSS en recevant tous les checkpoints des MH dans sa cellule, elle enverra un message de réponse à la requête pour l'initiateur.

10- L'initiateur en recevant tous les messages de réponse diffuse un message *commit* qui permettra à toutes les MSS de rendre toute tentative de checkpoint comme point de contrôle permanent.

11- Une MH, avant de se déconnecter prendra un checkpoint local et enverra les checkpoints à la MSS qui les sauvegardera n'importe où pour lui permettre de remplacer le MH déconnectée lors de la construction du point de contrôle global. Si

le dernier checkpoint de la MH est inférieur ou égal à celui de la requête, ce checkpoint sera sauvegardé sur le support stable de la MSS car la MH ne pouvant recevoir aucun message durant la période de déconnexion ce qui ne changera pas la cohérence du point de contrôle global. Dans le cas contraire c'est le point de contrôle de numéro *x* ou le premier supérieur à *x* sera pris comme tentative.

6. DETAILS DU PROTOCOLE

Dans cette section, nous allons donner en détail le protocole de checkpointing. Nous présenterons le format des messages et les variables utilisées du coté des stations fixes et celles utilisées de la part des stations mobiles.

6.1 FORMAT DES MESSAGES

Tout message aura la forme (P_i, P_j, *m*, *controle*).

- P_i : est l'identificateur du processus émetteur.
- P_j : identificateur du processus destinataire du message.
- *m* : c'est le contenu du message.
- *controle* : représente les informations de contrôle embarquées sur le message.

Certains des messages qui circulent sont des messages de calcul et d'autres sont des messages de contrôle.

La partie *controle* d'un message de calcul est composée de :

- *m.csn1* : valeur du csn local
- *m.csn2* : valeur du plus petit csn du dernier checkpoint.
- *m.weight* : variable réelle permettant de détecter la terminaison.
- *m.ckpt* : variable entière. Elle est différente de 0 si la MSS veut demander à travers ce message de calcul à la MH d'envoyer son point de contrôle de contrôle correspondant au numéro *m.ckpt*.

Parmi les messages de contrôle, nous nous intéresserons à :

La déconnexion d'une MH d'une MSS : (P_i, P_j, *disconnect*, *C*) où *m.C* est les points de contrôle locaux de P_i.

La reconnexion d'une MH à une MSS (P_i, P_j, *reconnect*, *csn*, *MSS*) où *m.MSS* est l'ancienne station locale de la MH $m.P_i$ avant de se déconnecter.

Une MH qui quitte sa cellule pour rejoindre une autre (P_i, P_j, *leave*, *MSS*)

Une MH qui rejoint une cellule après avoir quitté une autre (P_i, P_j, *join*, *MSS*)

La requête de checkpointing (P_i, P_j, *request*, *csn*, *weight*)

La réponse à la requête (P_i, P_j, *reply*, *weight*)

Le rejet de la requête (P_i, P_j, *reject, csn*)

Le message de reconnaissance (P_i, P_j, *commit*)

6.2 STRUCTURES DE DONNEES POUR MSS

csn_i : c'est le numéro local du dernier checkpoint pris.

gcn_i : Un tableau, représentant la vue sur l'état des csn locaux des autres processus sur MSS.

min_csn_i : est la valeur minimale dans le tableau gcn.

T : est la durée de l'intervalle avant de lancer la coordination vers les MH.

$cell_i$ tableau de structures contenant *cell[i].pid* est l'identificateur de la MH dans la cellule i, $cell_i$ *[i].csn* est le csn local de cette *MH,* $cell_i$ *[i].connect* est une variable booléenne permettant de marquer la MH comme étant connecté ou non.

cp_state_i : pour indiquer si la MSS est en protocole de checkpointing ou non.

rcv_i : variable booléenne pour indiquer si le processus a reçu un message depuis son dernier checkpoint ou non.

6.3 STRUCTURES DE DONNEES POUR MH

Pour un processus P_i s'exécutant sur une station mobile il maintient les variables suivantes :

csn_i : c'est le numéro local du dernier checkpoint pris.

rcv_i : variable booléenne pour indiquer si le processus a reçu un message depuis son dernier checkpoint ou non.

C_i : est une structure permettant de sauvegarder les points de contrôle.

$last_i$: numéro du dernier point de contrôle sauvegardé sur un support stable.

MSS_i : est l'identificateur de la MSS locale à la MH.

Procédures et fonctions

Les procédures ***checkpoint(k)*** *et* ***garbage_collector(k)*** sont identiques à celles des MH ci-dessous

Procedure insert_MH (k, pid, csn, stable)

$cell_i[k].pid:=pid$; $cell_i[k].csn:=csn$;
$cell_i[k].connect:=true$; $cell_i[k].C:=NULL$;
$cell_i[k].stable:=stable$; $cell_i[k].ckpt:=false$;

Function search_MH(pid) : integer ;

k:=0; Search k such that $cell_i[k].pid:=pid$; Return k;

Procedure init

$rcv_i := false$; $csn_i:=0$; $weight_i:=0$; $cp\_state_i:=0$;
for all k do $cell_i[k].pid:=NULL$;
for all l do $gcn_i[l]:=0$;
take checkpoint $C_i[0]$;

Procedure init_ckpt

If $cp\_state_i<>0$ return;
$Weight_i := 1$; $cp\_state_i:=csn_i$; take permanent checkpoint $C_i[csn_i]$
for all MSS P_k do weighti:=weighti/2; send(P_i, P_k, request, csn_i, $weight_i$);
Set timer T;

Actions lorsque P_i prend un point de contrôle spontané : *identique à une MH*

Actions lorsque P_i initialise le checkpointing ou reçoit la requête receive (P_j, P_i, request, csn, weight);d'une MH P_j

init_ckpt;

Actions lorsque P_i reçoit la requête d'une MSS P_j

receive (P_j, P_i, request, csn, weight);
if $cp\_state_i<>0$ and $m.csn<=csn_i$ then send(P_i, P_j, reject, m.csn);
if $cp\_state_i<>0$ and $m.csn<=csn_i$ then send(P_i, P_j, reply, m.weight);
if $cp\_state_i=0$ then $weight_i:=m.weight$; initiator:=P_j Set timer T;
$cp\_state_i:=m.csn$;

Actions lorsque T arrive à terme pour P_i

$mw_i:=weight_i$;
for all k do if ($cell_i[k].pid<>NULL$) and ($cell_i[k].connect$) and ($cell_i[k].stable<cp\_state$)
then mw:=mw/2; send(P_i, P_k, request, cp_state_i, mw_i);

Actions lorsque P_i reçoit un point de contrôle C d'une MH P_j

receive(P_j, P_i, C, weight);
$mw_i:= mw_i + m.weight$;
save C in stable storage as permanent;
if ($mw_i = weight_i$) and ($weight_i = 1$) then save all tentative checkpoints as permanent;
for all MSS P_k do send(P_i, P_k, commit);
if ($mw_i = weight_i$) and ($weight_i <> 1$) then send(P_i, initiator, reply, m.weight);

Actions lorsque P_i reçoit une réponse à la requête d'une MSS P_j

receive(P_j, P_i, reply, weight);
$weight_i:= weight_i + m.weight$;
save C in stable storage as permanent;
if ($mw_i = weight_i$) and ($weight_i = 1$) then For all MSS P_k do send(P_i, P_k, commit);
if ($mw_i = weight_i$) and ($weight_i <> 1$) then send(P_i, initiator, reply, m.weight);

Procédure de checkpointing

```
Procedure checkpoint(k)
    If (k>csn_i) then    csn_i:=k;    Take incremental checkpoint C_i[csn_i];
```

```
Procedure upload(k)
    If (k=0) then send(P_i, MSS_i, disconnect, C_i);
            else send(P_i, MSS_i, C_i[k], weight_i);
```

Procédure de chargement des points de contrôle vers la MSS
Procédure de ramasse-miettes
Procédure d'initialisation
Actions lorsque P_i prend un point de contrôle spontané
Actions lorsque P_i reçoit un message de calcul

```
rcv_i := false;    csn_i:=0;    MSS_i :=P_i;    weight_i:=0;
```

```
If rcv_i then    rcv_i := false;    x:= csn_i + 1
    else    x:=csn_i;
checkpoint(x);
```

```
weight_i := m.weight;
If m.csn1 >csn_i then
    If rcv_i  then checkpoint(m.csn)
        else  C_i[m.csn1]:=C_i[csn_i];    delete C_i[csn_i];    csn_i :=m.csn1 ;
x:=m.csn2;
if (m.ckpt>0) then    x :=Min { x / exists C_i[x] and x≥ m.ckpt };
```

```
send (P_i, MSS_i, request, csn_i, 1);
```

```
garbage_collector(x);    rcv_i:=true;    process m;
```

Actions lorsque P_i initie le checkpointing
Actions lorsque P_i reçoit la requête
Actions lorsque P_i envoie un message de calcul m à P_j

```
receive(MSS_i, P_i, request, csn, weight);
weight:=m.weight;
```

```
If rcv_i then    x := csn_i;
checkpoint(x);    upload(0);
```

```
upload(x);    last_i:=m.ckpt;    garbage_collector(x);    rcv_i:=true;
```

```
Send (P_i, P_j, m, csn_i, last_i, 0, 0);
```

Actions lorsque P_i se déconnecte
Actions lorsque P_i se reconnecte P_j
Actions lors du handoff pour P_i (déplacement de MSS_i vers P_j)

```
Send (P_i, MSS_i, leave, P_j);    MSS_i := P_j;
Send (P_i, MSS_i, join, P_j, csn_i, last_i);
```

Figure 6.2 : Protocole pour MH

7. PREUVE DE COHERENCE

7.1 COHERENCE

Pour prouver la cohérence du point de contrôle global récolté dans le support de stockage stable nous utiliserons une démonstration par l'absurde.

Supposons que l'algorithme résulte un point de contrôle global *C* non consistant. On suppose que ce point a pour numéro *x*. Cela veut dire qu'un processus n'a pas enregistre l'émission d'un message donné *M* et un autre processus a enregistré l'événement de réception du message.

Le message est envoyé par P_i donc avec l'information embarquée *y* où *y* est le *csn* local de P_i. pour que le checkpoint ne fasse pas partie de *C*, *y* doit être supérieur de *x*. P_j recevra *M* avec et son *csn* local *z*. Lors de la réception, P_j comparera *y* et *z*,

Si $z<x$ alors il doit prendre un point de contrôle forcé car logiquement $z<y$ avec comme csn *y* avant de délivrer le message. Le prochain checkpoint après la réception du message *M* aura une valeur supérieur à *y* car il y a eu un événement de réception depuis le dernier checkpoint. Ainsi l'événement de réception de *M* ne sera pas enregistré dans l'état global qui a pour numéro *x*. (*contradiction avec notre hypothèse*)

Si $z=x$ .cela veut dire que le checkpoint qui a pour numéro *x* a été pris par P_j et en recevant le message *M*, il doit d'abord prendre un checkpoint de numéro de numéro *y* car là aussi $y>z$ et ensuite délivrera le message. Donc l'événement de réception de *M* ne fera pas partie de *C*. (*contradiction avec notre hypothèse*)

Si $z>x$ Il est clair qu'aucun problème ne se produira et le point de contrôle *x* a été enregistré avant de recevoir *M*.

7.2 TERMINAISON

La terminaison de la coordination est détectée par les variables réelles dont la somme totale ne dépassera pas 1comme c'est le cas dans [PS,1996] et [CS,2001]. Pour notre protocole nous utilisons d'une variable réelle *weight*. Elle permet de détecter la fin de coordination vu que le nombre de processus n'est pas connu à l'avance. Cette variable est initialisée à 1. A chaque fois que la requête est envoyée à un processus, la variable est divisée par 2, une portion de la variable est envoyée avec la requête et l'autre est sauvegardée. A chaque réponse à la requête cette portion est envoyée avec la réponse à l'initiateur. Si tous les processus ayant reçu la requête répondront, la MSS aura en fin de compte, en additionnant toutes les valeurs reçue la valeur 1. Ainsi elle pourra envoyer un message de type *commit* pour que les processus enregistrent leurs tentatives de points de contrôle comme points de contrôle permanents.

8. LA REPRISE APRES DEFAILLANCE

Pour recouvrir après qu'un nœuds tombe en panne, le système doit être redémarré à partir d'un état cohérent et après le calcul peut continuer. Les algorithmes de recouvrement après défaillance pour systèmes distribués statiques n'ont pas pris en considération les contraintes des réseaux mobiles. Dans certains de ces algorithmes, si un nœud P_i tombe en panne, il aura à retourner en arrière jusqu'à son point de contrôle local $C_{i,k}$ sauvegardé sur un support de stockage stable. Il doit par la suite demander à tous les autres processus P_j de redémarrer à partir d'un point de contrôle $C_{j,l}$ cohérent avec $C_{i,k}$.

L'utilisation de la coordination facilitera la tâche de la reprise après défaillance. Ainsi, nous sommes assurés que les points de contrôle locaux permanents enregistrés sur les MSS forment un point de contrôle global cohérent. Donc le calcul de cohérence lors du recouvrement n'est plus nécessaire.

Nous proposons que le processus défaillant téléchargera son point de contrôle *C* enregistré sur un support de stockage stable et diffusera un message à tous les processus pour faire un retour arrière. Il doit aussi remettre les variables de checkpointing à leurs valeurs lors de la prise de *C*. Si le processus défaillant est une unité mobile, il demandera à son MSS locale de se charger de la diffusion de sa requête de retour arrière ce qui permettra à la MH d'économiser de l'énergie. Pour éviter les coûts de recherche des MH, la MSS envoie la requête de retour arrière uniquement aux nœuds statiques. Chaque nœud statique se chargera d'envoyer la requêtes aux MH qui se trouvent dans sa cellule.

Chaque processus recevant un message de retour arrière doit faire un retour arrière à *C* et remettre à jour les valeurs de ses variables.

Actions chez une MH P_i après défaillance

Download *last C_i from stable storage;*
Rollback *to C_i;*
Send (rollback) to local MSS;

Actions chez une MSS P_i après défaillance

Rollback *to last C_i in stable storage;*
Foreach *MSS P_k*
    Send*(rollback) to P_k;*
Foreach *MH P_j in cell*
    Send*(rollback,C_j) to P_j;*

Actions chez une MSS P_i lors de la réception du message rollback

receive*(rollback);*
Rollback *to C_i in stable storage;*
Foreach *MH P_j in cell*
    Send*(rollback,C_j) to P_j;*

Download *last C_i from stable storage;*
Rollback *to C_i;*

Actions chez une MH P_j lors de la réception du message rollback

Figure 6.3 : Protocole de retour arrière

9. ANALYSE DE PERFORMANCE

L'utilisation d'un protocole structuré en deux-tiers (deux niveaux) le rend très efficace dans un environnement mobile [BAI,1993b] [BAI,1994]. Avec cette technique les MH ne recevront jamais un message de coordination de la part des MH ou des MSS non locaux. Ceci supprimera le surcoût du à la recherche d'une MH et en même temps réduira le nombre de messages de coordination. Ceci n'est pas le cas du protocole proposé par Cao et Singhal dans [CS,2001].

L'utilisation d'un protocole hybride nous permet de tirer profit des avantages des techniques de checkpointing coordonnés (nombres de points de contrôle sauvegardés sur le support de stockage stable et un recouvrement simple et sans effet domino) et des techniques non coordonnés (déranger le moins possible les MH, le nombre de messages de coordination et traitement des initiatives concurrentes). Ce qui n'est pas le cas des protocoles proposé dans [CS,2001], [Man,2001] et [Lin,2001].

Sauvegarder des points de contrôle localement nous permet de diminuer le transfert d'information à travers le réseau sans fil comme c'est le cas de [Man,2001] et [Lin,2001].

Le protocole que nous proposons ne dépend pas du nombre de stations mobiles et leur nombre peut varier à tout moment durant l'évolution du calcul distribué sans affecter le protocole de checkpointing. Ceci n'est pas le cas pour les algorithmes proposés par [CS,2001] et [Lin,2001].

Notre protocole est non bloquant ce qui n'est pas le cas du protocole proposé par [Man,2001]. Les stations fixes ne sont pas réduites à de simples routeurs comme c'est le cas dans [Lin,2001].

Protocoles	Notre algo.	[CS,2001]	[Man,2001]	[Lin,2001]
checkpoints enregistrés sur stable storage	2	2	plusieurs	2
MH dérangées	peu	trop	non	peu
Dépend du nombre de MH	non	oui	non	oui
Messages de coordination	=|MSS|	≤ |MH|+|MSS|	0	=|MSS|
Information embarquée	peu	trop	peu	peu
Recouvrement	simple	simple	compliqué	simple
Effet domino	non	non	possible	non
Architecture en 2 tiers	oui	non	oui	oui
Exécution du calcul sur MSS	oui	oui	oui	non
Initiatives concurrentes	oui	non	oui	non

Tableau 6.1 : Tableau comparatif avec les principaux travaux récents

10. CONCLUSION

L'utilisation de la coordination permet de concevoir un checkpointing qui garantira un retour arrière simple, facile et sans effet domino en cas de défaillance. Il permettra aussi de transférer sur un support de stockage stable les checkpoints locaux. Ce qui permettra d'éviter le surcoût du transfert à chaque prise de checkpoint.

L'utilisation de la technique basée index permet de concevoir un checkpointing plus efficace pour les MH qui prendront des checkpoints soit de façon indépendante soit en recevant certaines informations de contrôle leur permettant de décider de la prise du checkpoint.

L'utilisation de la notion de checkpoint mutable permet de gagner le temps de transfert du point de contrôle de la MH vers le support stable.

Une MH est coordonnée uniquement par sa MSS locale ce qui fait que aucun surcoût de recherche n'est rajouté. Comme elle peut recevoir l'ordre d'envoyer son checkpoint à travers un message de calcul.

Utiliser un algorithme de checkpointing coordonné dans un environnement mobile n'est pas très efficace. Mais en déchargeant la MH de la tâche de coordination, permettra en plus d'économiser de l'énergie et la bande passante. Lors des déconnexions, le

checkpointing ne sera pas bloqué. De même lors de la mise en mode veille, la MH ne sera pas réveillé par les messages de réponses à la requête et pour envoyer des messages de diffusion type *commit*.

Ceci va améliorer la performance tout en assurant que le checkpoint global sera cohérent. Mais la coordination pose toujours problème. Une MSS peut envoyer une requête à une MH et il y aura un coût de recherche de la MH en plus du coût de transfert de la requête de la MSS vers la MSS locale à la MH et le transfert de la MSS locale vers la MH. La MH pourra aussi être en mode veille au moment de la réception de la requête. Ceci peut causer une grande consommation d'énergie. En plus de tout cela s'ajoute la quantité d'informations transférée soit sur les messages de calcul ou les messages de coordination.

CONCLUSION

Le calcul mobile a émergé rapidement et beaucoup d'intérêt lui a été donné par les spécialistes. Comme ça a été présenté dans le chapitre II, la mobilité a introduit sur les systèmes distribués classiques de nouvelles contraintes qu'il fallait prendre en considération. Ceci a obligé les chercheurs de revoir plusieurs protocoles existants pour les adapter à ce nouvel environnement. Les algorithmes de checkpointing qui rentrent dans le domaine de la tolérance aux défaillances et qui sont liés directement aux protocoles de recouvrement (retour arrière), n'ont pas toujours été implémenté sur de vrais systèmes distribués.

Notre travail avait pour objectif d'étudier les systèmes distribués en environnement mobile et analyser les travaux dans le domaine de checkpointing réalisés jusqu'à présent pour ensuite trouver un protocole de checkpointing pour les systèmes distribués en environnement mobile.

Nous avons commencé par étudier les systèmes distribués et l'asynchronisme de ces systèmes et comment les représenter et leur évolution à travers des graphes de communications, le diagramme espace-temps ou utiliser certaines structures logiques pour concevoir des solutions à certains problèmes. Nous avons aussi vu les principales défaillances connues sur les systèmes distribués et les principaux outils de tolérance à ces pannes. Par la suite nous avons vu comment réaliser un ordre causal dans un environnement asynchrone et la représentation du déroulement des événement. Ceci nous a permis de faire le tour des systèmes répartis conventionnels et puis nous avons vu les différentes contraintes qui rentrent en jeu lorsque certains sites deviennent mobiles. Comme les handoffs, la source d'énergie, la bande passante, le stockage…etc. Ce qui rend l'utilisation des structures logiques impossibles. Nous avons donné une définition au modèle de calcul d'un système réparti en environnement mobile et vu comment concevoir des algorithmes efficaces pour un calcul dans un tel environnement. Après, nous avons abordé les protocoles de checkpointing. Nous avons vu la classification de ces protocoles en environnement conventionnel. Ainsi on distingue les protocoles coordonnées, non coordonnés, induit communication. Nous avons par la suite parlé du checkpointing en environnement mobile et nous sommes arrivé à la conclusion que les classes d'algorithmes de checkpointing conventionnel ne deviennent plus efficaces. Nous avons vu une nouvelle classification des algorithmes de checkpointing puis nous avons vu et analysé dans le détail quelques protocoles. Ceci nous a permis de trouver des approches qui nous ont permis de réaliser un protocole de checkpointing.

Le protocole proposé est un protocole hybride, se déroulant en deux phases, la première adoptant la technique quasi-synchrone et la seconde la technique coordonnée. Ceci nous a permis de tirer profit au maximum des avantages des deux techniques. Nous avons structuré le protocole en deux niveaux l'un pour les MSS et l'autre pour les MH et ce dans l'objectif de faire supporter à la partie fixe du système la charge du protocole. Nous avons donné les preuves nécessaires et comparé notre protocole aux plus récents travaux réalisés dans ce domaine.

Les objectifs tracés au début de la thèse ont été atteint avec le protocole présenté en chapitre VI. Nous proposons comme perspectives à ce travail, d'implémenter le protocole de checkpointing proposé sur un système distribué réel en environnement mobile. A notre connaissance aucun protocole de checkpointing n'a été implémenté en environnement mobile [Man,2001].

Nous proposons de développer un nouveau protocole qui rentre dans le cadre d'un outil global pour la tolérance aux pannes. Le checkpointing est lié à d'autres applications et au comportement du système lui même. Il faut concevoir de nouveau certains protocoles (handoffs, routage, …) pour leur permettre une meilleure communication avec le protocole de checkpointing.

Dans cet outil, Il est intéressant de penser à l'utilisation des agents mobiles. Ce sont des programmes pouvant apporter assistance à des entités et fonctionner de façon autonome. L'agent a la possibilité de migrer et se dispatcher (son exécution et ses données) sur plusieurs sites du réseau et pourra mettre fin à l'exécution.

Ces agents peuvent dans le cas du checkpointing, réaliser la coordination, interagir avec les MSS pour écrire et lire des informations, surveiller certains événements, … La technologie des agents mobiles trouve ses applications dans les domaines de la recherche d'information, commerce électronique, gestion des workflow, télécommunications, … Par contre, ils n'ont pas, à notre connaissance, été utilisé dans le domaine de la tolérance aux pannes.

BIBLIOGRAPHIE ET REFERENCES

[ABI,1994] A. Acharya, B.R. Badrinath et T. Imielinski, "*Checkpointing Distributed Applications on Mobile Computers*". Proc. Third International Conf. Parallel and Distributed Information Systems, Septembre 1994.

[Bad,1995] N. Badache, "*La mobilité dans les systèmes répartis*". Rapport technique,. Publication interne n°962, IRISA, Rennes, France, Octobre 1995.

[Bad,1997] N. Badache, "*Ordre Causal et Tolérance aux Défaillances en Environnement Mobile*". Thèse de Doctorat d'état, USTHB, Alger, Algérie, Octobre 1998.

[BAI,1993a] B.R. Badrinath, A. Acharya et T. Imielinski, "*Impact of Mobility on Distributed Computations*". ACM Operating system review, vol. 27, n°2, Avril 1993.

[BAI,1993b] B.R. Badrinath, A. Acharya et T. Imielinski, "*Structuring Distributed Algorithms for Mobile Hosts* ". Technical report, Rutgers DCS-TR-298 / WINLAB TR-55, Avril 1993.

[BAI,1994] B.R. Badrinath, A. Acharya et T. Imielinski, "*Designing Distributed Algorithms for Mobile Computing Networks*". Technical report, Dept of Computer Science, University of Rutgers, USA, 1994.

[BHMR,1997] R. Baldoni, J.M. Hélary, A. Mostefaoui et M. Raynal, "*A Communication Induced Checkpointing Protocol that Ensures Rollback-Dependency Trackability*". In 27th IEEE Symp. on Fault Tolerant Comp. (FTCS-27), pp. 68-77, Juin 1997.

[BHR,1997] R. Baldoni, J.M. Hélary et M. Raynal, "*Consistent Records in Asynchronous Computations*". Acta Informatica, 1997.

[BJK,1996] P.W. Baier, P. Jung et A. Klein, "*Taking part of multiple access for third generation cellular phone systems—A European view*". IEEE Comm. Mob. pp. 82-89, Février 1996.

[BM,1993] O. Babaoglu et K. Marzullo, "*Consistent Global States of Distributed Systems : Fundamental Concepts and Mechanisms*". In ACM Press, 28th Distributed Systems (Ed. : S. Mullender), 1993.

[BS,1983] G. Barigazzi et L. Strigini, "*Application-Transparent Setting of Recovery Points*". Digest of papers Fault-Tolerant Computing Systems-13, pp. 48-55, 1983.

[CJ,1991] F. Cristian et F. Jahanian, "*A Timestamp-Based Checkpointing Protocol for Long-Lived Distributed Computations*". Proc. IEEE Symp. Reliable Distributed Systems, pp. 12-20, 1991.

[CL,1985] K.M. Chandy et L. Lamport, "*Distributed Snapshots: Determining Global States of Distributed Systems*". ACM Trans. Computer Systems, Février 1985.

[CS,1998] G. Cao et M. Singhal, "*On the Impossibility of Min-Process Non-Blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems*". Proc. 27th International Conf. On Parallel Processing, pp. 37-44, Août 1998.

[CS,2001] G. Cao et M. Singhal, "*Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems*". IEEE Trans. On Parallel and Distributed Systems, Vol. 12 n° 2, Février 2001.

[DP,1994] Y. Deng et E.K. Park, "*Checkpointing and Rollback-Recovery Algorithms in Distributed Systems*". J. Systems and Software, pp. 59-71, Avril 1994.

[EAWJ,1999] E.N. Elnozahy, L. Alvisi, Y.M. Wang et D.B. Johnson, "*A Survey of Rollback-Recovery Protocols in Message-Passing Systems*". CMU-CS-99-148, School of Computer Science, Carnegie Mellon University, Pittsburgh, Juin 1999.

[EJZ,1992] E.N. Elnozahy, D.B. Johnson et W. Zwaenepoel, "*The Performance of Consistent Checkpointing*". Proc. 11th Symp. Reliable Distributed Systems, pp. 86-95, Octobre 1992.

[FAG,1995] D.D. Falconer, F. Adachi et B. Gudmundson, "*Time division multiple access methods for wireless personal communications*". IEEE Commun. Mag., vol. 33, 1995.

[GJPW,1991] K.S. Gilhousen, I.M. Jacobs, R. Padovani, A. A. Weaver, Jr. et C. E. Wheatley III, "*On the cellular CDMA system*". IEEE Trans. Veh. Tech., 1991.

[God,1997] L.C. Godara, " *Prolog to Applications of Antenna Arrays to Mobile Communications, Part I: Performance Improvement, Feasibility, and System Considerations* ". In Proc. IEEE Vol. 85 N° 7 pp. 1029-1060, Juillet 1997.

[HMR,1997] J.M. Hélary, A. Mostefaoui et M. Raynal, "*Virtual Precedence in Asynchronous Systems: Concept and Applications*". In 11th Workshop on Distr. Algo. (WDAG11), Saarbrucken, Allemagne, Springer-Verlag, LNCS 1320, pp. 170-184, Septembre 1997.

[HMR,1998] J.M. Hélary, A. Mostefaoui et M. Raynal, "*Points de Contrôle Cohérents dans les Systèmes Répartis : Concepts et Protocoles*". In TSI, vol. 17 n° 10, pp. 1223-1245, Mars 1998.

[HNR,1998] J.M. Hélary, R.H.B. Netzer et M. Raynal, "*Consistency Issues in Distributed Checkpoints*". IEEE Trans. On Software Eng., 1998.

[HT,1998] H. Higaki et M. Takizawa, "*Recovery Protocol for Mobile Checkpoints*". Proc. 9th International Workshop and Database and Expert Systems Applications, pp. 520-525, 1998.

[Kar,1992] P. Karn, "*Amateur Packet Radio and TCP/IP*". 1992.

[Ken,1997] L. Kenney, "*Introduction to Packet radio, Part 1 to 20*". Février 1997.

[KMM,1995] R. Kohno, R. Meidan et L.B. Milstein, "*Spread access methods for wireless communications*". IEEE Comm. Mag., Vol. 33, pp. 58-67, Janvier 1995.

[KPD,1985] P. Karn, H. Price et R. Diersing, "*Packet radio in the amateur service*", IEEE Journal on Selected Areas in Communications, Vol. SAC-3, N°3, pp. 431-439, Mai 1985.

[KT,1987] R. Koo et S. Toueg, " *Checkpointing and Rollback Recovery for Distributed Systems* ". IEEE Trans. Software Eng., pp. 23-31, Janvier 1987.

[Lam,1978] L. Lamport, "*Time, Clocks and the ordering of events in a distributed system*". Communication of the ACM, Vol. 21 n°7, pp. 558-565, Juillet 1978.

[Lap,1991] J.C. Laprie, "*Sureté et fonctionnement : concepts de base et terminologie*". In Dependable Computing and Fault-TolerantSystems, pp. 47-95, Springer-Verlag, Juillet 1991.

[LB,1988] P.Y. Leu et B. Bhargava, "*Concurrent Robust Checkpointing and Recovery in Distributed Systems*". Proc. 4th IEEE International Conf. Data Eng., pp. 154-163, Janvier 1988.

[LD,2001] C.M. Lin et C.R. DOW, "*Efficient Checkpointing-based Failure Recovery Techniques in Mobile Computing Systems*". Journal of Information and Science and Engineering vol. 17, pp. 549-573, 2001.

[Lee,1991] W.C.Y. Lee, "*Overview of cellular CDMA*". IEEE Tchnol., vol. 40, 1991.

[Lin,2001] C.M. Lin, "*Checkpointing-based Fault Tolerance Techniques in Distributed and Mobile Computing Systems*". thèse de doctorat, http://www.mclab.iecs.fcu.edu.tw/thesis/æ □æ-£æ• □.pdf, 200

[LNT,1987] B.M. Leiner, D. L. Nielson et F. M. Pease, "*Issues in Packet Radio Network Design*". Proceeding of the IEEE, Vol. 75, N°1, pp. 6-20, Janvier 1987

[LSP,1982] L. Lamport, R. L. Shostak et F.A. Tobagi, ."*The Byzantine generals problem*". ACM Transactions on Programming Languages an Systems, pp. 382-401, Juillet 1982.

[LY,1987] T.H. Lai et T.H. Yang, "*On Distributed Snapshots*". Information Processing Letters, pp. 153-158, Mai 1987.

[Mah,1997] H. Mahmoudi, "*2001 l'odyssée des télécoms*". 01 Réseaux / N° 45, Décembre 1997.

[Man,2001] Y. Manabe, "*A Distributed Consistent Global Checkpoint Algorithm for Distributed Mobile Systems*". 8th International Conf. on Parallel and Distributed Systems (ICPADS 2001), KyongJu, Corée du Sud, Proceedings. IEEE Computer Society, pp. 125-134, , Juin 2001.

[Mat,1989] F. Mattern, "*Virtual Time and Global States in Distributed Systems*". In Parallel and Distributed Algorithms, ed. M. Cosnard & al., pp. 215-226, 1989.

[Mos,1994] A. Mostefaoui, "*Conception et expérimentation d'algorithmes pour la coopération répartie*". Thèse de Doctorat, Université de Rennes 1, France, Juin 1994.

[MS,1996] D. Mannivannan et M. Singhal, "*Failure recovery based on Quasi-Synchronous Checkpointing in Mobile Computing Systems*". Tech. Report n° OSU-CISRC-7/96-TR36, Dept of Comp. And Information Science, Ohio State Univ., USA, 1996.

[MS,1999] D. Mannivannan et M. Singhal, " *Quasi-Synchronous Checkpointing : Models, characterization, and classification*". IEEE Trans. Parallel and Distributed Systems, vol. 10, n° 7, pp. 703-713, Juillet 1999.

[NX,1995] R.H.B. Netzer et J. Xu, "*Necessary and Sufficient Conditions for Consistent Global Snapshot*". IEEE Trans. on Parallel and Distributed Systems, Vol. 6 n° 2, pp. 165-169, 1995.

[Pan,1995] R. Pandya, "*Emerging mobile and personal communication systems*". IEEE Commun. Mag., Vol. 33, pp. 44-52, Juin 1995.

[PGH,1995] J.E. Padgett, C.G. Gunther et T. Hattori, " *Overview of wireless personal communications*". IEEE Commun. Mag., vol. 33, pp. 28-41, Janvier 1995.

[PMS,1991] R.L. Pickholtz, L.W. Milstein et D.L. Schill, "*Spread spectrum for mobile communications*". IEEE Trans. Veh. Technol. Vol. 40, pp. 313-322, 1991.

[PS,1996] R. Prakash et M. Singhal, "*Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems*". IEEE Trans. Parallel and Distributed Systems, pp. 1035-1048, Octobre 1996.

[QBC,1997] F. Quaglia, R. Baldoni et B. Ciciani, "*A Checkpointing-Recovery Scheme For Domino Free Distributed*". Proc. 2nd Workshop on Fault-Tolerant Parallel and Distributed Systems, 1997.

[QCB,1998] F. Quaglia, B. Ciciani et R. Baldoni, "*Checkpointing Protocols in Distributed Systems with Mobile Hosts : a Performance Analysis*". IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, Orlando, pp. 742-755, Avril 1998.

[RS,1993] P. Ramanathan et K.G. Shin, "*Use of Common Time Base for Checkpointing and Rollback Recovery in a Distributed system*". IEEE Trans. Software Eng., pp. 571-583, Juin 1993.

[RU,1991] K. Raith et J. Uddenfeldt, "*Capacity of digital communications systems*". IEEE Trans. Veh. Technol. Vol. 40, 1991.

[SS,1983] R.D. Schlichting et F.B. Schneider, "*Fail-stop processors : an approach to designing fault-tolerant computing systems*". ACM transactions on Computer Systems, Août 1983.

ANNEXE A:

COMMUNICATION SANS FIL

1. INTRODUCTION

Si l'art et la manière de relier les ordinateurs entre eux sont relativement récents, les concepts essentiels mis en œuvre dans les réseaux datent du dix-neuvième siècle. La première idée des réseaux a vu le jour en 1844 par Samuel Morse lorsqu'il a testé avec réussite son télégraphe.

L'ordinateur et le téléphone créés à des époques différentes et à partir de technologies différentes, semblent cependant avoir été conçus l'un pour l'autre et ils ont fini par aboutir aux réseaux informatiques. Actuellement, la plupart des *P.C.* utilisés dans les entreprises ou dans les établissements d'éducation sont connectés à un réseau.

Aujourd'hui, au temps du téléphone mobile et du micro-ordinateur portable, il est tout à fait normal que la terminologie informatique soit enrichie par un nouveau terme qui n'est autre que le *réseau mobile*. Le terme *mobile* dénote un appareil de communication en mouvement incluant: Le portable de main (*hand-held portable*), le mobile véhiculé sur terre, navire ou avion.

Les services de communications mobiles et sans fil se diversifient en utilisant *le mobile terrestre* (*land-mobile*), *la radio interne* (*indoor-radio*), *les systèmes basés satellite* (*satellite-based systems*). Leur objectif est que tout appareil mobile puisse communiquer avec un autre n'importe où dans le monde à n'importe quel moment.

L'utilisation des communications mobiles sans fil est en croissance continuelle, couvrant ainsi différents domaines techniques. Ils offrent une grande flexibilité dans le design du réseau, un déploiement facile et rapide et sont plus convenables aux différents terrains et climats. La mobilité permet le développement de nouvelles classes d'applications: services d'informations avec accès à divers bases de données en tout lieu et en tout temps et des applications dites verticales relevant de domaines spécifiques: location, localisation, … [Bad,1995].

Ce chapitre a pour objectif de présenter les différentes possibilités de communications sans fil existantes dans le monde pour comprendre leur principe de

fonctionnement. Ainsi, nous introduirons la terminologie requise à ce domaine en présentant les différentes techniques de communication.

2. DEFINITIONS ET NOTIONS ELEMENTAIRES

Nous parlerons dans ce paragraphe de quelques détails et problèmes comme les différents *modes d'accès*, le *problème d'affectation d'un canal*, comment gérer un nombre limité de canaux, etc. Quelques notions que nous verrons dans ce paragraphe seront utiles pour la suite de ce chapitre.

2.1 LES ONDES RADIO

La transmission radio est basée sur le principe que l'accélération d'un électron crée un champ électromagnétique qui à son tour accélère d'autres électrons et ainsi de suite. Il est alors possible de provoquer le déplacement d'électrons d'un point *A* à un point *B* éloigné sous l'effet d'un champ électromagnétique. Plus le nombre d'électrons déplacés est important, plus le signal est fort et plus sera grande sa portée, avec une vitesse proche de celle de la lumière. Un déplacement coordonné constitue la base de la transmission d'information dans une communication sans fil. Le spectre des fréquences est divisé en plusieurs parties (bandes de fréquences). [Bad,1995]

Dans la table suivante nous présentons les différents spectres de fréquences qui existent comme définis par le standard IEEE [God,1997]:

La bande	Fréquences
H.F.	3 à 30 *MHz*
VHF	30 à 300 *MHz*
UHF	300 à 1000 *MHz*
L	1 à 2 *GHz*
S	2 à 4 *GHz*
C	4 à 8 *GHz*
X	8 à 12 *GHz*
Ku	12 à 18 *GHz*
K	18 à 27 *GHz*
Ka	27 à 40 *GHz*
V	40 à 75 *GHz*
W	75 à 110 *GHz*
mm	110 à 300 *GHz*

Table A.1: Le spectre des fréquences.

2.2 MODES D'ACCES MULTIPLE

Il existe trois méthodes de base pour les accès multiples qui sont: FDMA (Frequency Division Multiple Access), TDMA (Time Division Multiple Access) et CDMA (Code Division Multiple Access).

La première alloue différentes fréquences porteuses aux différents utilisateurs. La seconde (utile pour ses signaux numériques) alloue différentes tranches de temps aux différents abonnés en utilisant la même fréquence et ainsi elle répartit les signaux d'une façon organisée (plus de détails sur le mode TDMA dans [God,1997], [FAG,1995] et [RU,1991]). La dernière utilise des codes séparés pour chaque utilisateur, elle exige une largeur de bande plus grande et en même temps réduit la densité spectrale du signal. Les signaux CDMA occupent la même largeur de bande (discussion sur les caractéristiques de ce mode dans [Lee,1991], [GJPW,1991], [PMS,1991] et [KMM,1995].)

En théorie, les trois fournissent la même capacité et ne sont pas altérés par la division du spectre en fréquences, quantum de temps ou codes [God,1997] (un exemple est donné dans [Lee,1991]).

En pratique, chaque système a ses avantages et ses inconvénients [RU,1991][GJPW,1991]. CDMA possède des caractéristiques lui permettant de se distinguer des autres [God,1997] [KMM,1995].

Un autre système existe, il s'agit du SDMA (Space Division Multiple Access). Il utilise des antennes (array of antennas) pour contrôler l'espace en fournissant des canaux virtuels dans un domaine d'angle et les appels simultanés peuvent être établis à la même fréquence [God,1997].

2.3 CANAL DE COMMUNICATION

Le terme *canal* dénote *une fréquence* dans le système *FDMA, un quantum de temps* dans le système *TDMA, un code* dans le système *CDMA* ou une combinaison de ces trois paramètres dans le système mixte.

2.4 ALLOCATION D'UN CANAL:

L'*assignation d'un canal* est une opération complexe. Pour affecter un canal à un mobile ou à une station de base il existe deux méthodes extrêmes:

- Le système avec *assignation de canal fixe* où l'affectation des canaux aux différentes cellules se fait durant la planification. La répartition change rarement pour refléter les besoins du trafic. Pour le mobile le canal est affecté lors de l'initialisation de l'appel et le mobile communique avec la base en utilisant ce canal.[God,1997]

- L'autre système, par opposé au premier, est l'*assignation dynamique du canal*. C'est une façon d'utilisation efficace du canal dans l'environnement multi-utilisateurs. Un canal avec le minimum d'interférence est trouvé avant l'affectation. Le niveau d'interférence de tous les canaux utilisés et non utilisés est contrôlé périodiquement, ainsi l'affectation d'un canal durant l'appel peut être changé de: un avec *haute interférence* à un autre avec *basse interférence* (*canal silencieux*). L'environnement d'interférences est constamment en changement et ceci est dû au mouvement des mobiles et tant qu'il y aura de canaux silencieux disponibles, il est assuré que la performance du système ne sera pas affectée défavorablement.[God,1997]

Il existe d'autres façons d'affectation entre l'assignation fixe et dynamique, entre autres: l'*assignation flexible* qui altère l'affectation périodiquement pour refléter les besoins du trafic et l'*assignation empreintée* (*borrowing assignment*) où les canaux non utilisés d'une cellule sont empreinté par une cellule encombrée. Tous les deux sont une variation de l'assignation fixe.

2.5 PROTOCOLE *HANDOFF*:

Il arrive que le mobile en se déplaçant soit loin de sa station de base originale, cette distance est détectée par le centre d'aiguillage qui contrôle la puissance du signal arrivant du téléphone vers la station de base. Une fois le signal devient faible, le centre d'aiguillage affecte un nouveau canal de circulation via la station de base la plus proche du mobile et demande au mobile de se régler sur ce canal. Il s'agit donc de résoudre des contraintes d'ordonnancement de ressources. Ce processus est appelé procédure *Handoff* ou *handover* qui est généralement transparent à l'utilisateur. Dans un appel, il peut y avoir plusieurs *handoff*s. L'exécution du *handover* est basée sur certaine politique établie, qui implique le niveau de puissance mesuré et la qualité de réception. Une politique basée sur la qualité de réception peut impliquer une probabilité de blocage d'appel ainsi qu'une probabilité de bloquer les *handoff*s. Une politique basée sur un nombre minimum de *handoff*s peut conduire à une qualité pauvre de communication [God,1997].

3. LES SYSTEMES DE COMMUNICATION SANS FILS

3.1 SYSTEME RADIO AMATEUR (*PACTOR* ET *PACKET RADIO*)

Pactor et *Packet radio* sont les mode numériques des communications amateurs. Ils envoient des paquets de données via des fréquences radio vers une autre station de même type. Dans ce système le *modem téléphonique* est remplacé par un *TNC* (*Terminal Node Controller*), le *téléphone* est remplacé par un *transmetteur-récepteur* (*transceiver*) *radio* et le *système du téléphone* est remplacé par les *ondes radio*.

Fonctionnement

Lors de la transmission: Le *TNC* assemble un paquet de données reçues de l'ordinateur, calcule les erreurs, module le paquet dans des fréquences audio et génère les signaux appropriés pour transmettre le paquet sur la radio connectée.

Lors de la réception: C'est le processus inverse, il transforme les signaux audio que la radio reçoit en un groupe d'octets qui seront envoyés au processeur.

Le système *Packet radio* utilise les fréquences *VHF* (*Very high frequencies*) avec un rayon de communication de 15 *à* 50 *Km* et un taux de transmission de données de 1.2, 2.4 ou 9.6 *Kbps*. Pour étendre son rayon, des *stations Packet* peuvent jouer le rôle de répéteurs.

Le mode *Pactor* est le plus populaire utilisant les ondes *H.F.* (*High frequencies*) avec un rayon de communication de plus de 15 000 *Km* et un débit de transmission allant de 250 *b/s* à un maximum de 1.2 *Kbps* seulement. Pour plus de détails, le lecteur peut consulter [Ken,97], [LNT,1987], [KPD,1985] et [Kar,1992].

3.2 Station de Base (*BASE STATION*)

La zone servie par le système mobile est divisée en zones plus petites connues comme *cellules* (La configuration standard d'un système de communication cellulaire est un maillage de cellules hexagonales [Bad,1995]). Chaque cellule contient une station de base (B.S.), qui est connectée à un *centre d'aiguillage* (*switching center*), communique avec les téléphones mobiles sur le site par des *liens radio* et connecte ces mobiles au réseau de téléphone d'aiguillage publique. Une unité mobile ne peut être, à un instant donné, directement connectée qu'à une seule station de base. Il faut noter dans ce type de communication, la différence entre le *système mobile terrestre* (*land-mobile*) et le *système mobile en bâtiment* (*in-building mobile*) en raison du type des mobile qu'ils servent et l'environnement dans lequel ils opèrent. Pour le premier, il est généralement véhiculé alors que pour le second, le mobile est un portable de main se déplaçant à pieds.

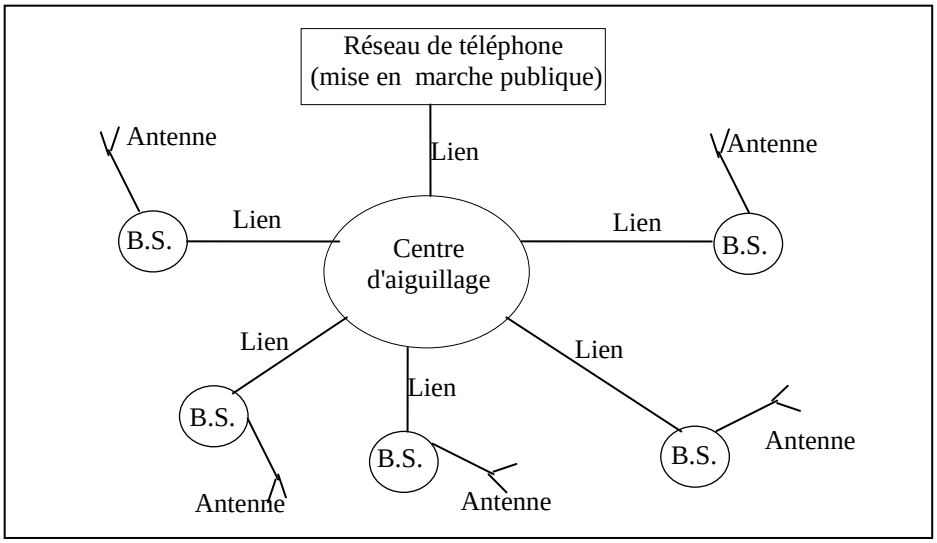


Figure A.1: Installation typique d'un système mobile de base.

Deux types de canaux radio sont utilisés:

- *Canal de contrôle* pour transporter les signaux de contrôle.
- *Canal de circulation* (*traffic channel*) pour transporter les messages.

Dans la littérature:

- La transmission de la station de base au mobile a plusieurs appellations: *downstream*, *forward link* ou *down link.*
- La transmission du mobile à la base: *upstream*, *reverse link* ou *up link.*

Fonctionnement

Dès que le mobile est mis en marche, il teste *le canal de contrôle* et se règle sur le canal au signal le plus fort (habituellement, arrivant de la station de base la plus proche). L'utilisateur s'identifie et demande l'utilisation du réseau, la *station de base* envoie cette demande au *centre d'aiguillage* connecté au réseau téléphonique, qui contrôle plusieurs stations de base. Il assigne *un canal de circulation radio* au téléphone sachant que les canaux de contrôle sont utilisés par tous les téléphones dans cette zone et ne doivent pas servir pour le transport des données.

Lorsque le mobile est appelé, *le centre d'aiguillage* envoie un message de localisation (*paging message*) à travers plusieurs stations de base. Le téléphone, réglé sur le canal de contrôle, détecte son numéro et répond en envoyant un signal de réponse à la base la plus proche, qui informe alors le centre d'aiguillage de l'emplacement du mobile. Le centre d'aiguillage affecte un canal et l'appel est complété.

Le mobile est normalement localisé par la transmission de message paging à partir de divers stations de base. Ceci devient cher et inefficace, si plusieurs stations de base sont impliquées dans le processus de paging. Cet inconvénient est évité par une *procédure* dite *d'enregistrement*. Cette procédure consiste à ce que le mobile fasse un enregistrement avec la base la plus proche de lui. Cette information est mémorisée avec le centre d'aiguillage de la zone, de préférence le centre d'aiguillage local du téléphone. La base locale du mobile est celle où il est enregistré en permanence. Une fois un appel est reçu pour ce mobile, son centre d'aiguillage local contacte le centre d'aiguillage où le téléphone circule couramment. Rechercher dans le voisinage du lieu connu précédemment aide à localiser le mobile. Dés qu'il répond l'appel peut être connecté comme décrit ci-dessus.

Channel reuse distance

Le nombre de canaux dans un système est limité, ce qui peut limiter la capacité du système à assurer les appels simultanés. La solution est d'utiliser un même canal pour plusieurs appels, pour cela il faut respecter une distance minimale pour éviter que l'un trouble ou dérange l'autre. De cette distance (dite *Channel reuse distance*) dépend la capacité du système. Les cellules utilisant le même ensemble de canaux sont connues comme *cellules cocanaux*. Généralement deux cellules peuvent utiliser le même canal si la distance qui les sépare est au moins égale à trois fois le rayon de la cellule. Les interférences causées par les radiations des cellules doivent être minimisées en limitant la puissance de transmission des mobiles et des stations de base qui partagent un même canal (Plus de détails seront trouvés dans [RU,1991][Lee,1991][GJPW,1991][PMS,1991]).

Fragmentation de cellules

Chaque cellule a un nombre limité de canaux et ainsi, à un moment donné, il est possible qu'elle ne puisse plus supporter d'autres appels. Une solution existe c'est de subdiviser la cellule en d'autres plus petites. La puissance de transmission des nouvelles stations de base est inférieure à l'ancienne. Cette procédure est dite *fragmentation des cellules* (*Cell Splitting*).

On appelle *système microcell* si le rayon de la cellule est inférieur à 1 *Km* et la puissance de transmission est réduite, sinon on dit que le système est *Macrocell*. Si la taille des cellules est réduite à moins de 100 *m* on appellera le système *Picocell* [God,1997].

3.3 COMMUNICATIONS MOBILES VIA SATELLITE

Utilisés au début dans les communications intercontinentales puis internationales, les systèmes satellites entrent en force dans le domaine des télécommunications grâce particulièrement à la maîtrise d'énergie, l'intégration électronique et la révolution numérique.

Plusieurs projets de constellations de satellites existent parmi lesquels on cite: *Iridium*, *Globalstar*, *Teledesic*, *Skybridge*, *Celestri* et d'autres. Ces constellations vont permettre à toute personne de communiquer à l'aide d'un mobile via satellite avec une autre personne à n'importe quel endroit[Mah,1997].

De ce fait la fabrication de satellites est passée d'un artisanat de luxe à une production en série à l'échelle industrielle. Les coûts de fabrication des satellites sont en baisse, leur durée de vie est de plus en plus grande et leur émission augmente en puissance et par conséquent les antennes terrestres diminuent en taille.

Le service de communication via satellite assure: *les services de messagerie professionnelle* (entre flottes de transporteurs routiers ou les taxis et leur central d'exploitation) ou de *GPS* (*Global positioning system*) ou bien la météo, l'observation et la protection de l'environnement et aussi la *localisation des véhicules*.

Les satellites

Les premiers satellites étaient cylindriques, épousant la forme des lanceurs pour être petits et faciliter leurs mise en orbite. L'inconvénient qui se posait était la taille des flancs ce qui ne permettait pas de soutenir les panneaux solaires.

La génération suivante avait la *forme cubique* en plus des bras qui portent les cellules photoélectriques (La surface totale peut atteindre 40 m^2). Pour optimiser le rendement, les cellules solaires en *silicium* sont transformées en *arséniure de gallium*. En altitude les éclipse solaires sont fréquentes ce qui nécessite l'utilisation d'une *batterie*. La batterie est lourde et pour diminuer de son poids, il fallait en premier lieu diminuer le temps de son utilisation et ainsi les satellites sont placés à l'ouest de leurs zones de diffusion. En deuxième phase, les batteries d'aujourd'hui sont au *lithium*, ce qui améliore leur acquisition d'énergie solaire et son stockage[Mah,1997].

Il existe trois types de satellites:

- ***LEO*(*Low Earth Orbit***): Circulent d'une façon constante autour de la terre à une très *basse altitude* (de 700 *à* 1 500 *Km*) avec un *délai de propagation* allant de 5 *à* 35 *ms* (à titre de comparaison le délai de propagation sur le réseau filaire est de 0.15 *s*). Les satellites *LEO* projettent sur leur trajectoire des cellules vers la surface de la terre qui sont similaires à l'architecture cellulaire. Le nombre d'orbites autour de la terre est très

grand, de ce fait plusieurs satellites sont nécessaires pour avoir une couverture globale et tout ça augmente le nombre de handoffs. On distingue trois types de satellites *LEO* par leur *bande de fréquences*: *Little LEO* qui utilise des fréquences inférieures à 1 *GHz*, le *Big LEO* avec des fréquences de 2 *GHz* et plus et enfin le *mega LEO* qui fonctionne entre 20 *et* 30 *GHz* (la bande *Ka*).

- ***GEO*** (***Geostationary Earth Orbit***): *Satellites géostationnaires* à environ 36 000 *Km* de la terre en *orbite équatoriale* apparaissent *fixes* aux stations terrestres. Il peuvent couvrir jusqu'au tiers de la planète. Ce qui fait que *trois à quatre satellites GEO*. suffisent pour couvrir toute la terre. Le *délai de propagation* est *fixe* (250 *ms*).
- ***MEO*** (***Medium Earth Orbit***): Ce système présente une orbite, délai de propagation et une architecture compris entre les systèmes *LEO* et *GEO*.

Les satellites à orbites *moyennes* et *basses* sont utilisés pour répondre à des *services de communication mobiles*.

Le tableau suivant compare les trois types de satellites :

	LEO	***MEO***	***GEO***
Altitude (en *Km*).	700 *à* 3 000	5 000 *à* 20 000	35 786
Délai de propagation.	5 *à* 35 ms	50 ms *à* 0.125 s	0.25 s
Nombre se satellites nécessaires pour la couverture globale	40 *à* 70	10 *à* 15	3 *ou* 4
Période orbitale (temps d'une révolution).	90 *à* 120 *minutes*	6 *à* 8 *heures*	24 *heures*
Visibilité du satellite (A partir d'un point fixe sur terre)	10 *à* 20 *minutes*	*Plusieurs heures*	*Toujours*
Complexité du système	Haute	Moyenne	Basse
Taille et poids	Bas	Moyens	Elevés
Durée de vie	Courte (4 *à* 7 *ans*)	Moyenne	Longue (10 *à* 20 *ans*)
Handoffs	Très fréquents	Occasionnellement	Presque inexistants.

Table A.2: Les types de systèmes satellites.

Il existe un *quatrième type de satellites* c'est les *HEO* (*Highly Elliptical Orbit*). Ce système peut être vu comme un système *MEO*, où les satellites tournent autour de la terre en *orbite elliptique* et non circulaire. Ceci étend la durée de couverture de certaines régions c'est à dire les régions sous l'apogée de l'orbite où le satellite apparaît presque comme géostationnaire durant les 2/3 de la période orbitale.

Les satellites géostationnaires sont les plus lents dans la transmission vu la grande distance qui les sépare de la terre alors que le délai de propagation des satellites *LEO* est inférieur même à celui du réseau téléphonique à fil. Par contre et concernant la durée de

vie et la complexité du système et le temps de visibilité le système *GEO* est le plus performant.

Le mobile via satellite

Une variété de configurations pour les communications sans fil via satellite existent: Les *satellites géostationnaires* jouent le rôle d'une *station de relais* entre un mobile et la station de base, ils offrent une *couverture globale et fixe*. Ils veillent à compléter le travail du réseau terrestre et ont un rôle majeur dans les zones où ce dernier n'est pas développé. L'inconvénient des systèmes *GEO* est le délai de propagation en plus de la difficulté de couvrir les zones polaires où le satellite apparaît à l'horizon. Par contre, les satellites *MEO* et *LEO* exigent moins de puissance et causent un délai moindre en parallèle il y a la gestion des *handoffs* qui est nécessaire. Dans ce cas, les mobiles communiquent directement avec le satellite, par opposition au système où la station de base joue le rôle de répéteur en communiquant avec le satellite d'un coté et les mobiles de l'autre. Les communications inter satellites jouent un rôle important dans les communications mobiles, particulièrement avec l'utilisation des satellites à basse altitude.

4. Les différentes générations du Mobile

La technologie des communications mobiles a parcouru un long chemin depuis le travail des pionniers du domaine les laboratoires *AT&T Bell* durant les années 1960 et 1970. En 1983, il y a eu l'ouverture du premier système cellulaire opérationnel à Chicago (USA).

La première génération de téléphone sans fil analogique et des systèmes cellulaires devenait populaire. La conception est basée sur un standard connu sous le nom de *AMPS* (*Advanced mobile phone services*). Ce système utilisé en Australie, Canada, les USA et en Amérique centrale et du sud. Il utilise la bande 824-849 *MHz* pour transmettre du mobile à la base et la bande 869-894 *MHz* pour le sens opposé avec 832 canaux d'une largeur de 30 *KHz*. Des standards similaires basés sur le mode d'accès multiple *FDMA* ont été développés dans le monde comme *TACS* (*Total access communication system*), *NMT* 450 (*Nordic mobile telephone*) et *NMT* 900 en Europe, en Grande Bretagne c'est le standard *ETACS* (*European Total access communication system*), *C*450 en Allemagne et *NTT, JTACS* et *NTACS* au Japon. La décision des *handoffs* est basée sur l'énergie reçue à la base et dans le cas du *C*450 c'est le délai du round-trip. La première génération du téléphone sans fil analogique était conçue pour communiquer avec une seule base pour remplacer le téléphone à fil par un terminal qui fournit une mobilité dans une petite zone.

Les systèmes de la deuxième génération sont conçus pour utiliser la transmission numérique, d'avoir des canaux séparés dédiés pour l'échange d'informations de contrôle entre base et mobiles et d'employer *TDMA* ou *CDMA* comme mode d'accès multiple. Ces systèmes incluent les paneuropéens *GSM* (*Global system for mobile communications*) et *DCS* 1800 (*Digital Cellular System*) utilisant le mode *TDMA*, les nord-américains *IS*-54 (qui utilise le mode *TDMA*) et *IS*-95 (mode *CDMA*) (les deux systèmes *IS* sont conçus pour fonctionner dans la bande de fréquences utilisée par le standard *AMPS*) et enfin le

système japonais *PDC* (*Personal Digital Cellular*) dont le mode est *TDMA*. La seconde génération numérique des systèmes sans fil a été développée de telle façon d'utiliser le même terminal où et d'être apte à recevoir et situer les appels. Le standard britannique *CT*2 (*Second generation cordless telephone*) et les standards japonais *DECT* (*Digital European cordless telecommunication*) et *PHS* (*Pocket handphone service*) font partie de cette génération.

La troisième génération porte le nom de *UMTS* (*Universal mobile telecommunications systems*) et *FPLMTS* (*Future public land-mobile telecommunication systems*). Son objectif est de fournir des services de communication avancés ayant des capacités large bande et utilisant un seul standard. Une étude avec plus de détail sera trouvée dans [PGH,1995], [Pan,1995] et [BJK,1996].

5. Conclusion

En général, les sites fixes sont interconnectés entre eux à travers un réseau de communication filaire fiable et d'un débit élevé, alors que les liaisons sans fil ont une bande passante limitée qui réduit sévèrement le volume des informations échangées [Bad,1995]. Ceci est le plus grand inconvénient des systèmes de communication mobiles, mais des améliorations dans ce domaine ont été réalisées.

Le système de radio amateur parait très efficace du point de vue coûts d'achat, d'installation et d'utilisation qui sont bas comparativement aux autres techniques. Ce système est disponible actuellement. Par ailleurs, il ne manque pas d'inconvénients dont le principal est la large variation imprévue de la qualité de ses performances en terme d'accessibilité, fiabilité (erreurs de transmissions) et disponibilité (coupures des appels). Le système *PACTOR* ou *Packet radio* sont et resterons toujours des techniques amateurs dont le développement reste incertain avec un très faible débit en plus c'est les utilisateurs eux même qui doivent entretenir et maintenir leurs matériels et de ce fait le réseau.

D'autre part, le coût d'installation et d'opération du système de station de base peut être très haut car à chaque cellule il faut installer une station de base même aux endroits où il y a peu d'utilisateurs. Tout de même, le mobile cellulaire est le plus utilisé dans le monde actuellement et ses coûts sont les plus bas.

Les prix d'achat et d'utilisation des systèmes satellites sont très chers. Mais à long terme, les prix de communication via satellite vont diminuer. Ainsi le système de station de base ou le système filaire ne pourront plus faire face aux satellites. Ces derniers donnent une couverture globale de la terre et un temps de propagation négligeable (pour les satellites *LEO*) en plus de la possibilité d'intégration avec les réseaux existants.

L'utilisation des antennes dans les systèmes de communication sans fil est très importante pour améliorer la qualité du signal et augmenter l'efficacité du système. Dans [God,1997], une étude sur l'utilisation des *array antennas* est présentée, il discute leurs avantages dans un système de télécommunication mobile et les améliorations qu'ils peuvent donner.

ANNEXE B:

ARCHITECTURE DES RESEAUX SANS FIL

1. INTRODUCTION

L'intérêt pour les réseaux est en croissance continu à tous les niveaux. Cet intérêt est motivé par l'utilisation croissante des terminaux portables en milieu industriel et logistique, le besoin d'un accès permanent des populations nomades au système d'information de l'entreprise et pour réaliser des transmissions de messages courts (bips, numériques, alphanumériques.), voix, données informatiques, fax, fichiers, textes, images...

Les réseaux mobiles permettent de réaliser des installations temporaires, de mettre en place des réseaux en un temps très court, éviter le câblage de locaux, de liaisons inter-bâtiments et de créer une infrastructure dans des bâtiments classés. Les réseaux sans fils sont rendus possibles grâce à la maîtrise de la téléphonie cellulaire sur une large échelle, numérisation des communications, miniaturisation des interfaces et la disponibilité de nouvelles fréquences.

Nous pouvons observé la mobilité dans un réseau (wired) comme étant un déplacement de son PC portable d'un LAN à un autre. Dans un réseau sans fil (wireless), la mobilité est vue comme le Changement de support de communication physique (cellule) lors du déplacement. Nous pouvons schématisé les différentes configurations de réseaux sans fils selon la zone de couverture par le schéma ci-dessous.

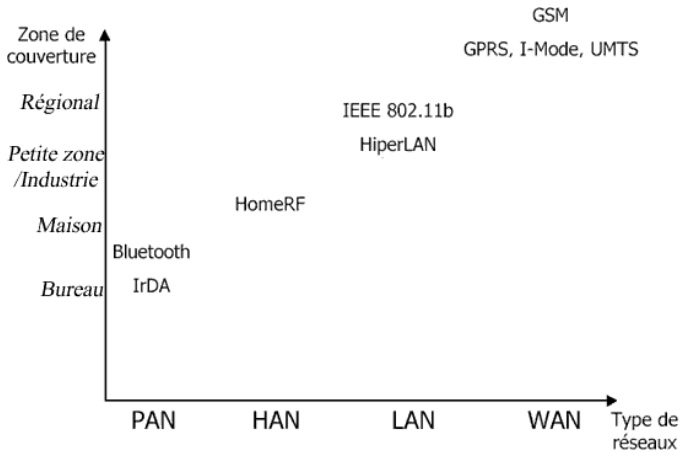


Figure B.1: Classification des réseaux mobiles et sans fil selon couverture.

1.1 RESEAUX PAN/HAN

Il permettent à l'usager une liberté dans les déplacements de quelques mètres autour de lui. Ce sont des réseaux déplacent avec l'usager sans stations relais. Parmi les architectures de ce type de réseaux nous citerons Bluetooth qui a une bande passante qui peut atteindre 1Mbps ou IrDA avec une bande passante arrivant jusqu'à 4 Mbps.

1.2 HAN/LAN

Ce sont des réseaux qui permettent de couvrir une localisation fixe. Ils Requièrent une station de relais pour faciliter les déplacement et les localisations. Ce genre de réseaux permettent un déplacement allant jusqu'à quelques centaines de mètres. Le débit de ses réseaux est lié à la distance. Ainsi, à 50 m on peut avoir un débit maximum et à 550 m le débit sera réduit. Parmi les architectures de ce type de réseaux nous citerons : IEEE 802.11, HyperLan, HomeRF, AirPort, DECT, …

Nous trouvons ce type de réseau dans les bureaux, salles de conférences, centres de formation, bureaux des filiales du groupe, aéroports, hôtels, chez soi …etc.

1.3 WAN

Ce genre de réseaux a évolué depuis le début de ce siècle en permettant au début juste le service de téléphonie et l'échange de voix jusqu'au transfert de données et multimédia actuellement.

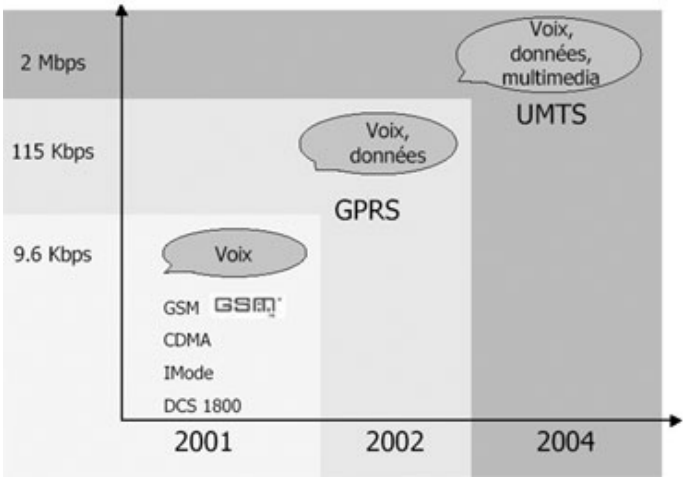


Figure B.2: évolution des réseaux WAN.

2. PERSONAL AREA NETWORK (PAN)

Les réseaux personnels permettent une communication intra-personnelle comme : PDA, GSM, Montre, Bague (*JavaRing*) ou des étiquettes électroniques RFID (*Tags*). Ils permettent aussi une communication entre deux personnes comme l'échange des cartes de visite. Dans la maison iuls permettent un échange entre plusieurs appareils.

Ces réseaux on été normalisé par IEEE à travers IEEE P802.15 Working Group for Wireless Personal Area Networks. Ils ont abouti aux réseaux BlueTooth.

3. HAN/LAN

Parmi les réseaux LAN nous citerons les réseaux de la norme IEEE 802.11 avec les trois versions a, b et g. Ils permettent un débit maximal de 54 Mbps pour *IEEE 802.11a*, 11 Mbps pour *IEEE 802.11b* et,20 Mbps pour *IEEE 802.11g*. Nous pouvons aussi citer pour cette norme, les réseaux optant pour la technologie radio comme le *DSSS (IEEE 802.11b)* et *OFDM (IEEE 802.11a,g).* Il y a aussi les réseaux LAN américain appelé aussi le *IEEE des USA* ou encore le *Pas de QoS*.

Un autre type de réseaux LAN est le HiperLan 1 et HiperLan 2 qui permettent un débit maximal de 23.5 Mbps pour HiperLan1 et de 35 Mbps pour HiperLan 2. Pour cette architecture nous pouvons citer les réseaux : ETSI (Europe), Déterministe, QoS, Tunneling IEEE 1394 (FireWire) pour réseaux multimédia domestiques, Cellules Hyperlan qui sont un complément de UMTS semi-publics (gare, aéroport, …).

Les autres types de réseaux LAN/HAN sont : HomeRF (1.6 Mbps), Apple Airport (11 Mbps), DECT, WLL et 5Wing qui est une convergence entre 802.11 et Hyperlan (IEEE et ETSI).

4. IP ET MOBILITE

Dans un but de permette d'avoir une adresse IP permanente universelle tout en maintenant des connexions actives en mouvement, plusieurs propositions comme le Mobile IPv6 (IETF), Cellular IP (Ericsson), ...

Ils permettent de supporter un très grand nombre d'utilisateurs et assurer la transparence pour les applications et l'adressage doit aussi assurer la sécurité des réseaux et des données.

Ils ont pour objectifs de traiter :

- Faire du routage mobile : Mobile Router.
- Roaming et HandOver.
- QoS et Temps Réel.
- Sécurité et Facturation.
- AAA : Authentication, Authorization, Accounting.
- Réaliser : l'Adressage et le Nommage.
- IPv6, DHCP et Dynamic DNS.
- Mobile ISP.