MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS-SETIF

UFAS (ALGERIE)

MEMOIRE

Présenté à la faculté des sciences de l'ingénieur

Département d'informatique

Pour l'obtention du diplôme de

MAGISTER

Par

M^{elle} : Benabid Sonia

Thème

Optimisation de la gestion des points de reprise dans les réseaux mobiles

Soutenu le: **devant la commission d'examen :**

Mr. Salem Yacine	M.A Chargé des cours l'université de Sétif	Président
Mr. Aliouat Makhlouf	M.A Chargé des cours l'université de Sétif	Rapporteur
Mr. Nekkache Mabrouk	M.A Chargé des cours l'université de Sétif	Examinateur
Mr. Benaouda Abdelhafid	M.A Chargé des cours l'université de Sétif	Examinateur

Année 2006 / 2007

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS-SETIF

UFAS (ALGERIE)

MEMOIRE

Présenté à la faculté des sciences de l'ingénieur

Département d'informatique

Pour l'obtention du diplôme de

MAGISTER

Par

M^{elle} : Benabid Sonia

Thème

Optimisation de la gestion des points de reprise dans les réseaux mobiles

Soutenu le: **devant la commission d'examen :**

Mr. Salem Yacine	M.A Chargé des cours l'université de Sétif	Président
Mr. Aliouat Makhlouf	M.A Chargé des cours l'université de Sétif	Rapporteur
Mr. Nekkache Mabrouk	M.A Chargé des cours l'université de Sétif	Examinateur
Mr. Benaouda Abdelhafid	M.A Chargé des cours l'université de Sétif	Examinateur

Année 2006 / 2007

Remerciements

Je tiens à remercier le bon Dieu et tous ceux qui ont été là de prêt ou de loin, ma famille, mes amis, mon encadreur, mes enseignants et mes collègues …..Tous ceux qui m'ont soutenu, épaulé et cru en moi.

Et particulièrement ceux qui m'ont recueilli, et je n'avais pas à réaliser ce travail que par leur aide et leur amour qui m'est précieux, même si on ne portait pas le même nom de famille.

A tous ceux qui me sont chers ……..

Résumé

Le travail présenté dans ce mémoire consiste à étudier la gestion des points de reprise (checkpoints) dans les applications réparties s'exécutants sur des réseaux fixes ou des réseaux mobiles.

L'étude des protocoles d'établissement de points de reprise dédiés à l'environnement réparti, nous a permis de classer ces protocoles selon des critères qui peuvent jouer sur la performance du système, voire la quantité du travail à refaire, le nombre de points de reprise à stocker, le nombre de site impliqués dans cette application et invoqués dans la procédure de reprise.

Tandis que l'étude faite sur l'environnement mobile dévoile des points cruciaux à prendre en considération lors de l'établissement des points de reprise. En effet, les caractéristiques du support mobile (largeur de la bande passante limitée, capacité da stockage restreinte, faible source d'énergie…) remettent en cause l'étude précédente et conclue que les protocoles dédiés aux réseaux fixes deviennent inadaptés une fois portés sur l'environnement mobile.

Mots clés : Applications distribuées, environnement mobile, tolérance aux fautes, Protocoles de checkpointing.

Liste des figures

Liste des tableaux

Table des matières

Introduction

Au cours de sa vie opérationnelle, un système contient inéluctablement des fautes de conception, quel que soit l'effort de spécification et de validation fourni. Le zéro faute n'est pas un objectif réaliste et un défit à relever étant donné les coûts de développement que cela induirait. Il est donc important, pour les systèmes dits critiques, d'évaluer le risque auquel sont soumis les utilisateurs de ce système.

Si on sous-entend par risque la défaillance du système, donc la non défaillance est l'objectif à atteindre et les moyens mis en oeuvre traitent l'objet qui en est l'origine : la faute. Qu'elle soit issue d'une mauvaise conception, de perturbations de l'environnement, etc..., elle provoque un dysfonctionnement interne du système caractérisé par un état indésirable appelé erreur. La défaillance constitue alors l'effet de cette erreur sur les interactions du système avec son environnement.

L'étude de ces moyens appelée sûreté de fonctionnement du système, est définie comme la propriété qui permet de placer une confiance justifiée dans le service qu'il délivre. Elle aborde le problème à travers quatre points de vue complémentaires : l'évitement, l'élimination, la prévention et enfin la tolérance aux fautes.
Puisque un comportement sûr d'une application repose entièrement sur la continuité du service délivré par le système, nous nous trouvons en face d'une incitation impérieuse à l'adoption de la tolérance aux fautes qui permet d'atteindre le but sollicité par le système (service) en dépit des problèmes. Cela signifie que le système doit survivre à des pannes éventuelles en les tolérant.

Le domaine des applications réparties, ne cesse de croître, et la maîtrise de ces dernières et des outils qui permettent de les construire passe en effet par la connaissance des éléments fondamentaux de ce qui est convenu d'appeler un système réparti par rapport à un système centralisé. Un système réparti présente une différence essentielle par l'échange des messages et l'absence de mémoire et d'horloge communes.
D'autre part, les récentes avancées dans le domaine des communications sans fil et les progrès technologiques dans les terminaux portables ont rendu possibles de nouvelles applications dans lesquelles l'usager peut avoir accès à l'information à n'importe quel moment et depuis n'importe emplacement. La société de l'information de demain bénéficiera de la mobilité qui deviendra la règle et non plus l'exception dans des

environnements saturés de moyens de calculs et de communication au service des usagers [1].

Nous sommes successivement passés des réseaux locaux fixes à des réseaux à grande échelle (Internet) filaires, puis à des réseaux sans fil interconnectant des machines mobiles comme des téléphones portables ou des ordinateurs portables. Cette évolution a abouti à la définition d'une nouvelle technologie, qui se base sur l'infrastructure des réseaux mobiles. Cette technologie est l'informatique mobile (nomade ou ubiquitaire) dans laquelle l'utilisateur peut continuer d'accéder à l'information fournie par une infrastructure distribuée, sans tenir compte de l'emplacement ou il se trouve.

L'informatique mobile souffre de plusieurs lacunes. La mobilité elle-même complique la procédure de localisation des noeuds au sein du réseau. Les terminaux mobiles sont limités en terme de capacité de stockage, d'énergie de la batterie qui a une durée de vie limitée et de la largeur de la bande passante. De plus, le terminal mobile est sujet à des déconnexions fréquentes voltaires ou non.
Parallèlement, les réseaux mobiles prennent une place plus importante chaque jour dans tous les aspects des activités humaines, mais l'actualité nous montre bien leur vulnérabilité aux pannes, perte et aux dommages physiques. Il devient donc indispensable de savoir et définir leur sûreté.

Si un processus s'exécute depuis un certain temps avant qu'une panne ne survienne, il est préférable de compléter son exécution (présumée correcte) plutôt que de recommencer le travail accompli, surtout s'il s'agit d'un processus long (applications exécutées sur des grappes de calculateurs), qui a déjà entamé des heures et des heurs d'exécution, et dont le retard pourrait générer un coût intolérable. Ou encore, si ce même processus modifie un enregistrement dans une base de données partagée, et tombe en panne lors de ces modifications, il devient nécessaire de défaire ces modifications partielles apportées par ce dernier sinon la base de données se retrouvera dans un état incohérent. Pour maintenir une continuité de service en dépit des occurrences des pannes on a eu recours aux techniques de tolérance aux fautes. Plus précisément : la reprise.

La reprise consiste en la sauvegarde périodique ou à des intervalles différents, un état du système, présumé correct, appelé point de reprise (Checkpoint) dans une mémoire

stable. A l'occurrence d'une panne, le système redémarrera à partir de ces points. Cette procédure paraît très simple, si le processus est le seul concerné par la reprise.
Dans les systèmes répartis, la reprise peut impliquer plusieurs processus avec lesquels le processus défaillant a communiqué de façon directe ou indirecte. Eux aussi doivent 'reculer' vers leur dernier point de reprise.
Beaucoup de problèmes apparaissent lors de la conception des algorithmes de reprise pour systèmes répartis fixes, comme l'effet Domino [29], la perte de messages, la perte de vivacité, etc.

La première partie de ce mémoire, est consacrée à l'étude de la sûreté de fonctionnement, ses attributs, ses entraves, ses moyens. La deuxième partie comportera l'étude de la tolérance aux fautes et des algorithmes de reprise dans les réseaux fixes, en concluant par une étude comparative comportant les avantages et les inconvénients des différentes stratégies de reprise. La troisième partie, reprend l'étude précédente tout en cernant les caractéristiques du mobile. En fait, on ne peut pas dire qu' il y'ait eu réellement de nouvelles propositions dans le cadre des réseaux mobiles, mais seulement des améliorations du réparti, auquel on a apporté de nouvelles spécifications et solutions adaptées à la mobilité du système. Nous terminons enfin par une conclusion appropriée.

Première Partie

L'étude de la sûreté de fonctionnement

Chapitre1

La sûreté de fonctionnement.

1.1 Les enjeux de la sûreté de fonctionnement

Les activités industrielles et humaines font presque quotidiennement les grands titres de l'actualité avec leurs cortèges d'incidents, d'accidents ou d'événements catastrophiques. En effet, le zéro défaut ou le risque zéro n'existe malheureusement pas pour les activités industrielles à cause de l'occurrence de défaillances humaines ou matérielles. Toutefois, pour tenter de réduire les risques à un niveau le plus faible possible et acceptable par l'opinion publique, des méthodes, des techniques et des outils scientifiques ont été développés dès le début du 20me siècle pour évaluer les risques potentiels, prévoir l'occurrence des défaillances et tenter de minimiser les conséquences des situations catastrophiques lorsqu'elles se produisent. L'ensemble de ces développements méthodologiques à caractère scientifique représente, à l'aube du troisième millénaire, la discipline de la sûreté de fonctionnement. La sûreté de fonctionnement consiste à connaître, évaluer, prévoir, mesurer et maîtriser les défaillances des systèmes technologiques et les défaillances humaines.

Elle pénètre progressivement dans les secteurs d'activités où les contraintes relatives à la compétitivité des produits et des services s'évaluent en termes d'économie, de qualité et de fiabilité.

1.2 Historique

L'histoire de la sûreté des systèmes fabriqués par l'homme reste à faire, mais elle n'est bien évidemment pas née avec l'informatique. Les premières études firent leur apparition essentiellement dans les transports ferroviaires pour le développement des recueils statistiques des pièces mécaniques. Les études quantitatives de l'époque reposaient sur l'identification d'éléments supposés critiques pour lesquels des améliorations de la conception technique s'imposaient. L'analyse des grandes catastrophes a démontré les limites du principe consistant à ne renforcer que les points critiques. Le naufrage du Titanic qui a coûté la vie à 1 500 personnes pendant la nuit du 14 au 15 avril 1912 en a été une preuve indiscutable. Considéré comme un modèle de sécurité, grâce à ses compartiments séparés par des cloisons étanches, le Titanic a néanmoins sombré après avoir heurté un iceberg, car on avait conçu des cloisons dont les hauteurs étaient insuffisantes. En basculant vers l'avant du navire tous les différents compartiments se remplirent, entraînant la perte du navire et d'une partie de ses passagers et de ses membres d'équipage.

Dans les années 60 et dans le cadre de leurs programmes de missiles intercontinentaux et de la conquête spatiale (programmes Mercury et Gemini) les Etats-Unis ont formalisé l'essentiel des méthodes d'analyse de la sûreté de fonctionnement utilisées encore aujourd'hui : analyse des modes de défaillance et de leurs effets (aéronautique et LEM), arbres des causes (aéronautique, missile Minuteman), méthode des combinaisons de pannes (SNIÀS : Concorde, puis Airbus). Dans l'industrie nucléaire, l'accident de Mile Island le 28 mars 1979, qui ne fît aucune victime mais qui eut un impact considérable sur l'opinion publique, conduisit à des développements comme ceux entrepris par Norman Rasmussen dans le cadre du rapport WASH-1400. En dépit de ses faiblesses, ce rapport a constitué l'ébauche des premières études structurées en matière d'analyses de risques. La normalisation des termes relatifs à la sûreté de fonctionnement commença à s'établir sous l'égide notamment de la CEI (Commission Electrotechnique Internationale).

Les dernières décennies, ont été marquées par la prise en compte énorme de la Sûreté de fonctionnement dans les études de cas critiques et la naissance des notions de maintenance, de disponibilité, de maintenabilité et les concepts associés: testabilité, survivabilité, diagnostic, soutien logistique intégré...).

1.3 Evolution de la discipline

La Sûreté de fonctionnement est appelée la science des défaillances [2]. D'autres désignations existent suivant les domaines d'applications : analyse de risque, science du danger, FMDS (Fiabilité, Maintenabilité, Disponibilité, Sécurité), en anglais RAMS (Reliability, Availability, Maintainability and Safety)... Elle se caractérise à la fois par les études structurelles statiques et dynamiques des systèmes, du point de vue prévisionnel, mais aussi opérationnel et expérimental (essais, accidents), en tenant compte des aspects probabilités et des conséquences induites par les défaillances techniques et humaines. Cette discipline intervient non seulement au niveau de systèmes déjà construits mais aussi au niveau conceptuel pour la réalisation des systèmes. Introduite en 1962 pour traduire le terme anglais reliability, la fiabilité est la probabilité de non défaillance d'un équipement sur un intervalle de temps donné (du latin fidare : faire confiance, fidus : fidèle et du latin médiéval fiablete ce qui est digne de confiance). La disponibilité se définit par la probabilité d'être en état d'accomplir sa fonction à un instant donné. Anglicisme introduit vers 1965, la maintenabilité est l'aptitude d'un

système à être maintenu en état. Elle correspond à la probabilité que la remise en état d'une entité en panne soit effectuée dans un intervalle de temps. Les mots sûreté et sécurité ont en fait la même racine étymologique (latin securus : sûr). La sécurité, en particulier en France, implique actuellement les aspects réglementaires de la sécurité des personnes. Le terme sûreté est plutôt utilisé par les techniciens pour la conception ou l'exploitation de biens et de services pour qualifier la fiabilité et la disponibilité du fonctionnement des installations. La Sûreté de fonctionnement s'est développée principalement au cours du 20e siècle pour être actuellement un domaine incontournable pour les industries à risques.

1. 4 La sûreté de fonctionnement informatique

On assiste de nos jours à une utilisation croissante dans divers domaines, des systèmes informatiques. Les défaillances de ces systèmes peuvent avoir des conséquences catastrophiques, aussi bien humaines, environnementales que économiques. Pour ces raisons, les aspects liés à la sûreté de fonctionnement des systèmes sont d'une importance primordiale. Or, la vitesse d'évolution de ces systèmes, entraîne une complexité d'utilisation énorme. Cette complexité va à l'encontre du problème de sûreté de fonctionnement en augmentant la probabilité d'avoir des éléments défaillants, ce qui rend ce problème de plus en plus difficile à appréhender. Il devient donc primordial d'étudier la sûreté de fonctionnement dans l'optique de proposer de nouvelles solutions mieux adaptées à l'évolution ainsi qu'aux spécificités de ces systèmes modernes.

Même si les systèmes informatiques, partagent des points communs avec les réalisations industrielles du début du siècle, ceux-ci nécessitent de développer des approches qui leur soient spécifiques.

Une tâche essentielle et partageable entre tous les domaines, a en particulier consisté à proposer un ensemble de notions claires pour appréhender la sûreté de fonctionnement indépendamment de la nature du système à laquelle elle s'applique.

1.5 Définition de la sûreté de fonctionnement

La sûreté de fonctionnement d'un système est son aptitude à délivrer un service de confiance justifiée. Cette définition mettant l'accent sur la justification de la confiance, cette dernière peut être définie comme une dépendance acceptée,

explicitement ou implicitement. La dépendance d'un système envers un autre système est l'impact, réel ou potentiel, de la sûreté de fonctionnement de ce dernier sur la sûreté de fonctionnement du système considéré [2].

La sûreté de fonctionnement informatique s'articule autour de trois principaux axes : les **Attributs** qui la caractérisent, les **entraves** qui empêchent sa réalisation et enfin les **moyens** de l'atteindre (la prévention, la tolérance, l'élimination et la prévision des fautes).

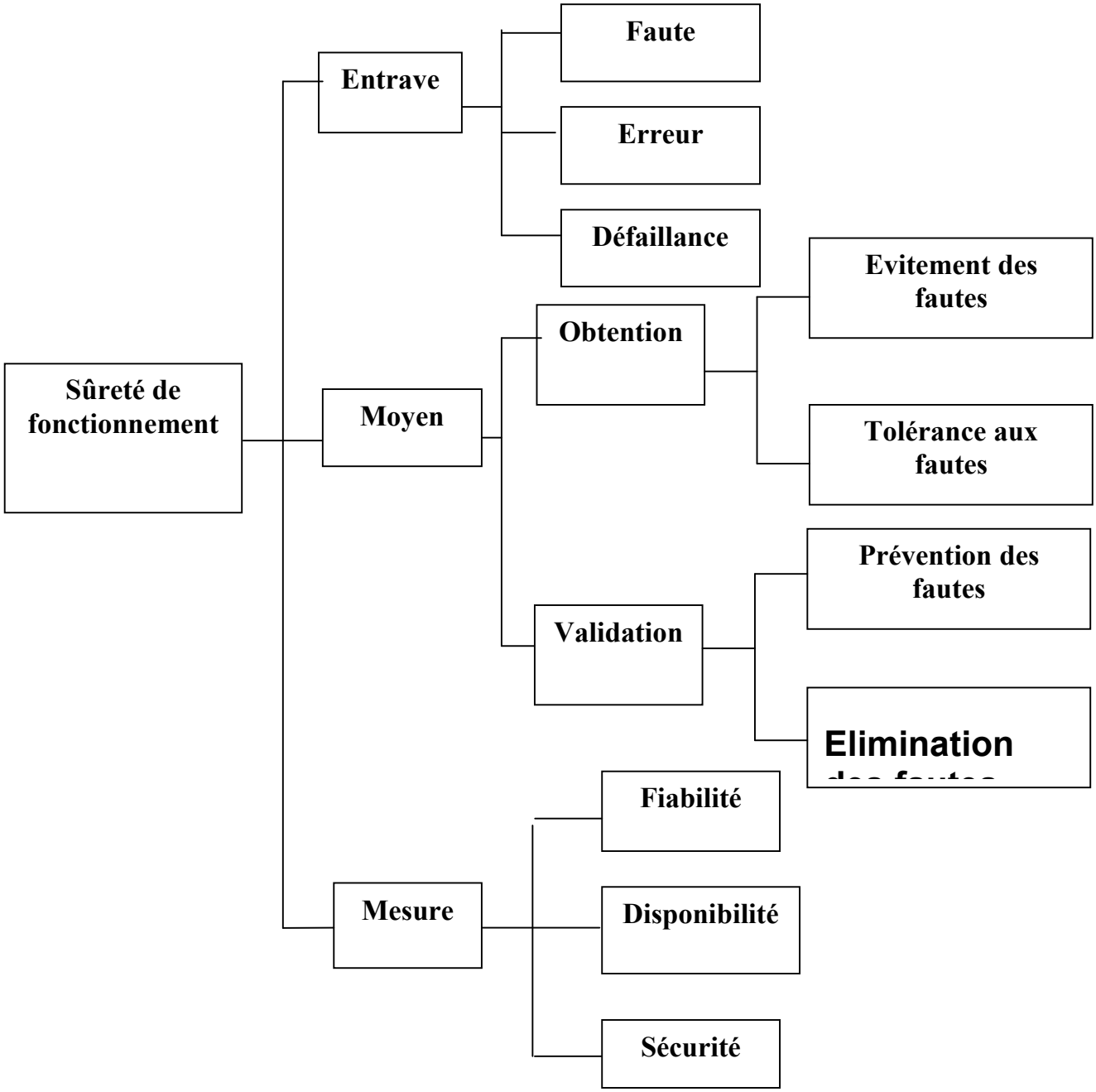


Figure 1[13] **:** Taxonomie de la sûreté de fonctionnement

1.6 Attributs de la sûreté de fonctionnement

Les attributs de sûreté de fonctionnement sont définis pour exprimer les propriétés de sûreté de fonctionnement du système. L'importance de chacun de ces attributs est relative aux applications auxquelles le système est destiné. On peut distinguer :

1.6.1 La fiabilité

La fiabilité (reliability) est définie comme la probabilité qu'un objet réalise la fonction requise dans les conditions établies pendant une période de temps donnée [3]. L'évaluation et la réalisation d'un niveau de fiabilité devient alors un problème très complexe [4].

Des études récentes sur la fiabilité des systèmes ont été réalisées dans le but de mettre en place des mécanismes de tolérance aux fautes afin d'augmenter le temps moyen jusqu'à la première panne MTTF (Mean Time To Failure) [5] [6].

1.6.2 La maintenabilité

La notion de maintenabilité (maintenability) va au-delà de la simple réparation ou changement des composants défaillants du système. En effet, elle peut aller de la prédiction de défaillances jusqu'à l'amélioration du système, en passant par la prévention des fautes. En définitive, la maintenabilité se conçoit comme la mesure de temps entre la dernière défaillance survenue et la restauration du système (transition de service incorrect à service correct) [7]. Un exemple de mesure de maintenabilité est le MTTR (Mean Time To Repair), le temps moyen de réparation ou de restauration du système dans l'état du bon fonctionnement.

1.6.3 La disponibilité

La disponibilité (availability) définit le fait que le système soit prêt à l'utilisation, ce qui importe, c'est que le service correct soit fourni au moment où l'utilisateur en a besoin [9] [10]. La disponibilité est une mesure sans unité, elle correspond à la proportion du temps de bon fonctionnement sur le temps total d'exécution du système. La disponibilité est tributaire à la fois de la fiabilité et de la maintenabilité.

1.6.4 La sûreté

La sûreté (safety) respecte la non occurrence de défaillance catastrophique [9], [10]. C'est-à-dire celles pour lesquelles les conséquences sont inacceptables vis-à-vis du

risque encouru par les utilisateurs du système. Accroître la sécurité d'un système peut parfois se faire au détriment de sa disponibilité. Dans le cas d'une centrale nucléaire, par exemple : L'arrêt d'un réacteur pour éviter la fusion du cœur.

1.6.5 La sécurité–confidentialité

La sécurité (security), prévoit la non occurrence des accès des utilisateurs non autorisés ou l'acquisition non autorisée des informations [9],[10], ou de façon plus générale, si la préoccupation essentielle est de lutter contre les fautes intentionnelles (virus, bombes logiques, chevaux de Troie,...).

La fiabilité, la disponibilité et la sûreté innocuité relèvent du domaine de la tolérance aux fautes, la dernière propriété concerne les attaques. Les propriétés sur lesquelles repose la sûreté de fonctionnement peuvent être considérées de manière indépendante mais elles sont étroitement liées [14].

1.7 Entraves à la sûreté de fonctionnement

Lors du développement d'un système informatique (distribué ou non), des **fautes** peuvent être introduites au cours de sa conception et/ou de son implémentation sans être corrigées lors de la phase de validation. Les **erreurs** correspondantes restent latentes pendant la durée de vie du système jusqu'à ce qu'elles soient activées. Par exemple, dans le système de gestion d'archivage du véhicule martien SPIRIT, l'oubli de la libération de l'espace mémoire après l'envoi des images sur la terre qui représente une erreur latente n'a été activée que lorsque le système d'acquisition avait saturé la zone de stockage des données, ceci a conduit à deux semaines d'indisponibilité.

1.7.1 Le passage de la faute à l'erreur

L'origine d'une **erreur** est une faute, qui dans la plupart des cas est d'ordre conceptuel ou physique. En d'autres termes, on peut souvent ramener l'origine d'une erreur à une faute humaine au niveau de la conception d'un système ou d'une faute physique d'un composant matériel, et ce, de façon causale. L'activation d'une erreur d'un composant d'un système distribué peut affecter le service qu'il rend, auquel cas le composant est déclaré **défaillant**.

Par ailleurs, la défaillance d'un composant représente une **faute** auprès d'autres composants du système avec lesquels il interagit. Les défaillances, les erreurs et les fautes représentent les **entraves** à la sûreté de fonctionnement.

Les entraves à la sûreté de fonctionnement sont les circonstances indésirables mais inattendues causes ou résultats de la non sûreté de fonctionnement. Dans l'ensemble des entraves, on distingue comme cela a déjà été dit, la défaillance du système qui survient lorsque le service délivré dévie de l'accomplissement de la fonction du système. L'erreur est la partie de l'état de système qui est susceptible d'entraîner une défaillance, et enfin la faute est considérée comme la cause adjugée ou supposée de l'erreur, illustration dans la figure 2.

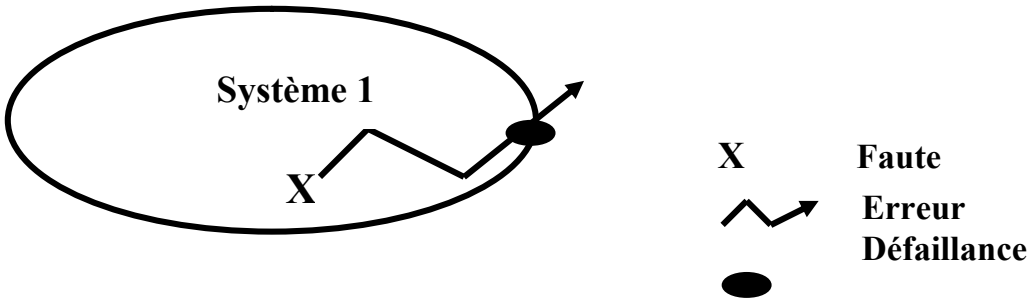


Figure 2 [15] **:** Faute, erreur et défaillance.

1.7.2 La propagation des fautes

Un système est usuellement composé d'autres systèmes plus petits, qui lui délivrent leurs services, ou, est toujours en interaction avec d'autres systèmes. La défaillance d'un de ces systèmes devient alors une faute pour les autres. Voici comment se fait la propagation ou la transitivité de l'erreur dans la figure 3.

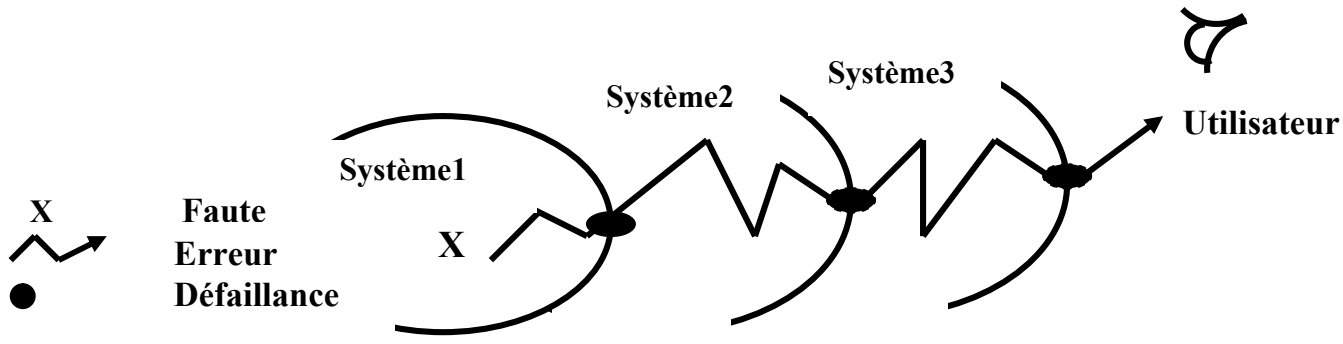


Figure 3[15] : La Propagation des fautes, erreurs et défaillances.

1.8 Moyens assurant la sûreté de fonctionnement

1.8.1 La prévention des fautes

Elle vise à empêcher l'occurrence ou l'introduction de fautes [8], [11]. Elle consiste à développer des systèmes informatiques de telle façon à éviter d'introduire des fautes de conception, et à empêcher que les fautes ne surviennent en phase opérationnelle.

1.8.2 La tolérance aux fautes

Elle correspond à un ensemble de moyens, destinés à assurer qu'un système remplit sa fonction en dépit des fautes [8], [10], [11]. Elle cherche à éviter que les fautes présentes dans le système n'engendrent des défaillances en phase opérationnelle. Les techniques associées ont pour but de compenser l'occurrence d'erreur en utilisant par exemple des méthodes basées sur la duplication de matériel ou logiciel (N-version) ou de traiter l'erreur (Reprise).

1.8.3 L'Elimination des fautes

Elle vise à réduire le nombre ou la sévérité des fautes [10], [8]. Elle cherche à détecter la présence de fautes puis à les localiser dans le but de les extraire ensuite. Les diverses techniques de test (fonctionnelles ou structurelles) répondent par exemple à ces objectifs.

1.8.4 La prévision des fautes

Elle consiste à estimer par évaluation le nombre de fautes courantes et futures ainsi que leurs conséquences [10], [8]. Elle cherche à prévenir la présence de fautes ou l'occurrence de pannes touchant la structure du système. Ceci est obtenu par exemple en contraignant la démarche de conception par des règles régissant celle-ci.

On peut même avoir des combinaisons entre deux ou plusieurs méthodes selon l'objectif requis par exemple :

- La prévention et l'élimination, pour un système exempt de fautes.
- La prévention et la tolérance, pour 1' obtention de la sûreté.
- L'élimination et la prévision, pour la validation de la sûreté.

La prévention des fautes et la tolérance aux fautes peuvent être vues comme des moyens d'obtention de la sûreté de fonctionnement, c'est-à-dire comment procurer au système l'aptitude à fournir des services conformes aux fonctions attendues.
L'élimination des fautes et la prévision des fautes peuvent être vues comme constituant les moyens de la validation de la sûreté de fonctionnement, c'est-à-dire comment avoir confiance dans l'aptitude du système à fournir des services conformes à l'accomplissement de sa fonction.

Conclusion

La sûreté de fonctionnement est née avec les premières inventions de l'homme, puisque celui-ci est devenu dépendant de leur bon fonctionnement. On ne tâtait pas son effet car la complexité du système était maîtrisable. Mais à mesure que la complexité augmentait, cette dépendance, prenait une grande envergure obligeant l'être humain à considérer la sûreté de fonctionnement comme étant une discipline à part entière ou la notion de confiance est fondamentale (étant donné que tout système matériel /logiciel contient des fautes). Donc, on ira jusqu'à dire que la sûreté de fonctionnement est la science des défaillances.

Il n'existe désormais pratiquement aucun domaine d'activité industrielle dans les pays développés où la sûreté de fonctionnement n'est pas prise en compte (même partiellement). La sûreté de fonctionnement constitue une discipline scientifique à part entière pour ceux qui ont le désir de s'approprier les méthodes telles que l'analyse fonctionnelle et les outils de la fiabilité, disponibilité, maintenabilité et sûreté [12].

Chapitre2

Entraves à la sûreté de fonctionnement.

2.1 Introduction

Qu'elle soit issue d'une mauvaise conception, de perturbations de l'environnement, etc. la faute provoque un dysfonctionnement interne du système caractérisé par un état non désiré, appelé erreur. La défaillance constitue alors l'effet de cette erreur sur les interactions du système avec son environnement. Donc c'est l'existence inévitable de ces phénomènes parfois à effets désastreux qui nous ont poussé à nous intéresser de plus prêt à un remède efficace.

Pour un utilisateur unique, un système peut défaillir en valeur ou temporellement. En effet, si la valeur du service délivré ne permet plus l'accomplissement de la fonction du système, ce dernier est déclaré défaillant par valeur. D'autre part, si les conditions de délivrance du service ne permettent plus l'accomplissement de la fonction du système, soit d'une façon permanente (arrêt) ou de façon transitoire (dépassement d'échéance), celui-ci est considéré comme défaillant temporellement.

En revanche, dans le cas où plusieurs utilisateurs pourraient demander la même fonction auprès d'un service, la défaillance d'un composant peut être perçue comme :

- **Cohérente** : tous les utilisateurs du système ont la même perception des défaillances.
- **Byzantine (incohérente)** : les utilisateurs du système peuvent avoir des perceptions différentes des défaillances [16].

2.2 Terminologie

Pour plus de clarté, et afin d'éviter toute ambiguïté, nous définissons ci-après certains concepts [14], [15] :

2.2.1 Défaut (Defect)

C'est un changement physique matériel ou conception incorrecte du logiciel [14].

2.2.2 Faute (Fault)

C'est un état erroné du matériel ou de logiciel résultant de défauts de composants ou d'une conception incorrecte. La faute reste invisible jusqu'à ce qu'un test (opération) la détecte. Elle devient active lorsqu'elle produit une erreur.

Une faute active est :

- ❖ soit une faute dormante activée par le traitement ;
- ❖ soit une faute externe ;

  Une faute interne peut passer cycliquement de l'état donnant à actif, etc.

2.2.3 Erreur (Error)

C'est la manifestation d'une panne (faute). Autrement dit, l'effet généré par l'opération (test) quand elle touche la faute (panne). Temporaire par nature, latente ou détectée :

- ❖ Latente, tant qu'elle n'a pas été reconnue.
- ❖ Détectée, soit par les mécanismes de détection et traitement d'erreurs, soit par son effet observé sur le service (défaillance).

Une erreur peut impliquer la propagation d'autres erreurs dans d'autres parties du système.

2.2.4 Défaillance (Faillure)

C'est la manifestation de l'erreur vers l'extérieur d'un composant (résultats erronés). Une défaillance survient lorsqu'une erreur traverse l'interface Système/ Utilisateur et altère le service délivré par le système.

Et nous aurons toujours le cycle :

... Défaillance .... Faute .... Erreur... Défaillance.... Faute...

Pour plus de précisions, nous allons dire qu'un défaut est une condition ou un état physique anormal dû à des aléas de conception et des problèmes de fabrication ou causé par des environnements intempestifs : des interférences électromagnétiques, etc. Lorsqu'un système est défectueux (c.-à-d. en défaut), il entre dans un état erroné, il donne naissance à une panne. Remarquons qu'une panne peut être logique ou non. Par exemple, suite à un défaut de la mémoire, un accès en lecture ou en écriture peut provoquer l'arrêt anormal du processus. Une erreur est alors la manifestation d'un défaut et peut conduire le système vers une défaillance. Effectuer une reprise revient à rétablir le système dans l'état qui précède l'apparition du défaut.

2.3 Classification des erreurs

2.3.1 Classification selon le franchissement de la frontière du système

Dans un système constitué d'un ensemble de composants, la conséquence de la défaillance d'un composant est :

2.3.1.1 Les fautes internes

Pour le composant englobant, une défaillance perçue au niveau d'un des éléments le composant est considérée comme une faute interne.

2.3.1.2 Les fautes externes

Les fautes internes d'un composant sont perçues comme une faute externe pour les composants avec lesquels il interagit, au sein d'un même système.

2.3.2 Classification selon leur persistance

2.3.2.1 Les fautes Permanentes

Les fautes permanentes sont des fautes qui demeurent toujours, c'est un défaut irréversible [10].

Exemples :

Panne franche de processeur, coupure de voie physique, certains types de programmes erronés (exemple boucle), système d'exploitation interbloqué.

2.3.2.2 Les fautes transitoires

Elles ont lieu suivant des conditions temporaires de l'environnement. En réponse à un événement (le composant ne peut fournir son service habituel pendant une certaine période => perte de quelques données). Ultérieurement il peut répondre à nouveau de façon correcte.

Exemples :

- Perte temporaire de messages sur une voie physique.
- Destruction de transaction ou de message pour éviter l'interblocage ou l'écroulement.

2.3.3 Classification selon leur nature

Tout comportement s'écartant des spécifications (principalement parce que les résultats sont non conformes) est qualifié de comportement byzantin. On distingue quelquefois :

2.3.3.1 Les fautes byzantines naturelles

Par exemple erreur physique non détectée (sur une transmission de message, en mémoire, sur une instruction), erreur logicielle amenant une non vérification des spécifications.

2.3.3.2 Les fautes byzantines malicieuses

Ce sont des fautes injectées dans le but d'avoir un comportement visant à faire échouer le système, par exemple : sabotage, virus...

2.3.4 Classification selon leur origine

2.3.4.1 Les pannes Logicielles

Elles sont dues à une conception incorrecte d'un logiciel. Elles n'ont aucune influence directe sur le matériel. La détection des erreurs générées par ces pannes ou fautes peut s'avérer très coûteuse, car il est nécessaire d'avoir pour chaque erreur possible un test de détection approprié et un traitant (comme le mécanisme des blocs de recouvrement et conversation développé par Randell [29]).

Remarque :

Les défauts logiciels sont toujours des défauts permanents, leur traitement nécessite la duplication de modèles de logiciels**.** Mais, les fautes logicielles peuvent être permanentes ou transitoires (les plus fréquentes). Car, une exécution d'un processus dépend des :

1) Données de l'utilisateur.

2) Variables de l'environnement, comme le cas des variables (ressources) partagées.

3) Les messages incluant ceux des événements et ceux des signaux, qui arrivent au processus : leur ordre, le temps d'arrivée, etc.

2.3.4.2 Les pannes matérielles

Elles peuvent être transitoires dont l'occurrence, peut se traduire par une altération de l'environnement interne du processus affecté (due à des phénomènes différents telles que poussière, rayons cosmiques, etc.) et éventuellement de celui de ses partenaires par effet de contamination (cette contamination s'observe lorsque une propagation d'erreurs a lieu). Le composant matériel affecté passe d'un état de fonctionnement normal à un état anormal et réciproquement. Comme elles peuvent être permanentes dues à des phénomènes physiques permanents (courts-circuits,...) Contrairement aux premières, un composant endommagé se voit dans l'incapacité d'assurer la continuité de ses services. La seule solution est la redondance matérielle et qui ne peut en général être assurée que dans les systèmes distribués.

L'occurrence des pannes matérielles est irrégulière, mais celle des pannes transitoires est très fréquente et mois coûteuse que celle des pannes permanentes.

2.3.5 Classification selon leur phase d'occurrence

2.3.5.1 Les fautes conceptuelles

Ce sont des fautes qui font surface dans la phase conceptuelle d'un système par exemple lors du développement d'un logiciel.

2.3.5.2 Les fautes opérationnelles

Ce sont des fautes dues aux manipulations de l'utilisateur sur le système.

2.4 Classification des erreurs

Une erreur étant définie comme susceptible de provoquer une défaillance. Elle peut être perçue dans :

- La composition du système.
- L'activité du système.

2.5 Classification des défaillances

2.5.1 Les défaillances byzantines

Quand le système dévie de ses spécifications et délivre des services non-conformes à ces dernières, son comportement sera caractérisé de byzantin.

2.5.2 Les défaillances temporelles

Lorsqu' une sortie correcte associée à une requête entrante se manifeste de façon incohérente avec les spécifications du système :

- Trop tard ou jamais.
- Trop tôt.

Dans ce cas nous avons une défaillance temporelle.

Exemples :

- Surcharge d'un processeur.
- Horloge trop rapide.

2.5.3 Les défaillances par omission

Nous pouvons dire que nous avons une défaillance par omission lorsque le système omet de délivrer un service.

2.5.4 Les défaillances par arrêt

Lorsque une défaillance par omission devient permanente, nous nous trouverons devant le cas le plus extrême qui est l'arrêt du système.

La figure 4 présente les classes de défaillances, des moins graves aux plus graves.

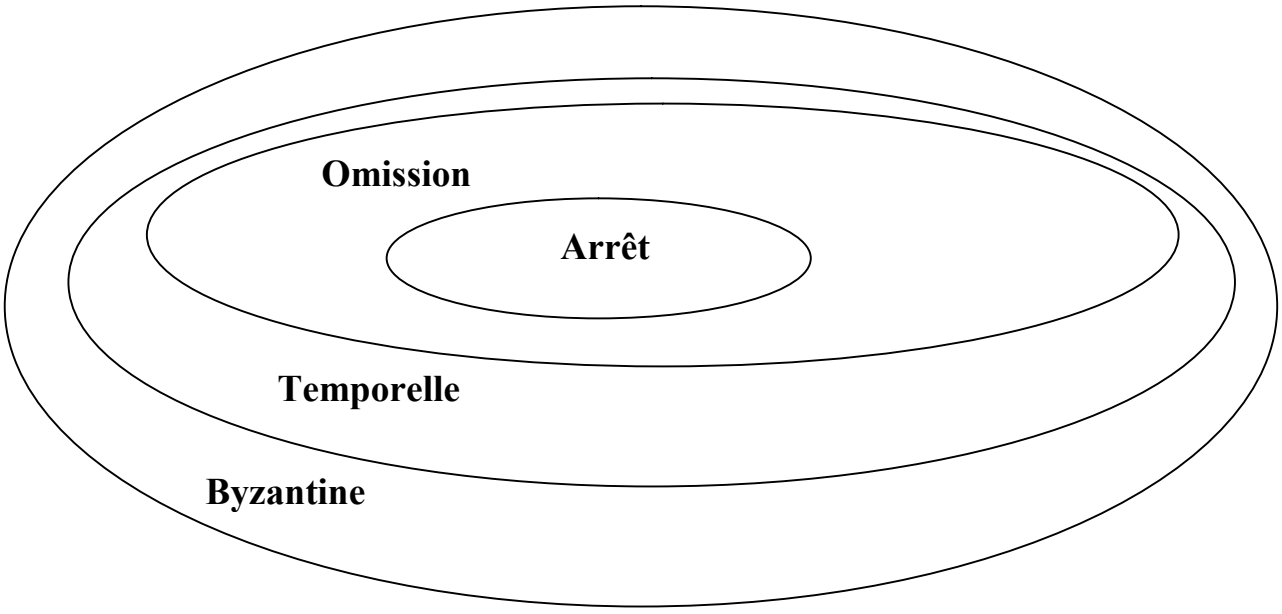


Figure 4 : Classes de défaillance.

Conclusion

Il n'existe pas de système exempt de fautes. Ces fautes peuvent provenir du système lui-même ou de son environnement .La faute est une évidence, il est donc important si ce n'est nécessaire d'évaluer le risque encouru par les utilisateurs d'un système afin de déterminer les moyens qu'il faut mettre en œuvre pour éliminer ou dans une moindre mesure tolérer la faute.

Toutes les fautes ne peuvent être évitées, quelles que soient les précautions prises. L'occurrence de fautes est inévitable. Cela ne veut pas dire qu'il ne faut pas essayer de réduire la probabilité de leur occurrence et l'effet de leurs conséquences.

Chapitre3

Introduction à la tolérance aux fautes.

« Spécifier, concevoir, réaliser et exploiter des systèmes où la faute est naturelle, prévue, et tolérable. » [17].

3.1 Introduction

L'un des objectifs poursuivis par les systèmes répartis est la tolérance aux fautes. Tant qu'un des sites est opérationnel, le système devrait continuer à fonctionner. Quelles que soient les précautions prises pour éviter l'introduction de fautes, et les efforts fournis pour les détecter et les corriger, les fautes qui ont échappé à la prévention, finiront par se produire (erreurs humaines, malveillances, vieillissement du matériel, catastrophes naturelles…). Pour être sûr de fonctionnement, un système doit donc comporter des mécanismes de tolérance qui lui permettent de rester opérationnel dans les cas ou les fautes qu'il comportent se manifestent par des erreurs. Le système doit alors traiter les erreurs avant qu'elles ne donnent lieu à des défaillances.

3.2 Définition de la tolérance aux fautes

La tolérance aux fautes, est une méthode qui permet à un système de remplir ses fonctions en dépit des fautes pouvant affecter ses composants, sa conception ou ses interactions avec des hommes ou avec d'autres systèmes. C'est fournir un service conforme aux spécifications en dépit de la présence ou de l'occurrence de fautes [18].

3.3 Techniques permettant d'atteindre la tolérance aux fautes

La tolérance aux fautes est mise en oeuvre par [19] :

- Le traitement de la faute (fault treatment) qui vise à éviter qu'une faute survenue ne se reproduise.
- Le traitement d'erreur (error processing) qui vise à éliminer une erreur avant qu'elle ne produise une défaillance

3.3.1 Traitement de faute

L'objectif du traitement de l'erreur est d'éliminer une erreur affectant le système afin qu'elle n'entraîne pas de défaillance.

Les étapes du traitement des fautes sont :

- Diagnostic de faute, qui consiste à déterminer les causes des erreurs en termes de localisation et de nature.
- Passivation des fautes ou actions destinées à empêcher une nouvelle activation des fautes. Cette tâche est accomplie en retirant les composants considérés comme

fautifs du processus d'exécution ultérieur. Cette étape n'est pas nécessaire si l'on estime que la faute n'est plus présente après le traitement d'erreur ou si sa probabilité de récurrence est suffisamment faible.

- Reconfiguration, qui comprend la modification de la structure du système, de telle sorte que les composants non défaillants permettent de rendre un service acceptable, bien que dégradé, si le système ne peut plus délivrer le même service qu'avant. La reconfiguration peut engendrer l'abandon de certaines tâches ou la réallocation de certaines tâches aux composants restants.

3.3.2 Traitement de L'erreur

Elle se base principalement sur la détection de l'existence d'un état incorrect (erreur) et le remplacement de l'état incorrect par un état correct (conforme aux spécifications). Le traitement de l'erreur passe par plusieurs étapes :

- La détection de l'erreur ou de la défaillance qui permet d'identifier ou de définir un état erroné.
- Le diagnostic de l'erreur, qui estime les dégâts causés par l'erreur détectée et par sa propagation.
- Le recouvrement d'erreur qui consiste à ramener l'état erroné à un état correct, fonctionnel avant l'occurrence de la faute.

Et peut s'exprimer sous deux formes :

- Le recouvrement d'erreur consiste à remplacer l'état erroné du système par un état correct.
- La compensation d'erreur consiste à compter sur la redondance présente dans le système pour que celui-ci continue à délivrer un service correct malgré un état erroné.

3.3.2.1 Recouvrement d'erreur (error recovery)

Pour recouvrer d'une erreur, le système doit être capable de substituer un état correct à l'état erroné [20]. Il existe plusieurs méthodes pour réaliser le recouvrement : la reprise et la poursuite.

- La reprise : Elle consiste à remplacer l'état erroné par un état correct dans lequel le système était avant l'occurrence de l'erreur.
- La poursuite : Elle consiste à remplacer l'étal erroné par un nouvel état correct construit a partir de l'état erroné.

3.3.2.1.1 La reprise

Retour en arrière vers un état antérieur présumé correct. On demande de faire régulièrement de sauvegardes de l'état du système appelé point de reprise. Pour transformer un état erroné en un état correct, on réinitialise l'état du système à partir d'un point de reprise, présumé correct.

La méthode de reprise consiste en une sauvegarde périodique ou non de l'état du système de façon à pouvoir, après avoir détecté une erreur, ramener le système à un état antérieur, supposé exempt d'erreurs. Cette définition se base sur la définition de point de reprise. Le passage à l'un de ces points, entraîne la sauvegarde de l'état au moyen d'un mécanisme de mémorisation qui protège l'information des effets de fautes que l'on essaie de tolérer.

3.3.2.1.2 La poursuite

Le recouvrement par poursuite consiste à la recherche d'un nouvel état acceptable pour le système à partir duquel celui-ci pourra fonctionner. C'est une tentative de reconstitution d'un nouvel état correct à partir de l'état erroné, sans retour arrière. La reconstitution est souvent seulement partielle, d'ou service dégradé. Elle dépend de l'application et doit être prise en charge lors du développement de cette dernière (les exceptions).

Comparaison entre reprise et poursuite

Les mécanismes transactionnels offrent un support élégant adapté au fonctionnement par reprise. L'approche par poursuite peut se faire au travers des traitements d'exceptions.

3.3.2.2 La compensation d'erreur (error masking)

Elle se fait grâce à l'utilisation de redondances pour permettre au système de continuer à fournir le service correct en dépit de fautes. Le principe consiste à dupliquer voire tripler les équipements ou composants logiciels les plus " critiques " et ceux contribuant aux services les plus importants à fournir aux utilisateurs. Une gestion de vote majoritaire ou de basculement sur des équipements " secours " est alors mise en oeuvre.

Elle peut prendre deux formes

- La détection et compensation d'erreur.
- Le masquage d'erreur.

3.3.2.2.1 La détection et compensation d'erreur

Elle se fait suite à une erreur et consiste à remplacer le composant erroné par un composant correct. Le masquage d'erreur consiste en une compensation d'erreur. La compensation est déclenchée sur détection d'erreur. L'état du système est suffisamment redondant pour qu'il soit inutile de ré exécuter une partie du traitement, par exemple le code correcteur d'erreur : la validité de la valeur codée est vérifiée en permanence, en cas de détection d'erreur, l'algorithme de correction est lancé.

3.3.2.2.2 Le masquage

C'est une compensation systématique, les traitements se font sur 3 ou plusieurs composants identiques dont les sorties sont votées, seuls les résultats majoritaires sont transmis. Si les exemplaires sont identiques et synchronisés, et si le traitement est déterministe l'algorithme de vote peut être simple. Le vote étant appliqué systématiquement, le temps d'exécution est identique qu'il y ait ou non erreur.

3.4 Méthodes d'implémentation de la tolérance aux fautes

Plusieurs questions se posent concernant l'implémentation de tolérance aux fautes : comment détecter l'erreur, cibler ses dégâts, la diagnostiquer et recouvrir l'état erroné et beaucoup d'autres choses. Tous ces aspects sont très importants mais ce qui nous intéresse est le recouvrement de l'erreur.

En général, le recouvrement d'erreur dans les systèmes informatiques est une tache qui demande de restaurer un état correct à partir d'un autre état erroné. Il est classé en deux catégories : recouvrement avant (Forward recovery) et le recouvrement arrière (Backward recovery) [21].

3.4.1 Recouvrement avant (Forward recovery)

Il est basé sur l'identification de la nature et la connaissance des conséquences de l'erreur. Il remédie à la défaillance par correction de l'état erroné du système en agissant seulement sur la partie endommagée puis il mène le système en avant pour qu'il continue son travail. Cette démarche nécessite au préalable une évaluation précise des dommages subis par le système. Les exceptions et leur traitement en sont des mécanismes de récupération d'erreur qui adoptent une telle approche.

Généralement, elle utilise les codes correcteurs d'erreurs et des moyens de compensation d'où l'utilisation d'une redondance de composants (matériel ou logiciel),

c à d avoir plusieurs composants qui exécutent la même fonction et contenant des copies d'informations. Son utilisation est très coûteuse, et nécessite des systèmes spéciaux qui gèrent cette redondance (elle n'est pas d'usage général).

3.4.2 Recouvrement Arrière (Backward recovery)

Contrairement au recouvrement avant, on n'a pas à avoir une idée sur l'erreur, ni les moyens de la traiter, tout simplement nous devons essayer de ramener le système à un état antérieur sauvegardé présumé correct.

Il se caractérise par une indépendance vis-à-vis de l'estimation et la prédiction des dommages causés par l'erreur, et vis-à-vis de l'application elle-même. D'ou, le recouvrement de l'erreur consiste à considérer que l'état du système affecté ne peut être corrigé par suppression ou isolement de cette erreur, c à d il n'est pas possible de corriger l'état erroné du système en agissant seulement sur la partie endommagée. Il convient alors (pour corriger l'état du système), de restaurer l'étal du système (présume correct) antérieur à l'instant de la panne, et c'est à partir que cet état que la relance du processus aura lieu.

3.4.3 Comparaison entre les deux techniques

- l'implantation de recouvrement en avant demande des systèmes spéciaux qui sollicitent une redondance de composants et une possibilité de prédiction et d'identification des erreurs qui peuvent affecter le système. Donc, il n'est pas possible de généraliser l'implantation de cette approche dans tous les systèmes. Il existe une relation entre le mécanisme de reprise et les applications des programmeurs (pour les traitements des exceptions), donc l'utilisateur doit développer ses applications avec considération de ce mécanisme [22]. Ce type de recouvrement est plus spécialisé dans le traitement des fautes logicielles. Tandis que le recouvrement arrière est standard pour tous les systèmes, n'est pas exigeant et supporte aussi bien le traitement des pannes matérielles que celui des fautes logicielles.

- La transparence des mécanismes de recouvrement arrière par rapport à l'utilisateur [22] [23].

Conclusion

Beaucoup de mécanismes de tolérance aux fautes ont été proposés chacun pour des fautes particulières. Mais les solutions proposées donnent leur meilleure exécution seulement dans un environnement particulier d'exécution ou ils réalisent un sous ensemble des objectifs de l'utilisateur mais pas tous. D'ailleurs, les solutions ne considèrent pas les autres fautes que l'application pourrait rencontrer en même temps. La raison est que chaque faute possède ses propres caractéristiques. Donc il n'est pas possible de satisfaire tous les besoins d'utilisateur avec un seul mécanisme de tolérance aux fautes.

Deuxière Partie

Recouvrement arrière dans les systèmes répartis fixes.

Chapitre4

Introduction au recouvrement arrière.

4.1 Introduction

Les applications informatiques réparties sont omniprésentes dans tous les secteurs de l'économie. Désormais, un calculateur se trouve rarement isolé du reste du monde. Au contraire, il est implicitement intégré dans un réseau (local ou de plus grande ampleur) et la plupart des calculs qui lui sont soumis nécessitent une coopération avec d'autres calculateurs connectés de manière permanente ou temporaire. De façon générale, le paysage informatique actuel se caractérise donc par une interconnexion massive et évolutive de calculateurs hétérogènes, géographiquement distants.

Cet avènement de l'informatique répartie, qui a été rendu possible grâce aux progrès technologiques réalisés, est également le fruit de l'intense activité de recherche qui au cours de ces dernières années a eu pour objectif de maîtriser les problèmes fondamentaux propres aux systèmes répartis dont le problème majeur consiste à garantir sa sûreté de fonctionnement.

4.2 Les systèmes distribués

Les systèmes informatiques distribués sont composés de plusieurs calculateurs reliés par un réseau de communication. Ces calculateurs ne partagent ni mémoire ni horloge communes, cela nous induit à une impossibilité d'observation instantanée d'un état global. Ils sont modélisés par un ensemble de processus qui interagissent entre eux par l'échange de messages, donc un calculateur peut engager deux primitives : l'envoie ou la réception d'un message.

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

Quelles que soient les précautions prise pour éviter les fautes et les efforts fournis pour détecter et corriger les fautes, elles seront toujours présentes dans le système. Par ailleurs, nous devons être réalistes et accepter le fait qu'elles soient inévitables et par conséquent, essayer de cohabiter avec la faute.

Pour être sûr de fonctionnement, un système doit comporter des mécanismes de tolérance aux fautes, d'une part pour lui permettre de rester opérationnel même en présence de fautes d'autre part vu la multiplicité des ressources gérées par celui-ci.

4.3.1 Propriétés d'un système tolérant aux fautes

Un système tolérant aux faute doit satisfaire deux propriétés : La sûreté et la vivacité.

4.3.1.1 La sûreté

La sûreté est une propriété continue, garantie par la détection des fautes.

4.3.1.2 La vivacité

La vivacité est une propriété éventuelle, elle est assurée par la substitution de l'état erroné par un nouvel état présumé correct.

4.3.2 Fonction d'un système tolérant aux fautes

La tolérance aux fautes est équipée de composants et de programmes additionnels, ayant pour but d'assurer l'atténuation de l'effet des états erronés pour qu'ils ne deviennent jamais des défaillances menaçant le système.
Pour qu'un système soit tolérant aux fautes, il doit être capable d'assurer au moins les trois actions suivantes :

4.3.2.1 La détection de l'erreur

Elle consiste à détecter qu'une erreur (défaillance) s'est produite ou qu'un composant ou un processus est fautif (défaillant), son but est de confiner l'erreur pour qu'elle ne puisse pas se propager vers d'autres composants. Elle est assurée par plusieurs mécanismes.

La plupart des mécanismes de détection d'erreurs (de défaillances de composants ou processus) reposent sur l'utilisation d'un oracle. Un oracle étant un composant distribué que les processus peuvent consulter pour faire un choix, en particulier, pour savoir si un autre processus est défaillant ou pas. En algorithmique distribuée, deux principaux types d'oracles sont utilisés : les détecteurs de défaillances et les évènements aléatoires.

4.3.2.1.1 Les détecteurs de défaillances

Les détecteurs de défaillances ont été introduits par Chandra et Toueng [61], ce sont des oracles permettant de distinguer les processus corrects de ceux défaillants. Il se comporte comme un processus gardien qui surveille l'émission périodique des messages de vie (I am alive) [62], provenant d'un ou de plusieurs processus. Si ces messages ne sont pas reçus à temps par le gardien, cela implique la présence de faute (erreur ou défaillance) dans le composant surveillé. Chaque processus Pi du système est équipé d'un module FDi du détecteur de défaillance qui maintient une liste de processus qu'il suspecte d'être défaillants. Dans un système asynchrone, ces détecteurs deviennent non fiables parce que leurs suspicions peuvent être erronés ou inconsistantes. Ceci peut être

expliqué par la difficulté de différencier entre un processus extrêmement lent est un processus qui a définitivement stoppé ses activités.

4.3.2.1.2 Les événements aléatoires

Un oracle aléatoire est capable de générer des valeurs aléatoires [63] en l'absence d'informations certaines, les valeurs aléatoires permettent aux processus de faire un choix (processus correct ou incorrect) avec une probabilité qui tend vers 1.

4.3.2.2 Le diagnostic de l'erreur

Il consiste à déterminer l'origine de l'erreur ou du composant erroné et estime les dégâts causés par cette dernière.

4.3.2.3 Le recouvrement de l'erreur

Il consiste à masquer l'état (ou le composant) erroné en le remplaçant par un état correct.

Comme déjà évoqué dans la première partie de notre étude, chapitre 3 section 3.4, il est classé en deux catégories : recouvrement avant et le recouvrement arrière.
Pour les raisons synthétisées dans la section 3.4.3 du chapitre 1 de la première partie, nous nous verrons dans l'obligation de choisir le recouvrement arrière.
Nous pouvons dire que la technique du recouvrement arrière est la plus convenable connue jusqu'ici pour garantir la tolérance aux fautes dans les systèmes répartis, vu :

- ✓ Son coût réduit.
- ✓ La prise en charge de tout type de pannes.
- ✓ Sa généralité.

4.4 Description du mécanisme de recouvrement arrière

Le recouvrement de l'erreur (correction de l'état erroné du système), consiste à restaurer un état du système (présumé correcte) antérieur à l'instant d'erreur, à partir duquel une relance aura lieu [23].

Dans l'état normal, le seul état qui peut satisfaire cette opération est l'état initial du système, car tous les états intermédiaires entre l'état initial et l'état erroné disparaissent. Donc, l'étal initial du système est un état correct qui peut être restauré

pour relancer le système, mais le fait de réinitialiser tout le système dès le début à chaque fois qu'une erreur ait lieu peut entraîner un overhead très important et par conséquent causer des catastrophes sur tout dans les systèmes temps réel ou le temps d'exécution des travaux est très important [24].

Pour éviter ces catastrophes et permettre un recouvrement de l'erreur sans relancer le système dés le début, il faut sauvegarder des états entiers différents (à des instants différents) du système et ce, pendant l'exécution normale [23], [25], [26]. Donc ces états sont présumés corrects, on les appelle points de reprise ou (Checkpoints). En cas d'erreurs décelées, on restaure le plus récent et cohérent point de reprise et on relance l'exécution du système. L'opération de relance du système à partir de l'un des états sauvegardés (point de reprise) s'appelle : opération de reprise (recovery opération). Donc Le recouvrement arrière passe par trois étapes :

- ✓ La prise d'un Checkpoint.
- ✓ l'enregistrement ou le stockage de ce dernier.
- ✓ Le recouvrement à partir de ce Checkpoint.

4.5 Quelques définitions utiles

4.5.1 L'état global d'un système réparti

Définition 01 : Un **état global** d'un système réparti est un ensemble d'états locaux dans lequel on trouve un enregistrement d'état local par processus.

Définition 02 : L'**exécution répartie** d'une application à partir de l'état global est constituée de la fusion suivant un temps absolu des exécutions des processus. Par déduction, **l'histoire répartie** d'une exécution répartie est constituée de la fusion suivant un temps absolu des histoires séquentielles. Par définition, l'histoire des messages d'une exécution répartie est égale à l'histoire répartie de laquelle toutes les actions internes sont retranchées. Dans la pratique, les actions d'émission et de réception sont repérées respectivement par des numéros d'ordre d'émission et de réception. L'histoire d'un message est alors constituée du quadruplet (Identité de l'émetteur, Numéro d'ordre d'émission, Identité du récepteur, Numéro d'ordre de réception).

Définition 03 : Dans la figure 5, nous montrons trois processus ayant chacun pris un enregistrement local de leur état. Pourtant, l'ensemble de ces trois points ne représente pas un **état global cohérent.** En effet, un état cohérent est un état par lequel peut passer une exécution normale. Or, on voit ici que le message m4 a déjà été reçu par le processus R, alors qu'il n'a pas encore été enregistré comme envoyé par Q. On parle alors de **message orphelin**.

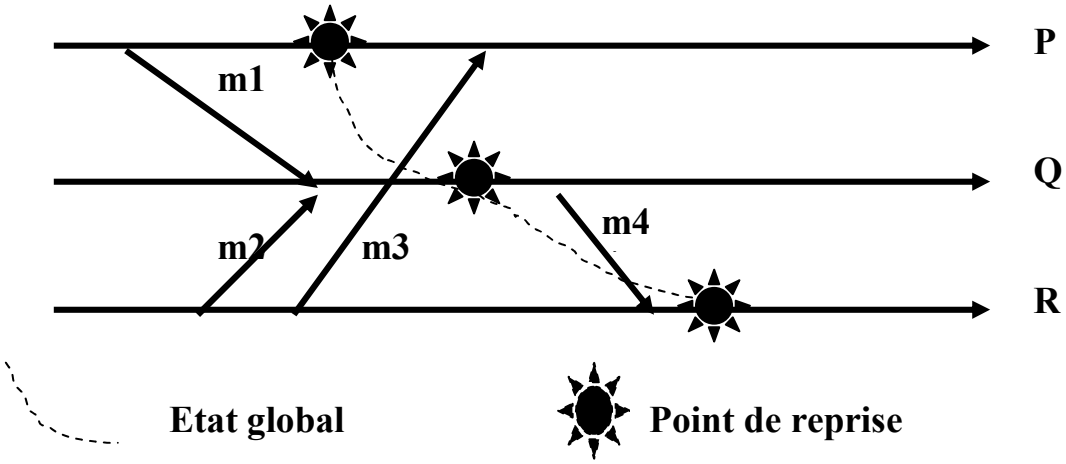


Figure 5 : Etat global incohérent.

Définition 04 : Un **message orphelin** par rapport à un état global est un message qui a été envoyé après un enregistrement d'état local appartenant à cet état global et reçu avant un enregistrement d'état local appartenant à cet état global.

Définition 05 : Un état global est dit **cohérent** s'il n'existe pas de message orphelin par rapport à cet état. En d'autres termes, si un état local d'un processus P a enregistré la réception d'un message provenant du processus Q, alors il faut que l'état local sur Q appartenant au même état global, ait enregistré l'émission de ce message. On note qu'un message peut avoir été envoyé mais non reçu (voire figure 6) comme le cas du message m3 dont l'émission par R est enregistrée mais pas la réception.

L'état global reste cohérent, au sens de Chandy et Lamport [30]. Le message m3 qui sera perdu en cas de reprise du système est appelé un **message en transit**.

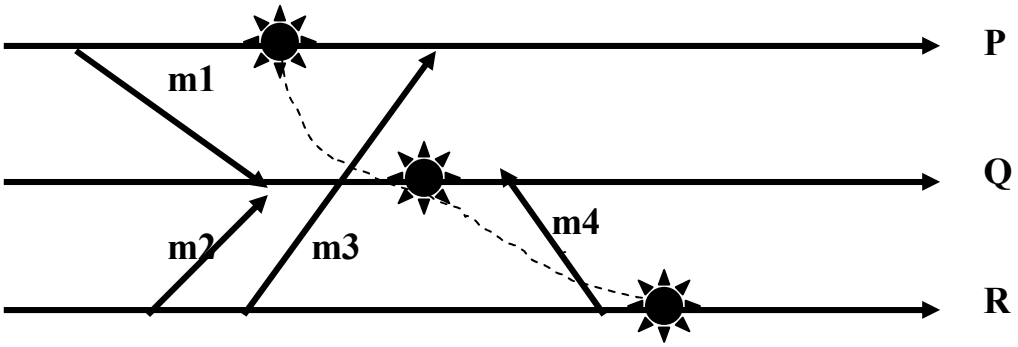


Figure 6 : Etat global cohérent.

Définition 06 : Un **message en transit** par rapport à un état global est un message qui a été envoyé avant un enregistrement d'état local appartenant à cet état global et reçu après un enregistrement d'état local appartenant à cet état global. Nous verrons par la suite une notion de cohérence plus stricte qui interdit aussi ce type de message.

4.5.2 La reprise dans un système réparti

Définition 07 : Un **point de reprise** (checkpoint) noté Rp est l'instant où l'activité d'un processus est interrompue momentanément pour sauvegarder son état local courant afin de pouvoir le restaurer ultérieurement en cas de panne. On peut prendre plus d'un Rp par processus, l'intervalle d'activité entre deux Rp successifs est appelé Région de reprise notée RR. Nous avons donc :

- **Rp actif :** est le point de reprise le plus récemment établi.
- **Rp de relance** : noté RpR, est le Rp choisi par un processeur pour relancer l'exécution. d'un processus impliqué dans une opération de reprise.
- **Rp dominant :** soient deux Rp, Rpi et Rpj appartenant au même processus. Rpi est dit dominant de Rpj, si Rpi est crée avant Rpj (on note Rpi< Rpj).

Définition 08 : Une **opération de reprise ou de recouvrement** est l'action entreprise par un mécanisme approprié, pour recouvrer un système perturbé par une panne, en lui substituant un état correct lui permettant d'assurer la continuité de ses services.

4.5.3 La propagation de la reprise

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

a) Cas des processus indépendants

Dans le cas ou le processus n'a eu une communication avec aucun des processus du système, la reprise est purement locale au processus défaillant [26].

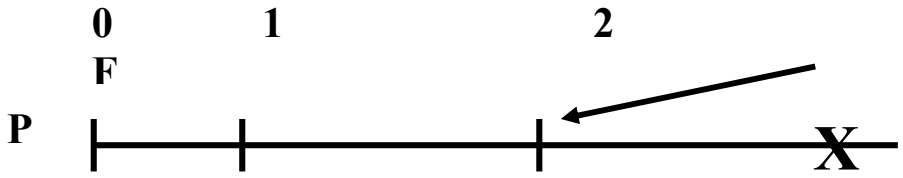


Figure 7 : Reprise de processus indépendants.

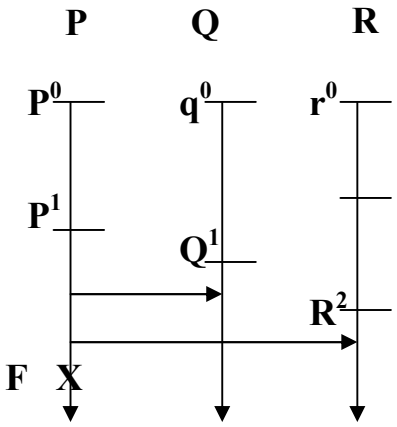
.

La figure 7, illustre l'évolution d'exécution de P dans le temps, P prend des points de reprise aux instants 0, 1 et 2.à l'instant F le processus p détecte une erreur, alors p exécute une reprise à partir de l'état 2.

b) Cas de processus communicants

La communication entre processus implique l'envoi ou la réception de messages entre eux, si P envoie un message m à Q, Q devient **dépendant** de P. Contrairement au premier cas, pour doter un système de processus communicants, d'un mécanisme de reprise, il faut prendre en compte le problème très complexe de la **propagation** d'erreur relative aux dépendances directes ou transitives [22], [26], [27]. [28], [29]. Cette propagation peut conduire à la contamination de tous les processus communicants au processus défaillant et comme il peut y avoir une propagation de l'erreur, il serait normal d'avoir une propagation de reprise.

Figure 8 : Propagation de reprise.



La figure 8, montre l'évolution de trois processus communicants P, Q et R. A l'instant F, le processus P détecte une erreur, alors il lance une opération de reprise à partir du point P^1. Comme Q a reçu des donnés de P, ces données sont potentiellement erronées, alors Q doit aussi exécuter une reprise à partir de Q^1. De même pour R doit restaurer l'état R^2 et effectuer une reprise à ce point.

Remarque :
La propagation interprocessus de la reprise a deux causes :

1. Manque des messages indispensables (à la ré exécution et donc à la reprise, c à d si le processus, dans son exécution normale utilise des messages qui sont générés par d'autres processus, alors lors de sa ré exécution, il a besoin de ces messages. Il faut donc propager la reprise vers les producteurs de ces messages pour les régénérer, nous parlerons alors de rétro propagation.

2. Réception des messages erronés, c- à- d si un processus P émet des messages a un autre processus Q et puis il détecte une erreur, alors Q risque d'être potentiellement défaillant il faut, donc le reprendre, nous dirons que nous avons une post propagation .
De ce fait chaque processus devra avoir des renseignements concernant les processus avec les quels il a pu recevoir ou transmettre des messages.

Définition 09 : La **Liste des dépendants** (notée LD) d'un processus P, est l'ensemble des processus aux quels il a envoyé des messages.

Définition 10 : La **Liste des propagateurs** (notée LP) d'un processus P, est l'ensemble des processus de la part desquels, il a reçu des messages.

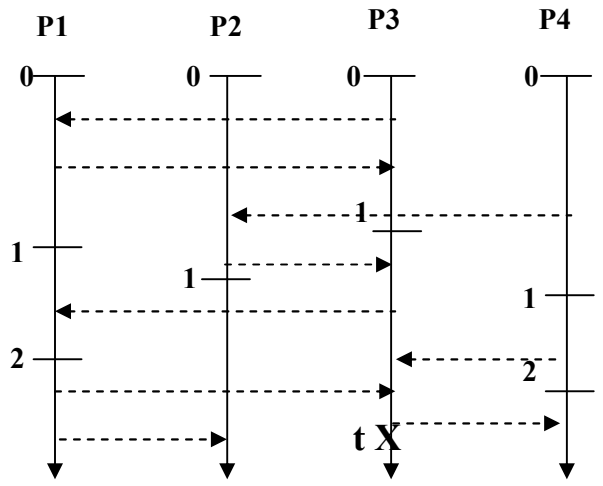


Figure 9 : Propagateurs et dépendants.

La figure 9, illustre l'évolution temporaire de quatre processus Pl, P2, P3 et P4. Pour le $P3_{.1}$, nous avons :

- Les dépendants directs sont : $P1_1$ et $P4_2$.
- *Les* dépendants indirects sont : $P1_1$, $P4_2$ et $P2_1$.
- Les propagateurs directs sont : P_{20}, $P4_1$ et PI_2.
- Les propagateurs indirects sont : $P2_0$, $P4_1$, PL_2 et $P4_0$.
- La liste des dépendants de P3 à l'instant t est : LD= {Pl_0, Pl1, P42}.
- La liste des propagateurs de P3 à l'instant t est : LP= {Pl_0, $P2_0$, $P4_1$, Pl_2}.

Les listes de dépendance et de propagateurs sont utilisées pour générer les graphes de dépendances et de reprises.

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

4.5.4 La détermination d'une ligne de recouvrement

Définition 11 : Une **ligne de recouvrement** est l'état global cohérent le plus récent. Cette ligne de recouvrement doit toujours être assez récente, de façon à minimiser le retour arrière en cas de reprise. C'est à dire le nombre de checkpoints entre le dernier pris et celui utilisé pour la reprise. Cependant, cette ligne n'est pas toujours créée pendant l'exécution, mais recherchée dans l'ensemble des états globaux formés, après qu'une panne soit survenue dans le système. Cette dernière méthode est fortement sujette à ce qu'on appelle **l'effet domino** [29].

Définition 12 : L**'effet domino** est caractérisé par une cascade de retours arrière lors de la reprise du système après une panne. Considérons la figure 10. Lorsque le processus Q tombe en panne, au point x, le système va tenter de reprendre depuis l'état global le plus récent, c'est à dire celui formé par les points de reprise CP3, CQ3 et CR3. A cause du message orphelin m, cet état ne sera pas cohérent. On va donc faire un retour arrière sur P, jusqu'au point CP2. Mais l'état global ainsi formé n'est toujours pas cohérent. De la même manière, le système va devoir reculer jusqu'à la dernière ligne (formée de CP1, CQ1et CR1) et donc perdre une grande quantité de travail.

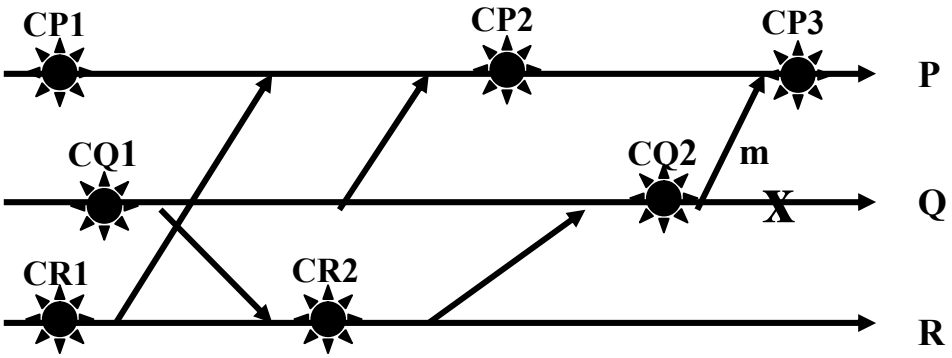


Figure 10 : Effet Domino.

4.5.5 Caractérisation de la cohérence d'un état global

Nous donnons ici certaines conditions pour qu'un état global soit cohérent au sens de [30].

Définition 13 : Soient P et Q deux processus. Soient a et b deux évènements. On dit que l'événement a **précède** l'événement b (ou b dépend causalement de a) si et seulement si :

- a et b surviennent sur le même processus, et a survient avant b.
- a est l'envoi d'un message par un processus P, et que b est la réception de ce message par un processus Q.
- si a précède l'évènement e et e précède b (transitivité).

Cette relation de causalité entre évènements(est basée sur la relation happened-before de Lamport [31]), permet de définir une condition nécessaire pour qu'un état global soit cohérent.

Propriété : Si deux points de reprise C1 et C2, appartenant à un état global cohérent SG1, alors il n'existe pas de dépendance causale entre ces deux points. En effet, s'il existe une dépendance causale entre deux checkpoints du même état global, alors il existe dans cet état global, deux checkpoints CPi sur le processus P et CQj sur le processus Q qui sont en dépendance causale directe, c'est à dire qu'un message a été envoyé après la prise de CPi, et reçu avant la prise de CQj. Ce message est donc un message orphelin, et l'état global considéré n'est pas cohérent. De cette manière, on peut vérifier que chaque communication directe (point à point) n'entraîne pas une dépendance causale entre un checkpoint de l'émetteur et un checkpoint du récepteur, et

prendre les mesures nécessaires si c'est le cas, comme par exemple prendre un checkpoint avant l'émission ou la réception. Cependant, cette règle n'est utilisable que si on considère les communications directes entre processus : il faut qu'elle soit vraie deux à deux pour tous les processus ayant eu une communication directe. Il existe cependant une autre méthode pour caractériser la cohérence d'un état global, qui considère aussi les relations indirectes entre processus. Elle a été proposée par Netzer et Xu dans [32] et est basée sur la notion de chemin "zig-zag", une généralisation de la relation de causalité de Lamport.

Comme on peut le voir dans la figure 11, CP1 n'est pas en relation causale avec CR1, pourtant ils ne peuvent pas appartenir tous deux à un même état global cohérent, puisque soit le message m1 dans l'état global [CP1, CQ2, CR1], soit le message m2 dans l'état global [CP1, CQ1, CR1] serait orphelin.
Cette relation qui existe entre CPi et CR j se nomme chemin "zig-zag". C'est une extension de la relation de causalité de Lamport et peut se définir comme :

Définition 14 : Il existe un chemin "zig-zag" entre deux points de reprise CP et CQ si et seulement si, il existe des messages m1, m2, ..., mn tels que :

- m_1 est émis par P après la prise de CP.
- si m_k $(1 \leq k \leq n)$ est reçu par un processus R, alors le message m_{k+1} est envoyé par R dans le même intervalle de points de reprise (ou dans un intervalle suivant). m_{k+1} peu être envoyé aussi bien avant que après la réception de mk.
- mn est reçu par Q avant la prise de CQ.

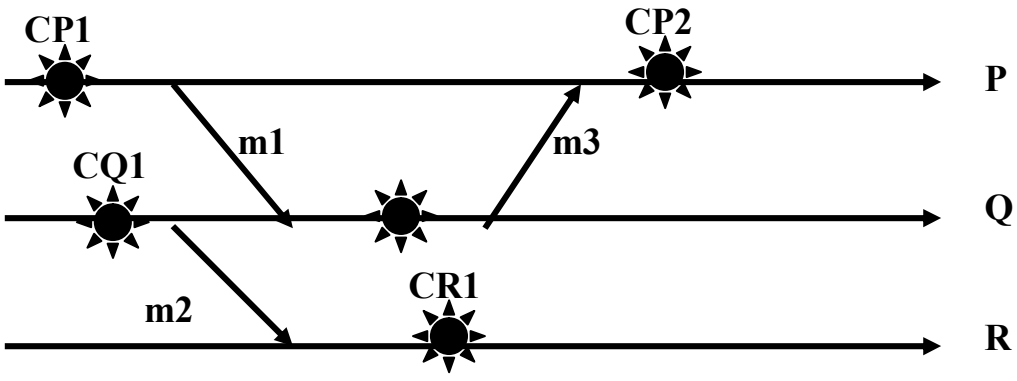


Figure 11 : Chemin en ZIG-ZAG.

Lorsqu'il existe un chemin "zig-zag" entre un point de reprise et lui-même, on parle alors d'un cycle "zig-zag". De fait, un point de reprise pris dans un cycle "zig-zag" ne peut bien sûr appartenir à aucun état global cohérent, et est donc inutile. Par exemple, dans la figure 9, le point CQ2 est pris dans un cycle "zig-zag" à cause des messages m1 et m3.

On utilise ces notions de chemin et de cycle "zig-zag" pour déterminer les lignes de recouvrement, ou pour savoir s'il est utile, à un instant donné, de prendre un point de reprise [33], [34].

Définition 15 : La cohérence au sens de [30] n'interdit pas les messages en transit : nous parlerons alors de **cohérence simple**, et l'application doit pouvoir supporter la perte de message. Ce sont cependant des hypothèses très contraignantes, et c'est pour cela que l'on introduit la notion de **cohérence forte**.

La cohérence forte étend la cohérence simple, avec la condition supplémentaire qu'il n'existe pas de messages en transit dans l'état global considéré. Hélary, Netzer et Raynal montrent dans [35] que si l'absence de chemin "zig-zag" entre les points de reprise assure l'absence de message orphelins, alors l'absence de chemin "zig-zag" inverse assure l'absence de messages en transit. Un chemin "zig-zag" inverse est un chemin "zig-zag" que l'on trouverait dans l'exécution si on inversait tous les messages de sorte que les émetteurs deviennent les récepteurs, et réciproquement. En effet, on peut constater qu'un message en transit est le cas inverse, ou dual, d'un message orphelin : l'un n'a plus d'émetteur et l'autre n'a plus de récepteur.

Parmi ces définitions, on décèle explicitement des problèmes dans le recouvrement arrière mais aussi implicitement, les solutions à ces problèmes. Donc, lors de la mise en ouvre d'un système tolérant aux fautes basé sur le recouvrement arrière nous devons avoir comme objectif primaire, l'obtention d'un état global cohérent vérifiant les conditions citées tout au long de ce chapitre.

Chapitre 5

La mise en œuvre d'une stratégie de recouvrement arrière

5.1 Objectifs attendus d'une stratégie de recouvrement arrière

L'objectif explicite du recouvrement arrière, est de recouvrir un état erroné du système par un autre état capturé et sauvegardé sur un support stable avant l'occurrence de la faute.

Une stratégie de recouvrement arrière, est choisie selon des critères de base. Ces critères, évaluent le degré de transparence de l'exécution du mécanisme de recouvrement, par rapport à l'exécution de l'application répartie.

Nous identifions quatre objectifs du recouvrement arrière : le degré de tolérance aux fautes, les surcoûts en consommation de ressources pendant l'exécution, l'inhibition pendant l'exécution et la quantité de travail à défaire ou à refaire.

5.1.1 Le degré de tolérance aux fautes

Le degré de tolérance aux fautes est le nombre maximum de fautes simultanées tolérées. Les fautes comprennent aussi bien les fautes de l'application que celles des mécanismes de recouvrement arrière. C'est l'objectif le plus important. Une politique est efficace si les mécanismes choisis tolèrent la faute globale du système réparti sans effet domino.

5.1.2 Les surcoûts pendant l'exécution

Les surcoûts pendant l'exécution sont de trois ordres [36] :

5.1.2.1 La charge du réseau

Toute application telle qu'elle soit, génère un trafic de messages sur le réseau de communication.

5.1.2.2 Le stockage

La procédure de recouvrement arrière commence par une capture d'un état de l'application et, est automatiquement suivie de la sauvegarde des informations en mémoire stable.

5.1.2.3 Temps d'exécution

Cette perte de temps est due à la fois aux accès à la mémoire stable, pour le stockage, et au mécanisme de capture de l'état lui-même.

Donc, l'objectif serait, de capturer des états de l'application, qui soient utiles, de taille la plus minimale possible et échelonner les sauvegardes et d'espacer leur fréquence.

5.1.3 L'inhibition pendant l'exécution

Toutes les applications possèdent un temps d'exécution propre. Les stratégies de recouvrement arrière, seront d'autant plus performantes si elles s'interfèrent moins avec l'application. En d'autres termes, l'objectif est qu'elles utilisent les ressources aux moments ou elles sont disponibles.

Par ailleurs, les mécanismes respectant la cohérence forte d'un système, inhibent l'exécution de l'application.

5.1.4 La quantité de travail à défaire ou à refaire

La performance des stratégies de recouvrement arrière peut également être évaluée en terme de quantité de travail à défaire ou à refaire. Elle se décline selon deux métriques :

5.1.4.1 La distance de recouvrement arrière

La distance de recouvrement arrière mesure l'effet domino ou quantité de travail à défaire.

5.1.4.2 Le nombre de processus impliqués par la stratégie

Le nombre de processus devant se ré exécuter (c'est le premier paramètre de la quantité de travail à refaire) varie selon les applications et les politiques de recouvrement arrière.

5.2 Mécanismes de base pour la mise en œuvre d'une stratégie de recouvrement arrière

Pour mettre en œuvre une stratégie de recouvrement arrière, il faut en dégager les ingrédients essentiaux. Alors, en l'absence de fautes nous devons capturer un ou plusieurs états, que doit il contenir ? Ensuite nous devons nous assurer de leur stockage dans un endroit stable : pourquoi pas la mémoire vive ? …s'il y'a une faute qui surgit, nous devons la déceler, et essayer de la tolérer.

Toutes ces questions serviront de fondement pour cette mise en œuvre.

5.2.1 La détection des défaillances

C'est l'événement déclencheur du recouvrement, de ce fait la première étape d'une stratégie de recouvrement arrière est la détection de la défaillance d'un processus. Cette procédure est expliquée dans le précédent, section 4.3.2.1.

5.2.2 la sauvegarde d'état du processus

Un programme informatique qui s'arrête perd toutes les informations relatives à son état. La mémoire de travail physique des ordinateurs, la « mémoire vive », étant volatile.

5.2.2.1 Que faut il sauvegarder

Si nous devons sauvegarder l'état d'un processus, nous devons nous assurer que cet état doit être suffisamment complet pour qu'un arrêt du processus suivi d'un recouvrement préserve toutes les caractéristiques du processus et permettent notamment le rétablissement des communications et des entrées/sorties.

Il faut donc conserver le tas, la pile, le code et l'état des registres dans le système d'exploitation. Il est également nécessaire de sauvegarder le répertoire courant, les comptes à rebours, l'identité de l'utilisateur, celle du processus, l'arborescence du processus, les librairies partagées liées à l'application, les fichiers ouverts surtout en écriture et puisque nous nous trouvons dans un environnement réparti, nous devons parfois sauvegardé l'histoire répartie du processus (voire chapitre 4 section 4.5.1).

5.2.2.2 Où faut il sauvegarder

La mémorisation des états, doit se faire sur un support qui résiste aux défaillances des machines (sûreté) et n'altère pas l'information conservée (sécurité). Aussi, ce support doit être accessible en lecture et en écriture.
En fonction du nombre de fautes à tolérer, des performances et des moyens disponibles, plusieurs solutions sont possibles pour réaliser une mémoire stable.

La solution la plus simple, est l'utilisation de la mémoire vive d'un serveur considéré comme fiable [37]. Le passage à l'échelle est rendu difficile et il est parfois nécessaire de dupliquer le serveur, le système perdant alors une bonne partie de sa simplicité. Une autre solution est la sauvegarde du point de reprise dans la mémoire vive d'un nœud distant Ceci ne permet de résister qu'à une seule faute mais évite la saturation d'un serveur et augmente considérablement l'utilisation mémoire. Pour éviter ce surplus

d'utilisation mémoire, une variante consiste à sauvegarder le point de reprise sur un disque local d'un nœud distant. Le défaut est alors le coût en latence dû aux accès en écriture sur disque. De nos jour, les méthodes utilisées utilise la philosophie de dupliquer la sauvegarde sur un deux sites distants.

5.2.2.3 Qui doit faire la sauvegarde

On peut lancer la sauvegarde de trois manières : par l'application elle-même par l'insertion, au sein de l'application, des codes permettant la réalisation de la sauvegarde, par une librairie liée à l'application qui initiera les sauvegarde .ou encore par le système d'exploitation. La dernière solution, est l'intégration dans le système d'exploitation de la génération des points de reprise. Le système d'exploitation peut fournir une primitive d'initiation des points de reprise ou les initier de manière transparente. Dans ce dernier cas, aucune modification des applications ou librairies

5.2.2.4 Libérer l'espace de stockage (le ramasse miette)

A un certain moment, l'état sauvegardé perdra sa cohérence, il ne sera plus d'actualité vu la progression de l'application. Ces mises à jour au niveau de l'exécution du processus imposeront une nouvelle sauvegarde, donc une nouvelle occupation d'espace en mémoire. Faut-il mettre à jour ces sauvegardes, les supprimer parce qu'elles sont devenues inutiles ou tout simplement les garder pour usage ultérieur. La libération de l'espace occupé par les sauvegardes d'état, est décidée par protocole de recouvrement arrière adopté.

5.2.3 Classes de mécanismes établissant la sauvegarde

Les mécanismes de recouvrement arrière se basent principalement sur :

5.2.3.1 Mécanismes à base de constitution de points de reprise (Checkpointing)

Qui encapsule les classes :

Classe1 : Constitution d'un état global cohérent : construction explicitement synchronisée d'un état global cohérent.

Classe2 : Constitution d'une barrière de synchronisation par arrêt global de l'application avec vidage des canaux de communication pour qu'il n'y ait pas de message en transit et pour que les points de reprise soient tous utiles.

Classe3: Constitution de points de reprise non synchronisée, ne cherchant pas à former un état global cohérent.

Classe4: Constitution implicitement synchronisée, déclenchée par le calcul de prédicats sur les estampilles des messages échangés.

5.2.3.2 Mécanismes à base de journalisation des messages (Message logging)

Qui encapsule les classes :

Classe1 : Journalisation pessimiste : journalisation régulière, en mémoire stable de tous les messages.

Classe2 : Journalisation optimiste : journalisation facultative en mémoire stable de tous les messages.

Classe3 : Journalisation causale : journalisation facultative, en mémoire stable de tous les messages, décidée par les dépendance causales entre processus.

Et on peut avoir des mécanismes à base de la concaténation de la constitution des points de reprise et de la journalisation.

5.2.4 Recouvrement arrière sur l'erreur

Un site défaillant peut entraîner la défaillance d'une partie si ce n'est de tout le système (propagation de l'erreur chapitre1, section 1.7.2).
Une propagation de l'erreur est automatiquement suivie d'une propagation de reprise, dans ce cas nous devons déterminer le point recouvrable de l'application répartie, constituée des points recouvrables de chaque processus impliqué dans le recouvrement.

Dans ce chapitre, nous avons évoqué les principes de base pour la mise en œuvre des stratégies de recouvrement. Passons maintenant l'implémentation de Protocole de recouvrement arrière.

Chapitre 6

Protocole de recouvrement arrière dédiés à l'environnement réparti fixe.

6.1 Hypothèses

Dans tout ce qui va suivre, les hypothèses suivantes sont prises en considération :

- ✓ le canal adopte une stratégie FIFO (First In, First Out) pour véhiculer les messages.
- ✓ Le canal est fiable.
- ✓ Le modèle de faute considéré est le modèle fail-stop c-à-d que le processus fonctionne normalement, soit il est défaillant et s'arrête complètement de fonctionner.

6.2 Présentation des Protocole de recouvrement arrière

Comme déjà dit dans le chapitre précédent, on peut distinguer deux approches différentes dans la tolérance aux pannes par recouvrement arrière: L'une basées sur les points de reprise, l'autre sur la journalisation.

6.2.1 Protocoles de recouvrement arrière basés sur l'établissement de points de reprise

Pendant l'exécution, le mécanisme d'établissement de points de reprise (Checkpointing), enregistre en mémoire stable les états locaux des processus. Chacun de ces algorithmes appartiennent à une classe bien déterminée (voire chapitre 5 section 5.2.3.1) et ceci selon la stratégie suivie. On distingue, trois stratégies principales :

6.2.1.1 Points de reprise non coordonnés (uncoordinated checkpointing)

Aussi appelée points de reprise indépendants ou asynchrones. Cette méthode consiste à faire prendre aux différents processus du système des points de reprise de manière totalement indépendante. La cohérence des états globaux formés n'est donc pas assurée et c'est lors d'un recouvrement, que le protocole recherche une ligne de recouvrement parmi tous les états globaux formés. Cette recherche s'effectue par la création d'un graphe de dépendance entres les points de reprise locaux. Beaucoup de travaux à cet effet ont vu le jour, [38], [39], [40], [41], [42], etc. L'idée est basée sur la dépendance causale. Cette méthode est donc fortement sujette à l'effet domino, puisque rien n'assure lors de la prise d'un checkpoint que celui-ci pourra faire partie d'un état global cohérent. Elle vise à maximiser les performances en fonctionnement normal.

L'objectif est de minimiser le surcoût lié à la sauvegarde des points de reprise lors d'une exécution sans fautes.

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

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

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

Cependant, cette technique possède différents inconvénients. Premièrement, le calcul de la ligne de recouvrement peut s'avérer coûteux, surtout si le nombre de processus est grand et les communications importantes. De plus, il y a un risque d'effet domino qui a pour conséquence de ramener l'application dans son état initial, perdant ainsi tout le calcul déjà effectué. Ainsi, compte tenu des dépendances causales entre processus et des points de reprise sauvegardés, la ligne de recouvrement est ici l'état initial. D'autre part, pour chaque processus, il est nécessaire de conserver plusieurs points de reprise, ce qui entraîne un surplus d'espace de stockage.

6.2.1.2 Points de reprise coordonnés (coordinated checkpointing)

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

Cette approche repose sur une coordination et la synchronisation globale de tous les processus de l'application [30], [27], [43], [44].

La coordination peut être :

- **Centralisée :** Il y'a l'existence d'un site coordinateur, c'est lui qui coordonne l'algorithme d'établissement de points de reprise.

- **Répartie :** Il n'y a pas de coordinateur, la coordination est implicite faite via des horloges.

La synchronisation peut être :

- **bloquante :** la cohérence de l'état global est assurée par le fait que tous les processus sont stoppés lors de la prise de cet état global.

- **non bloquante :** On essaie de ne pas stopper le travail en cours lors de la prise d'un checkpoint par le processus.

Pour chacune de ces propriétés, est implémentée, une variante du protocole d'établissement de points de reprise.

6.2.1.2.1 La coordination centralisée bloquante (Sync-And-Stop SNS)

Dans cette approche, le coordinateur (un processus de l'application, désigné dynamiquement ou statiquement), sauvegarde son point de reprise et diffuse la requête CKPT, demandant à tous les autres processus de l'application d'établir leur point de reprise. Quand un processus reçoit le message, il stoppe (bloque) son exécution, vide ses canaux de communication, prends un point de reprise provisoire, et envoie un message d'acquittement au coordinateur [45], [46]. Une fois que le coordinateur ait reçu l'ensemble des acquittements, celui-ci diffuse un message de validation du point de reprise. Chaque processus remplace alors, de manière atomique, son ancien point de reprise provisoire par le point de reprise permanent. Lors de la reprise, il suffit de restaurer l'ensemble des processus de l'application à partir de leur point de reprise respectif. Cette technique assure que l'ensemble des points de reprise forme un état global cohérent et évite ainsi l'effet domino. De plus, la reprise est très simple. Un autre avantage de cette technique est que l'espace de stockage nécessaire pour conserver les points de reprise est minimisé puisqu'il n'est nécessaire que d'en conserver un seul par processus.

Le principal inconvénient de cette stratégie est la latence importante qu'elle implique lors de la sauvegarde du point de reprise, étant donné qu'elle nécessite une coordination globale de tous les processus.

Deux problèmes se posent donc ici, la perte de performance y compris en l'absence de fautes, et la gestion du passage à l'échelle. En effet, plus le nombre de processus augmente, plus la latence risque d'être importante, chaque processus devant attendre la fin de l'opération avant de pouvoir reprendre son exécution normale.

6.2.1.2.2 La coordination centralisée non bloquante

Les processus peuvent entamer leur travail courant normalement et continuer à recevoir et envoyer des messages.

Pour améliorer le protocole de coordination centralisée bloquante [47], [48], on force chaque processus à précéder chaque message de l'application, envoyé immédiatement après l'établissement d'un point de reprise, par une requête CKPT. Donc les messages de l'application seront reçus et traités selon deux cas :

* Soit le message reçu, arrive tout seul, ce qui veut dire que c'est un message normal que nous devons seulement exécuter.

* Soit le processus a reçu une requête CKPT avant la réception de ce dernier. La traitement de ce message sera différé jusqu'après l'établissement d'un point de reprise provisoire et la retransmission de la requête CKPT vers les autres processus avant de reprendre sa tâche ordinaire.

Cette technique assure qu'aucun message de l'application ne soit traité entre le moment de l'émission de la requête CKPT et la prise en compte de cette décision en effectuant un point de reprise provisoire.

6.2.1.2.3 La coordination répartie bloquante (Checkpointing based time)

Dans cette approche, La création de points de reprise est coordonnée indirectement par des horloges synchronisées et il y'a pas besoin d'élire un initiateur. On remarque que la notion de coordinateur n'a pas totalement disparue, mais est incluse implicitement dans le protocole. La synchronisation est garantie par la simulation de la notion de temps physique [49], [50] en introduisant des horloges virtuelles. Ces dernières facilitent la coordination.

Chaque processus s'initient lui-même prend un point de reprise provisoire et attend un temps égal à la somme des temps maximal de déviation entre les horloges virtuelles se trouvant dans le système, et le temps maximal entre la détections de deux pannes sur ce système, pour rendre ce point provisoire permanent. Aucun échange de messages n'est permis pendant cette opération.

6.2.1.2.4 La coordination répartie non bloquante

Pour éliminer la contrainte de blocage [51], il suffit tout simplement d'indexer les points de reprise et tous les messages circulant dans l'application avec des numéros séquentiels et de rajouter des informations supplémentaires dans les messages.

6.2.1.3 Points de reprise induits par les communications (checkpointing induced communication)

La technique des points de reprise induits par les communications combine les protocoles d'établissement de points de reprise coordonnés et indépendants [52], [53] Dans chaque message de l'application, des informations du protocole sont encapsulées. L'idée est d'initier des points de reprise en fonction des informations contenues dans ces messages échangés, donc la synchronisation se fait ici de manière "paresseuse". Chaque processus prend régulièrement des points de reprise de manière indépendante, mais parfois il pourrait s'avérer utile de prendre des points de reprise additionnels en forçant le processus à le faire. L'algorithme de décision assure (ou maximise la probabilité) que le point pris fasse partie d'un état global cohérent, et donc qu'il existe une ligne de recouvrement toujours assez récente.

On distingue deux familles de protocole :

6.2.1.3.1 Les protocoles model-based

Ces protocoles sont généralement basés sur les notions de détection de chemins et de cycles "zigzag" [32], et le forçage d'un point de reprise additionnel pour éviter l'incohérence les états globaux.

Les communications et les établissement de points de reprise, doivent respecter un certain motif : par exemple, si tout envoi et toute réception de message est précédé d'un point de reprise, alors tous les points au moins, appartiennent à un état global cohérent [59], [55], et le dernier point pris fait toujours partie de la ligne de recouvrement.

6.2.1.3.2 Les protocoles index-based

Les checkpoints sont indexés [42], [52], [56], chaque message porte l'index du dernier point de l'émetteur. Les indexes sont insérés dans les messages d'application, pour aider les récepteurs à décider quand ils doivent effectuer un point de reprise forcé. Sur réception d'un message indiquant un index supérieur au sien, le récepteur doit prendre un point de reprise avant la prise en compte du message.

Aucun message spécifique de coordination n'est envoyé, mais les points de reprise sont réalisés lors d'évènements particuliers. Les techniques de sauvegarde de checkpoints induits par les communications évitent l'effet domino tout en ne nécessitant pas la coordination de tous les processus de l'application. La difficulté tient ici dans la mise en oeuvre de la méthode de décision quant à la nécessité de créer un checkpoint. De plus, si les messages de l'application sont très petits, l'encapsulation dans chaque message des informations, entraîne un surcoût qui peut être important.

6.2.2 Comparaison entre les trois techniques basées sur les points de reprise

La comparaison entre les différentes stratégies se fait en considérant les aspects suivants :

- le degré de tolérance aux fautes, c'est à dire le nombre de fautes simultanées tolérées, pendant l'exécution normale ou pendant la reprise.
- le surcoût pendant l'exécution, en terme de CPU et de réseau.
- le nombre de retours arrière moyen.
- le surcoût de stockage, et la facilité du ramassage des checkpoints stockés devenus inutiles.

On ne parlera pas de l'inhibition citée dans le chapitre précédent parce qu'elle est explicite dans chaque méthode.

6.2.2.1 Degré de tolérance aux fautes

Les protocoles synchronisés sont particulièrement sensibles aux pannes des noeuds coordinateurs de la création d'une ligne de recouvrement. Ces noeuds contrôlent le début et la terminaison du protocole de constitution de la ligne de reprise, et leur panne doit être gérée comme un cas particulier dans le protocole, si on veut pouvoir les

tolérer. Dans les deux approches, la panne simultanée de plusieurs processus est supportée (en dehors de la création de la ligne de recouvrement) : il existe un état global antérieur à partir duquel tous les processus du système doivent de toutes façons reprendre, en cas de panne d'un ou de plusieurs processus.

6.2.2.2 Surcoût pendant l'exécution

L'utilisation de messages additionnels rend les protocoles synchronisés gourmands en ressources durant une exécution normale, que ce soit en terme de réseau ou en terme de temps d'exécution, particulièrement pour les protocoles bloquants qui stoppent les processus au cours de la synchronisation. C'est principalement ce surcoût qui rend les protocoles synchronisés en général incapables de passer à l'échelle.

L'approche induite par messages ne rajoute pas de messages additionnels durant l'exécution : le surcoût principal se trouve dans le traitement des messages applicatifs. En effet, il faut filtrer tout message entrant pour récupérer les informations utiles à la reprise. Il y a aussi un surcoût au traitement lié à la taille des messages qui peut être très différent d'un protocole à l'autre, allant de sans surcoût à un vecteur d'entiers de la taille du nombre de processus dans le système. Bien sûr, le coût de la prise d'un checkpoint par un processus, est le même dans toutes les approches.

6.2.2.3 Nombre de retours arrière moyen

Les protocoles synchronisés ont ici un avantage certain : tous les points de reprise sont utiles, et lorsqu'un processus prend un point de reprise, il est assuré de ne pas devoir retourner plus loin que ce point en cas de panne. Dans le cas des protocoles induits par messages, on trouve deux politiques : certains choisissent d'assurer que tout point de reprise pris est utile, c'est à dire qu'il fait partie d'un état global cohérent, alors que d'autres protocoles tentent de maximiser la probabilité que ce point soit utile, sans pour autant l'assurer. Cependant, pour les deux politiques (indépendants et induit par communication), le temps de création de la ligne de recouvrement n'est pas borné (contrairement à l'approche synchronisée) puisqu'il dépend de la propagation des messages applicatifs. La ligne de recouvrement peut s'étaler sur une grande période (le dernier point la constituant a été pris longtemps après le premier point) : on peut ainsi avoir une perte considérable de travail effectué pour certains processus.

6.2.2.4 Surcoût de stockage et ramassage

Sur ce point aussi, les protocoles synchronisés sont avantageux : puisque tout état global nouvellement créé est forcement un état cohérent, alors l'état global précédent peut être effacé de la mémoire. Le surcoût est donc minimal puisqu'il n'est nécessaire de conserver qu'un seul état global (plus bien sûr celui nouvellement construit), et le ramassage des états devenus inutiles devient trivial. Le ramassage des points de reprise dans le cas des protocoles induits par messages sera dépendant de la politique choisie, et nécessitera le plus souvent un processus chargé de déterminer les points devenus inutiles.

6.2.3 Protocoles de recouvrement arrière basés sur journalisation de messages

La tolérance aux fautes par journalisation des messages (ou message logging) combine le checkpointing avec la sauvegarde de l'histoire des messages qui circulent dans le système, pour pouvoir "rejouer" les communications en cas de panne d'un processus [57]. Ainsi, un processus enregistre sur un support stable tous les messages entrants ou sortants et, occasionnellement et de façon totalement indépendante, prend un point de reprise. En cas de panne de ce processus, il sera relancé depuis son point de reprise le plus récent, puis tous les messages enregistrés depuis la prise de ce point, seront rejoués. Nous pouvons avoir :

- **Une journalisation basée sur l'émission :** L'émetteur enregistre tous les messages sortants avant des les émettre.

- **Une journalisation basée sur la réception :** Cette fois c'est le récepteur qui doit enregistrer tous les messages entrants (reçus).

- **Une journalisation sur un serveur centralisé.**

Nous citons trois catégories de journalisation : la journalisation pessimiste, la journalisation optimiste et la journalisation causale.

6.2.3.1 Journalisation pessimiste

La journalisation pessimiste repose sur l'hypothèse suivante : une défaillance peut se produire juste après un envoi ou une réception de message. Tous les messages doivent être conservés au cas où il est nécessaire de les rejouer.

Le premier algorithme est proposé par Powell et Presotto [58]. L'ordre de ces messages est conservé, de manière à recréer lors de la reprise la même histoire

d'exécution de message donc, on associe à chaque message un identifiant et une estampille temporelle pour les archiver en mémoire stable dans un journal (log) dès la réception du message avant qu'il ne puisse être exploité par l'application [59]. Pour chaque message émis nous devons recevoir un accusé de réception confirmant sa réception celui-ci est conservé dans le journal.
Lors de la reprise, le processus défaillant est restauré à partir du dernier point de reprise et de son journal. Cette technique propose un retour arrière très limité et ne demande de conserver qu'un seul point de reprise. De plus, la restauration est simple à réaliser. Cependant, la nécessité de stocker l'acquittement de réception des messages de l'application en mémoire stable entraîne une latence importante et donc une perte de performance. D'autre part, le stockage des messages déjà émis peut s'avérer coûteux en espace.

6.2.3.2 Journalisation optimiste

La journalisation optimiste est fondée sur l'hypothèse que la journalisation se termine avant l'occurrence d'une défaillance [57], [60], [64]. À la différence de la journalisation pessimiste, le journal n'est pas écrit directement en mémoire stable mais est d'abord écrit en mémoire volatile pour être vidé périodiquement vers la mémoire stable. L'enregistrement des messages en transit dans le système se fait de façon asynchrone. Prenons le cas où, une panne survient avant que tous les messages soient enregistrés : les informations enregistrées en mémoire volatile du processus tombé en panne seront alors perdues. Dans ce cas, certains processus seront orphelins.

Le calcul de la ligne de recouvrement est plus compliqué que pour la journalisation pessimiste. En effet, comme le journal est partiellement écrit, certains évènements ne peuvent être rejoués et il peut être nécessaire de restaurer plusieurs processus de l'application alors qu'un seul est défaillant. Les processus doivent alors maintenir les informations sur les relations causales entre eux, pour pouvoir déterminer au moment de la reprise, quels sont les processus qui doivent reprendre pour que l'état global reste cohérent. Cette technique a pour avantage un faible surcoût en exécution sans faute. En effet, celle-ci n'est pas affectée par la latence produite à chaque message en journalisation pessimiste par l'écriture du journal en mémoire stable. Toutefois, les inconvénients de cette approche sont majeurs : la reprise est délicate et aucune garantie n'est fournie quant à la possibilité de retrouver un état global cohérent à partir du

dernier point de reprise. Il faut donc en conserver plusieurs, d'où un surcoût en espace disque.

6.2.3.3 Journalisation causale

La dernière catégorie est la journalisation causale qui essaie de combiner les avantages de la journalisation optimiste (faible surcoût en fonctionnement normal) et de la journalisation pessimiste.

Le journal est sauvegardé en mémoire volatile mais la décision de l'écrire en mémoire stable est prise à partir des dépendances causales entre les évènements. Le principe, est de diffuser l'histoire séquentielle d'un processus sur les noeuds où s'exécutent les autres processus de l'application. L'idée est que seuls les processus dépendant du message m reçu par un processus P, ont besoin de connaître son histoire. Donc, p transmet l'histoire de m en estampille des messages émis après la réception de m.

Le premier algorithme de journalisation causale a été proposé par Elnozahy et Zwaenepoel [64]. Les processus construisent un graphe de dépendances. Localement, pour un processus, ce graphe contient l'histoire répartie dont le processus dépend depuis la dernière sauvegarde en mémoire stable du graphe. Régulièrement et de façon optimiste, le processus enregistre en mémoire stable son graphe local non encore enregistré. Les estampilles des messages contiennent les graphes locaux non encore enregistrés en mémoire stable. A la réception d'un message, le processus ajoute le graphe contenu dans l'estampille à son graphe local [65], [66].

Cette stratégie, complexe à mettre en oeuvre, élimine le problème de la latence en journalisation pessimiste et permet un recouvrement limité au pire, au dernier point de reprise de chaque processus défaillant.

6.2.4 Analyse et comparaison des différentes approches par journalisation

Nous allons comparer les différentes stratégies de la journalisation selon les critères suivants :

- le nombre de processus devant reprendre en cas de panne.
- le surcoût en terme de temps d'exécution.
- la taille des messages.

6.2.4.1 Nombre de processus à reprendre

C'est là, le point fort de la journalisation pessimiste : lors de la panne d'un processus, elle assure que seul le processus fautif devra reprendre depuis son dernier état enregistré.

Les messages étant réémis, soit par un processus extérieur dans le cas basé sur la réception, soit par les autres processus de l'application dans le cas basé sur l'émission, les autres processus n'ont pas à recouvrer un état antérieur. Dans la journalisation optimiste, le nombre de processus qui reprennent après une panne peut être égal dans le pire des cas au nombre de processus du système. En fait, ce nombre dépend du maillage et du taux de communication de l'application : dans le cas d'une application fortement maillée ou les processus communiquent très souvent, la panne d'un processus entraînera certainement la reprise de tout le système.

6.2.4.2 Surcoût

La journalisation pessimiste a bien sur un coût très élevé à l'exécution. Tout message envoyé doit être journalisé de façon synchrone avant d'être reçu, ce qui en moyenne va doubler les temps de communication.

La journalisation optimiste a un surcoût à l'exécution "réglable". En effet, il est possible d'adapter la fréquence des enregistrements sur support stable en fonction de la fréquence des pannes et du surcoût maximal accepté durant une exécution sans panne. Bien sur, des enregistrements peu fréquents entraînent un temps de reprise en cas de panne important, puisque plus les intervalles entre enregistrements sont grands, plus les processus ont le temps de communiquer. Dans ce cas, un processus orphelin peut rendre orphelin un grand nombre d'autres processus.

6.2.4. 3 Taille des messages

La journalisation pessimiste n'ajoute aucune information supplémentaire sur les messages, et n'a donc aucun impact sur la taille des messages contrairement à la journalisation optimiste qui doit conserver assez d'information pour pouvoir identifier les processus qui ont communiqué avec le processus fautif lors d'une panne : la plupart des propositions utilisent des vecteurs dont la taille est égale au nombre de processus du système (dans le meilleur des cas, ces vecteurs sont des vecteurs d'entiers). Ils doivent

être ajoutés à tous les messages en transit dans le système, ce qui peut augmenter considérablement le temps d'émission mais aussi de traitement des messages.

Synthèse

Le tableau ci-dessous, présente un résumé des implications liées aux différentes stratégies de recouvrement arrière par points de reprise ou par journalisation. Toutes ces techniques différentes offrent donc des services différents. Cependant, de nombreux concepts sont partagés.

	Points de reprise			Journalisation		
	Non coordonnées	coordonnées	Induits par communication	Pessimiste	Optimiste	Causale
Effet domino	Possible	Non	Non	Non	Non	Non
N° CKP par processus	Plusieurs	1	Plusieurs	1	Plusieurs	1
Etendu de retour arrière	Illimité	Dernier CKP global	Plusieurs CKP possibles	Dernier CKP	Plusieurs CKP possibles	Dernier CKP
Protocole de reprise	Distribué	Distribué	Distribué	Local	Distribué	Distribué

Tableau 1 : Comparaison de différentes stratégies de recouvrement arrière

Nous terminons ce chapitre par une récapitulation sur les protocoles de recouvrement arrière (voir figure 12).

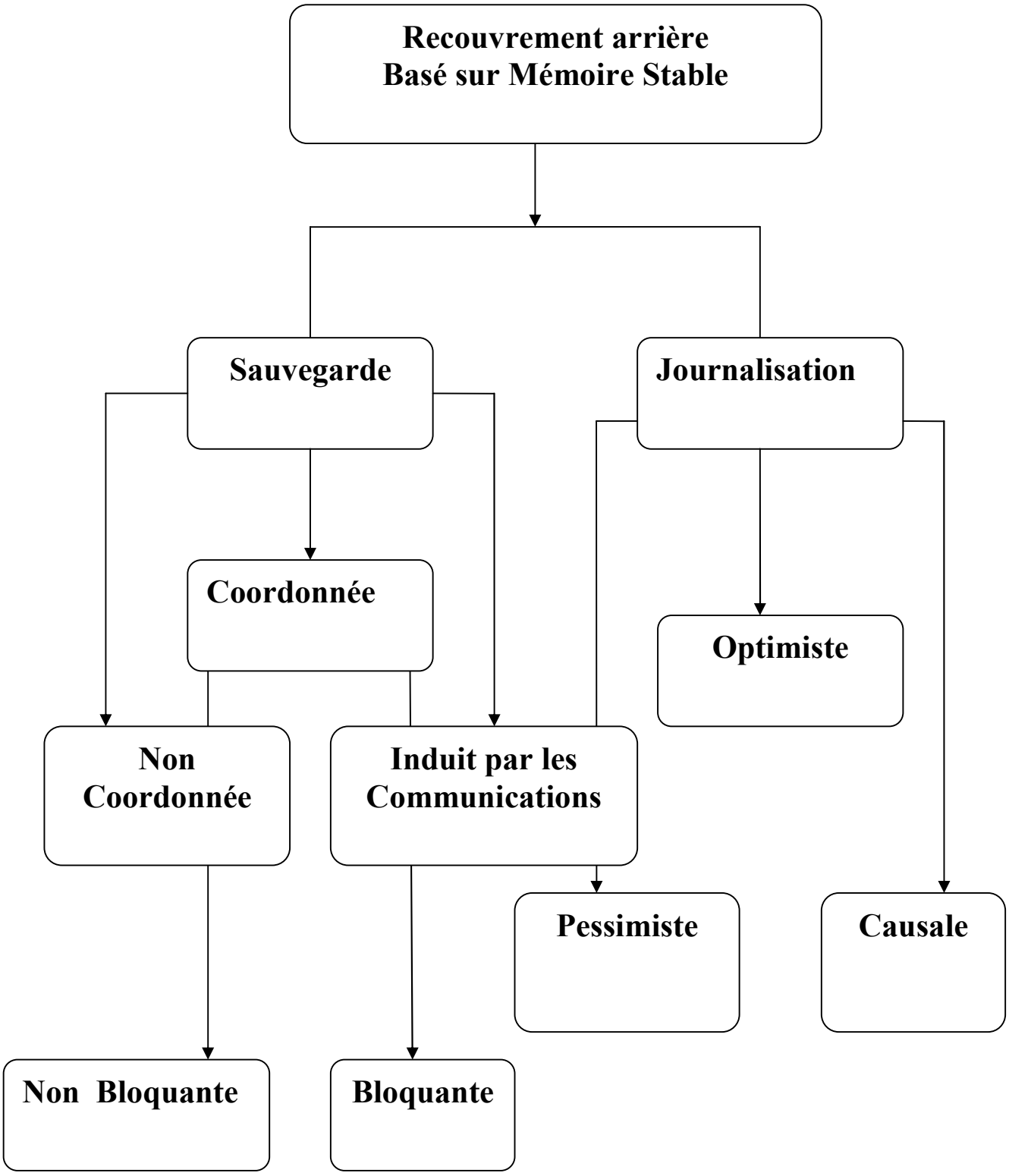


Figure 12 : Différents Protocoles de recouvrement arrière.

Troisième Partie

Etendue du Recouvrement arrière dans les systèmes répartis mobiles.

Chapitre 7

Introduction à la mobilité.

7.1 Définitions et concepts de base

Dans les systèmes de communications mobiles, les unités mobiles (ou les nœuds mobiles), sont libres de se déplacer à un niveau local, régional, nationales, voire international, tout en recevant et en initiant des appels. Avec l'utilisation croissante d'ordinateurs portables et des périphériques mobiles, des usagers mobiles souhaitent de plus en plus accéder à des services Internet au cours de leur déplacements .Les possibilités qu'offrent ces technologies ont donné naissance à un nouveau paradigme: L'informatique mobile.

7.1.1 L'informatique mobile

L'informatique mobile se réfère à la possibilité pour des usagers munis de périphériques ou d'ordinateurs portables d'accéder à des services et des applications évoluées à travers une infrastructure partagée de réseau, indépendamment de leur localisation physique ou de leur comportement de mouvement.

Elle se distingue donc de l'informatique classique fixe par les deux aspects suivants :

- La mobilité des usagers et d'ordinateurs.
- Les contraintes de ressources mobiles telles la largeur de bande sans fil du support de communication, l'autonomie limitée des batteries, ...etc.

7.1.2 La mobilité

C'est la possibilité de se déplacer dans le réseau tout en gardant la même adresse. On peut accéder aux services offerts par le réseau de n'importe où et à n'importe quel moment. Cela nécessite d'une part des mécanismes de localisation de l'utilisateur, et d'autre part une assurance de la continuité des communications en cours de déplacement.

7.1.3 Les réseaux mobiles

Ils permettent de connecter plusieurs composants mobiles en utilisant des mediums non filaires, c'est pour cela qu'on les appels aussi réseaux sans fil [67]. L'objectif principal des réseaux mobiles est de permettre à leurs utilisateurs de se déplacer en conservant une même adresse tout en restant

connecté dans une zone géographique plus ou moins étendu (zone de couverture) et d'accéder au réseau de n'importe où et à n'importe quel moment.

7.1.4 Les réseaux sans fil

Le concept de sans fil [67], est étroitement associé au support de transmission. Un système est dit sans fil s'il propose un service de communication totalement indépendant de prises murales. Dans cette configuration, d'autres moyens d'accès sont exploités, tels que l'infrarouge ou les ondes hertziennes.

On peut conclure que la classe des réseaux de mobiles est incluse dans la classe des réseaux sans fil. Un réseau de mobile est implicitement un réseau sans fil, mais un réseau sans fil n'est pas forcément un réseau de mobile, la téléphonie sans cordon de résidence DECT fut un exemple. Ce téléphone donne accès au RTC (Réseau Téléphonique Commuté), le réseau classique du téléphone, ou au RNIS (Réseau Numérique à Intégration de Service). Le support de communication utilise l'interface radio pour qu'un abonné puisse appeler depuis son jardin ou sa cuisine, mais l'utilisateur n'a pas toutes les possibilités de déplacement, il doit toujours rester au voisinage de son réseau d'abonnement.

7.2 Comparaison entre réseaux mobiles et réseaux statiques

L'environnement mobile offre beaucoup d'avantages par rapport à l'environnement habituel. Cependant de nouveaux problèmes peuvent apparaître ; ils sont caractérisés par :

7.2.1 La mobilité

C'est la capacité d'accéder à partir de n'importe quel endroit, à l'ensemble des services normalement disponibles dans un endroit fixe et câblé. Aussi de rejoindre n'importe quel réseau sans avoir besoin de changer d'adresse IP.

7.2.2 L'utilisation moindre de fil

Dans les réseaux mobiles les fils sont supprimés ou moins utilisés, cela convient bien aux situations où le câblage est difficile à établir (zones rurales, bâtiments historique,…). Les fils sont remplacés par des liaisons infrarouges ou des fréquences radio comme canaux de communication.

7.2.3 Autonomie

Les unités mobiles ont une contrainte liée à la durée de vie des batteries due au poids limité des unités portables. Il faut économiser autant que possible les transmissions inutiles.

7.2.4 Interférences

Dans les réseaux sans fil, inévitablement, deux signaux envoyés sur la même fréquence radio s'interfèrent et s'altèrent mutuellement, pour y remédier plusieurs techniques sont utilisées illustrées dans [68] [69] :

- FDMA (Frequency Division Multiple Access) le spectre de fréquence est divisé en plusieurs sous bandes qui sont chacune placée sur une fréquence spécifique du canal (porteuse ou carrier). Chaque porteuse ne peut transporter que le signal d'un seul utilisateur.

- TDMA (Time Division Multiple Access) où la totalité de la bande de fréquences est offerte à chaque utilisateur pendant un intervalle de temps (slot).

- CDMA (Code Division Multiple Access) ou accès multiple par répartition de codes. En CDMA, les utilisateurs peuvent communiquer simultanément dans une même bande de fréquence. La distinction entre les différents utilisateurs s'effectue alors grâce à un code qui leur est attribué et connu exclusivement par l'émetteur et le récepteur.

7.2.5 Débit et portée faibles

L'une des limites de la communication sans fil vient de la relative faiblesse de la bande passante des technologies utilisées. Plusieurs facteurs limitent la portée d'une transmission sans fil, comme la faible puissance du signal, les obstacles qui empêchent, atténuent, ou réfléchissent les signaux.

7.2.6 Fiabilité limitée

La communication sans fil est moins fiable que la communication dans les réseaux filaires. La propagation du signal subit des perturbations (erreurs de transfert, diminution de la puissance du signal, micro-coupure, timeout) dues à l'environnement, qui altère l'information transférée.

7.2.7 Détection des collisions

L'atténuation rapide du signal en fonction de la distance, induit l'impossibilité pour un émetteur de détecter une collision au moment même où il transmet.

7.2.8 La sécurité limitée

Les réseaux sans fil offrent de nouvelles failles aux pirates. De part la nature immatérielle du support physique, l'écoute clandestine sur un réseau sans fil est facile. Il faut donc protéger l'accès aux ressources sans fil et aux informations qui circulent dans les trames. Les systèmes mobiles sont généralement équipés de processeurs moins puissants que les machines fixes, et ne peuvent se permettre d'effectuer de longs calculs demandés par les systèmes de cryptographie. L'énergie de la batterie est une ressource rare et pose également des problèmes de sécurité.

7.3 Classification des environnements mobiles

7.3.1 Classification selon l'infrastructure

Les réseaux mobiles (ou sans fil) peuvent être classés en deux catégories : les réseaux avec infrastructure et les réseaux sans infrastructure.

7.3.1.1 Le modèle avec infrastructure

Ce système est composé de deux ensembles d'entités distinctes :

1- Les sites fixes du réseau filaire (wired network).

2- Les sites mobiles (wireless network).

Certains sites fixes, appelés stations mobiles de support (Mobile Support Station) ou station de base (SB) sont munis d'une interface de communication sans fil pour la communication directe avec les sites ou unités mobiles (UM), localisés dans une zone géographique limitée, appelée cellule.

A chaque station de base correspond une cellule à partir de laquelle des unités mobiles peuvent émettre et recevoir des messages. Les sites fixes sont interconnectés entre eux à travers un réseau de communication filaire,

généralement fiable et d'un débit élevé. Les liaisons sans fil ont une bande passante limitée qui réduit sévèrement le volume des informations échangées.

Dans ce modèle, une unité mobile ne peut être, à un instant donné, directement connectée qu'à une seule station de base. Elle peut communiquer avec les autres sites à travers la station à laquelle elle est directement rattachée. L'autonomie réduite de sa source d'énergie, lui occasionne de fréquentes déconnexions du réseau. Sa reconnexion peut alors se faire dans un environnement nouveau, voire dans une nouvelle localisation.

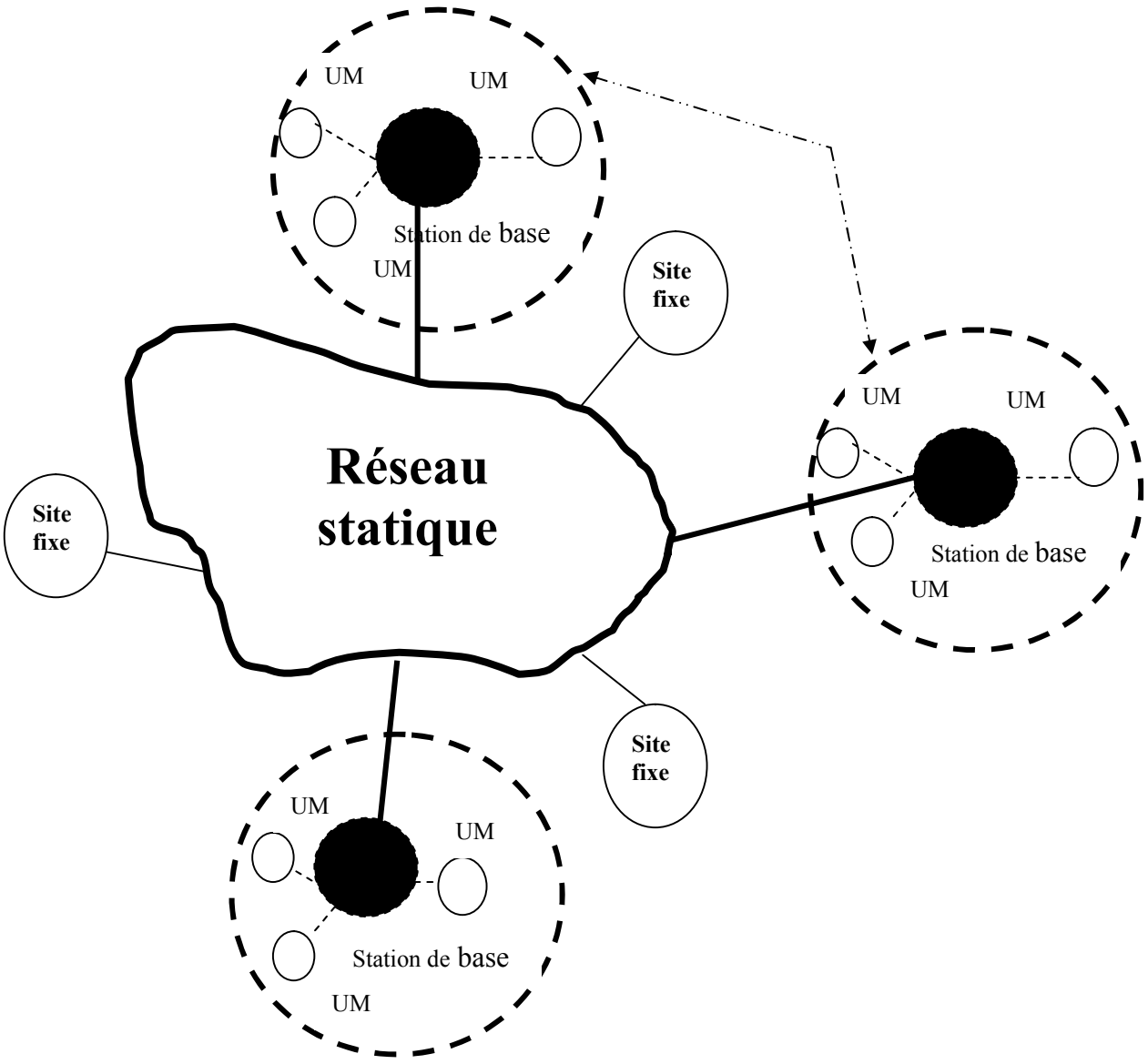


Figure 13 : Le modèle des réseaux mobiles avec infrastructure.

7.3.1.2 Le modèle sans infrastructure : Ad hoc

Le terme latin ad hoc signifie « pour cela », « en vue de cela », « à cet effet » et a un caractère temporaire.

Un réseau mobile ad hoc, appelé généralement MANET (Mobile Ad hoc NETwork), consiste en une population, relativement dense, d'unités mobiles qui se déplacent dans un territoire quelconque et dont le seul moyen de communication est l'utilisation des interfaces sans fil, sans l'aide d'une infrastructure préexistante ou administration centralisée. Le concept des réseaux mobiles Ad hoc essaie d'étendre les notions de la mobilité à toutes les composantes de l'environnement.

Les réseaux mobiles ad hoc sont caractérisés par ce qui suit :

1- Une topologie dynamique

2- Une bande passante limitée.

3- Des contraintes d'énergie.

4- Une sécurité physique limitée.

5- L'absence d'infrastructure.

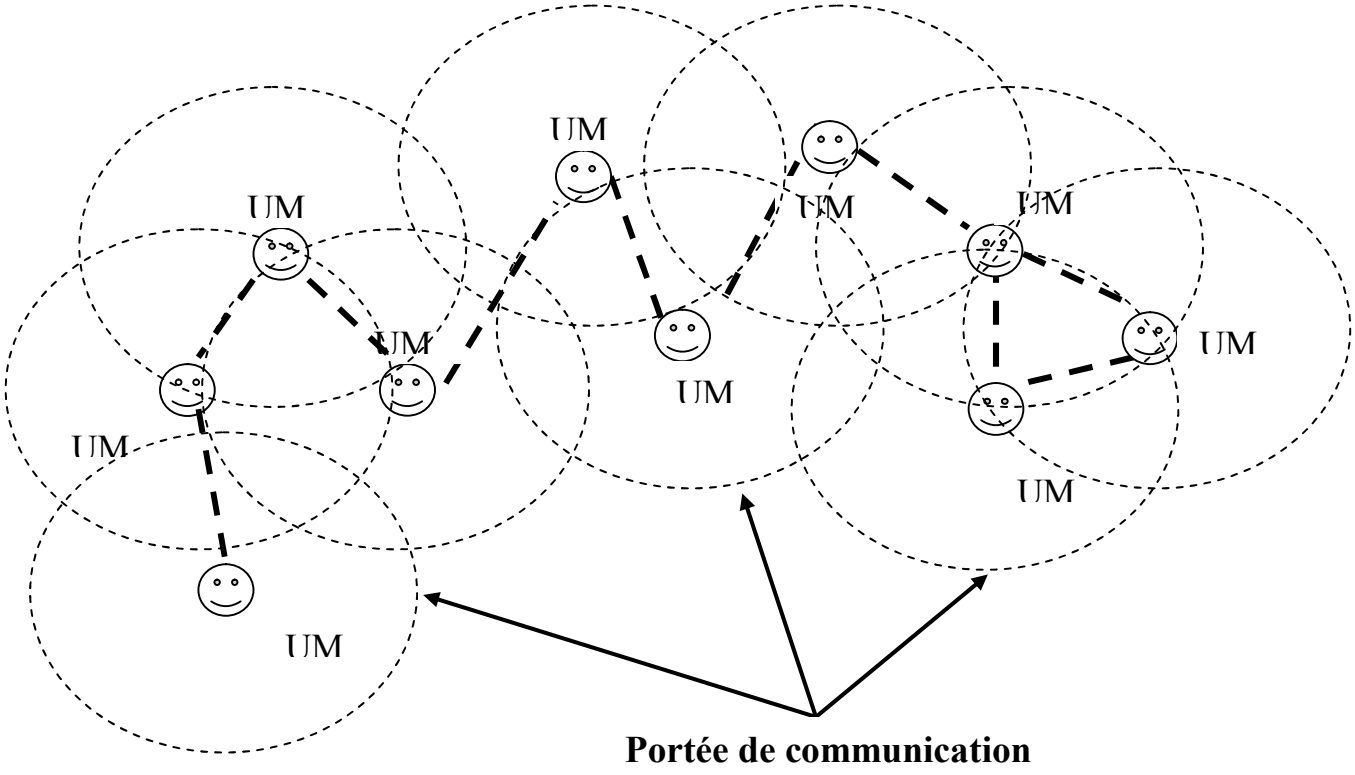


Figure 14 : Le modèle des réseaux mobiles sans infrastructure.

7.3.2 Classification selon le périmètre géographique

On distingue habituellement plusieurs catégories de réseaux sans fil, selon le périmètre géographique offrant une connectivité (appelé zone de couverture) [70]. Cette distinction a donné naissance à de nombreuses technologies sans fil standardisées. Il n'existe pas de technologie parfaite et chacune d'elle représente un équilibre entre différents facteurs (portée, débit, etc.). Le choix d'une technologie sans fil dépend donc de l'usage que l'on souhaite en faire. Une première distinction entre les réseaux sans fils dépend de leur champ d'action. Suivant leur portée, ils pourront permettre la mise en place de **réseaux personnels** (WPAN, Wireless Personal Area Network), **locaux** (WLAN, Wireless Local Area Network), **métropolitains** (WMAN, Wireless Metropolitan Area Network) ou même distants (WWAN, Wireless Wide Area Network).

Bien sûr, une architecture permettant de relier les équipements d'une personne au réseau mondial sera constituée de différents types de réseaux. A chaque niveau d'échelle, du personnel au global, les données pourront passer sur des réseaux sans fils ou filaires.

7.3.2.1 Réseau personnel sans fil WPAN

Le réseau personnel sans fil (appelé également réseau individuel sans fil ou réseau domestique sans fil et noté **WPAN** pour Wireless Personal Area Network) concerne les réseaux sans fil d'une faible portée : de l'ordre de quelques dizaines de mètres. Ce type de réseau sert généralement à relier des périphériques (imprimante, téléphone portable, appareils domestiques, ...) ou un assistant personnel (PDA) à un ordinateur sans liaison filaire ou bien à permettre la liaison sans fil entre deux machines.
Il existe plusieurs technologies utilisées pour les WPAN :

7.3.2.1.1 Le Bluetooth

C’est la principale technologie des WPAN. Elle a été lancée par le suédois Ericsson en 1994, et ce nom fait référence à Harald Blåtand dit "la dent bleue" qui a été le roi du Danemark de 940 à 981. Proposant des connexions avec une consommation réduite d’énergie et bas prix. Rejointe par d’autres grands industriels en 1998 dont IBM, Intel, Nokia, Toshiba, en mai de cette année ils créent le SIG (Special Interest Group) de Bluetooth, ce groupe a été élargi par l’arrivée de 3Com, Agree (Lucent technologies), Microsoft, et Motorola durant

l'année 2000 et ne cesse de s'accroître, aujourd'hui le SIG compte plus de 2500 constructeurs [68].

Bluetooth offre des débits atteignant 720 Kbp/s sur un rayon allant de 10 à 30m. Développé principalement pour remplacer les câbles de connexion entre les machines ou entre les machines et leurs périphériques (clavier, souris, imprimante, …) ou encore communication entre la télévision et le lecteur DVD,….

Bluetooth est normalisé par l'IEEE sous la référence IEEE 802.15.1. Une nouvelle version approuvée en août 2003, basée sur l'IEEE 802.15.3, baptisée UWB (Ultra Wide Band), offre un débit maximal de 55Mbit/s, une portée de 100 m et jusqu'à 245 connexions simultanées tout en offrant un degré de sécurité renforcé grâce à un codage AES (Advanced Encryption Standard). Elle possède l'avantage d'être très peu gourmande en énergie, ce qui la rend particulièrement adaptée à une utilisation au sein de petits périphériques.

7.3.2.1.2 HomeRF

C'est un standard développé en 1998 par le "Home Radio Frequency Working Group", consortium qui inclut au départ : Compaq, IBM, HP, Intel et Microsoft, qui a atteint par la suite environ une centaine de membres. Le standard HomeRF est dérivé de la norme 802.11b [68]. Il utilise la bande de fréquence de 2.4 GHz. Il offre un débit théorique de 10 Mbp/s pour un débit pratique de 3 à 4 Mbp/s partagé entre tous les utilisateurs connectés. Sa portée varie entre 50 et 100 m.

La norme HomeRF a été abandonnée en Janvier 2003, car les concepteurs de processeurs misent désormais sur les technologies Wi-Fi embarquée.

7.3.2.1.3 ZigBee

C'est le prolongement de HomeRF, initiée par Motorola. Zigbee est un réseau pour transporter essentiellement les commandes et non des données. Il permet la mise en place de réseaux personnels sans fils en étoile à très bas coûts. La technologie est soutenue par le consortium "ZigBee Alliance" composé actuellement de 124 compagnies. (Honeywell, Mitsubishi, Samsung, Philips, Chipcon, Freescale, ember…).

Les débits autorisés sont relativement faibles, mais c'est véritablement sa très faible consommation électrique qui en fait son atout principal.

Zigbee fonctionne globalement sur la bande de fréquences de 2,4 GHz mais également 915 MHz en Amérique et 868 MHz en Europe. Les débits offerts sont : 250 Kbits/s à 2.4 GHz (10 canaux), 40 Kbits/s à 915 MHz (6 canaux) et 20 Kbits/s à 868 MHz (1 canal). ZigBee permet de connecter jusqu'à 255 matériels par réseau sur une portée allant jusqu'à 100 mètres.

Le ZigBee, une norme sans fil qui devrait arriver prochainement, semble se dédier à une utilisation en domotique, pour surveiller et contrôler divers appareils, notamment les interrupteurs de lumière et les fours par exemple.

Il existe deux versions de Zigbee :

- IEEE 802.15.4 qui permet de communiquer ave un débit de 250 Kb/s jusqu'à 10 mètres pour relier au maximum 255 appareils (bande de fréquence des 2,4 GHz).
- IEEE 802.15.4a qui est limité à 20 Kb/s mais permet une portée jusqu'à maximum 75 mètres pour relier au maximum 65 000 appareils (bande de fréquence des 900 KHz).

7.3.2.2 Réseaux locaux sans fil WLAN

Le réseau local sans fil (noté **WLAN** pour Wireless Local Area Network) est un réseau permettant de couvrir l'équivalent d'un réseau local d'entreprise, soit une portée d'environ une centaine de mètres. Il permet de relier entre eux les terminaux présents dans la zone de couverture. Il existe plusieurs technologies concurrentes :

7.3.2.2.1 Le Wifi

Ce standard adopté par l'IEEE en 1999, est défendu par l'Alliance pour la Compatibilité de l'Ethernet sans fil (WECA, Wireless Ethernet Compatibility Alliance) [71]. Cet organisme est chargé de maintenir l'interopérabilité entre les matériels répondant à la norme 802.11 b.

Le Wifi permet la mise en place de réseaux locaux, entre de nombreux utilisateurs, avec un débit de transfert théorique de 11 Mbits/s pour un rayon d'action de quelques dizaines de mètres dans la bande de fréquence de 2,4 GHz. Il peut être étendu si on rajoute des relais.

Des évolutions de la norme 802.11b sont en cours de standardisation par l'IEEE :

• Le standard IEEE 802.11 peut être cité à titre historique comme le premier standard de la série (débit théorique 2 Mb/s).

• Le standard IEEE 802.11b : débit théorique 11 Mb/s, une portée de 100 m à quelques centaines de mètres maximum et une bande de 2,4 GHz. Ce standard a permis l'essor des réseaux sans fils ces dernières années.
• Le standard IEEE 802.11a : débit théorique 54 Mb/s (mais décroît avec la distance), une portée d'une trentaine de mètres sur une bande de 5 GHz.
• Le standard IEEE 802.11g : débit théorique 54 Mb/s, portée d'une centaine de mètres et une bande de 2,4 GHz.
• Le standard IEEE 802.11n : débit théorique 320 Mb/s, une portée d'une trentaine de mètres, utilisant les deux bandes 2,4 et 5 GHz. Le 802.11n intègre en base la qualité de service (le standard IEEE 802.11e).
Les extensions :
• Le standard IEEE 802.11e : extension pour un réseau avec signalisation et Qualité de Service.
• Le standard IEEE 802.11f : extension pour le handover (passage d'une cellule à l'autre sans coupure).
• Le standard IEEE 802.11i : extension sécurité.

Le tableau 2, présente les différentes révisions de la norme 802.11 et leur signification.

7.3.2.2.2 Le HiperLAN2 (High Performance Radio LAN 2.0 Hiperlan)

C'est une norme exclusivement européenne, la première version (Hiperlan1) a été définie en juillet 1998 par le comité RES-10 du projet BRAN (Broadband Radio Access Networks) de l'European Telecommunications Standards Institute (ETSI), elle exploite la bande de fréquence 5 à 6,25 GHz en offrant un débit théorique de 20 Mbp/s. Une seconde version (Hiperlan2) [72] développée par l'H2GF (HyperLan 2 Global Forum) fondé en 1999 par Bosch, Dell, Ericsson, Nokia, Telia et Texas Instruments. Ils ont été rejoints un an après par d'autres industriels, tels Canon, Motorola ou encore Samsung. Elle offre un débit de 54 Mbp/s, exploite la gamme de fréquence de 5 GHz, sur une portée de 100m.

Tableau 2 : différentes révisions de la norme 802.11

Norme	Nom	Description
802.11a	Wi-Fi 5	La norme 802.11a (baptisée WiFi 5) permet d'obtenir un haut débit (54 Mbps théoriques, 30 Mbps réels). Cette norme spécifie 8 canaux radio dans la bande de fréquence des 5 GHz.
802.11b	Wi-Fi	La norme 802.11b est la norme la plus répandue actuellement. Elle propose un débit théorique de 11 Mbps (6 Mbps réels) avec une portée pouvant aller jusqu'à 300 mètres dans un environnement dégagé. La plage de fréquence utilisée est la bande des 2.4 GHz, avec 3 canaux radio disponibles.
802.11c		La norme 802.11c n'a pas d'intérêt pour le grand public. Il s'agit uniquement d'une modification de la norme 802.1d afin de pouvoir établir un pont avec les trames 802.11 (niveau liaison de données).
802.11d	Internatio-nalisation	La norme 802.11d est un supplément à la norme 802.11 dont le but est de permettre une utilisation internationale des réseaux locaux 802.11. Elle consiste à permettre aux différents équipements d'échanger des informations sur les plages de fréquence et les puissances autorisées dans le pays d'origine du matériel.
802.11e	Amélioration de la qualité de service	La norme 802.11e vise à donner des possibilités en matière de qualité de service au niveau de la couche liaison de données. Ainsi cette norme a pour but de définir les besoins des différents paquets en terme de bande passante et de délai de transmission de manière à permettre notamment une meilleure transmission de la voix et de la vidéo.

802.11f	Itinérance	La norme 802.11f est une recommandation à l'intention des vendeurs de point d'accès pour une meilleure interopérabilité des produits. Elle propose le protocole Inter- Access point roaming protocol permettant à un utilisateur itinérant de changer de point d'accès de façon transparente lors d'un déplacement, quelles que soient les marques des points d'accès présentes dans l'infrastructure réseau. Cette possibilité est appelée itinérance (ou roaming en anglais).
802.11g		La norme 802.11g offrira un haut débit (54 Mbps théoriques, 30 Mbps réels) sur la bande de fréquence des 2,4 GHz. Cette norme n'a pas encore été validée, le matériel disponible avant la finalisation de la norme risque ainsi de devenir obsolète si celle-ci est modifiée ou amendée. La norme 802.11g à une compatibilité ascendante avec la norme 802.11b, ce qui signifie que des matériels conformes à la norme 802.11g pourront fonctionner en 802.11b.
802.11h		La norme 802.11h vise à rapprocher la norme 802.11 du standard Européen (HiperLAN 2, d'où le h de 802.11h) et être en conformité avec la réglementation européenne en matière de fréquence et d'économie d'énergie.
802.11i		La norme 802.11i a pour but d'améliorer la sécurité des transmissions (gestion et distribution des clés, chiffrement et authentification). Cette norme s'appuie sur l'AES (Advanced Encryption Standard) et propose un chiffrement des communications pour les transmissions utilisant les technologies 802.11a, 802.11b et 802.11g.
802.11j		La norme 802.11j est à la réglementation japonaise ce que le 802.11h est à la réglementation européenne.

7. 3.2.3 Réseau métropolitain sans fil WMAN

Le réseau métropolitain sans fil (**WMAN** pour Wireless Metropolitan Area Network) est connu sous le nom de **Boucle Locale Radio** (BLR). Les WMAN sont basés sur la norme IEEE 802.16. La boucle locale radio offre un débit utile de 1 à 10 Mbit/s pour une portée de 4 à 10 kilomètres, ce qui destine principalement cette technologie aux opérateurs de télécommunication. La norme de réseau métropolitain sans fil la plus connue est le **WiMAX**.

7.3.2.3.1WiMax

C'est une nouvelle norme développée pour offrir des réseaux sans fil à longue distance et à hauts débits. Basée sur des fréquences allant de 2 à 10GHz, elle pourra offrir un débit allant jusqu'a 130 Mbitp/s. En pratique, on pourra avoir un débit de 70 Mbitp/s (par secteur radio et par canal de 20MHz) sur une couverture d'une dizaine de kilomètres. WiMax devrait permettre l'extension des connexions Internet haut débit aux zones non couvertes par l'ADSL, permettant d'obtenir des débits de l'ordre de 70 Mbit/s sur un rayon de plusieurs kilomètres.

Il a défini 2 standards principaux

- IEEE 802.16 pour les fréquences entre 10 et 66 GHz, avec IEEE 802.16c qui propose plusieurs profils (choix d'options) pour ce standard.

- IEEE 802.16a pour les fréquences entre 2 et 11 GHz.

7.3.2.4 Réseaux étendus sans fil WWAN

Le réseau étendu sans fil (**WWAN** pour Wireless Wide Area Network) est également connu sous le nom de réseau cellulaire mobile.
La configuration standard d'un système de communication cellulaire est un maillage (grid) de cellules hexagonales. Initialement, une région peut être couverte uniquement par une seule cellule. Quand la compétition devient importante pour l'allocation des canaux, la cellule est généralement divisée en sept cellules plus petites, dont le rayon est égal à un tiers du rayon de la cellule de départ. Cette subdivision peut être répétée et l'on parle alors de systèmes micro cellulaires. Les cellules adjacentes dans le maillage doivent utiliser des fréquences différentes, contrairement à celles qui sont situées sur les côtés

opposés du maillage et qui peuvent utiliser la même fréquence sans risque d'interférence. La subdivision des cellules se fait comme suit :

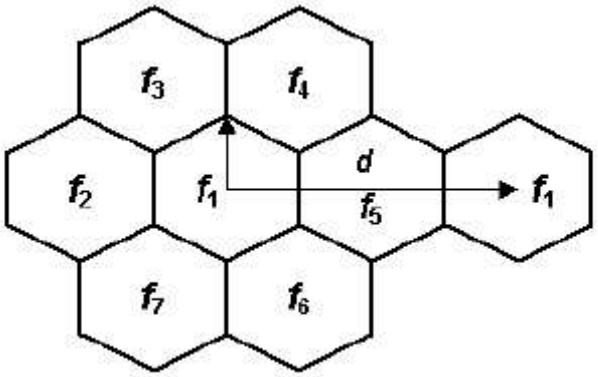


Figure 15 : Réseau cellulaire.

Il s'agit des réseaux sans fil les plus répandus puisque tous les téléphones mobiles sont connectés à un réseau étendu sans fil. Les principales technologies sont les suivantes :

• **GSM** (Global System for Mobile Communication ou Groupe Spécial Mobile) : C'est en 1982, lors de la Conférence Européenne des Postes et Télécommunications (CEPT) que fut créé le Groupe Spécial Mobile (GSM). Celui-ci avait pour objectif de spécifier un système de normes européennes pour les radiocommunications. En 1992, le GSM est rebaptisé « Global System for Mobile communications ».Ce changement de nom symbolise le passage du concept de laboratoire au produit commercial.

Le GSM autorise l'envoi et la réception de messages courts (SMS : Short Message Services), la transmission de données en mode circuit, ainsi qu'un grand nombre de services supplémentaires. La bande de fréquence de cette norme européenne s'élève à l'origine à 900 MHz. Elle a été ensuite mise en oeuvre avec des fréquences autour de 1800 MHz [68].

• **GPRS (**General Packet Radio Service)

Connue également sous le nom de GSM Phase 2+, est une évolution et une amélioration de la norme GSM avec l'introduction de la commutation de paquets, ce mode de transmission a pour avantage de n'utiliser le réseau qu'au cours de la transmission du paquet, et non, comme le GSM pendant toute la

durée de la communication .Avec ce mode de transmission le GPRS offre des débits variables compris entre 9,6 kbits/s et 171,2 kbits/s [68] [69].

• **UMTS** (Universal Mobile Telecommunication System)

Désigne une technologie retenue au niveau de la normalisation internationale dans la famille dite "IMT 2000" comme norme pour les systèmes de télécommunications mobiles de troisième génération [68].

L'UTMS permet des améliorations par rapport au GSM, notamment :

• Un accès plus rapide à Internet depuis les téléphones portables, par un accroissement significatif des débits des réseaux de téléphonie mobile.

• Amélioration de la qualité des communications en tendant vers une qualité d'audition proche de celle de la téléphonie fixe.

• Etablissement d'une norme compatible à l'échelle mondiale, contrairement aux technologies actuelles (les normes utilisées aux Etats-Unis et au Japon ne sont pas toutes compatibles avec le GSM).

• Tend à résoudre le problème croissant de saturation des réseaux GSM, en particulier dans les grandes villes.

• Utilisation du codage DS-CDMA (Direct Sequence – Code Division Multiple Access) qui, outre un nombre important d'avantages, permet la réutilisation de la même bande de fréquence pour toutes les cellules voisines (avec GSM, les cellules voisines ne peuvent utiliser les même bandes de fréquence car elles interfèrent les unes avec les autres d'où un arrangement minutieux et compliqué de bandes de fréquence des cellules).

Les technologies développées autour de la norme UMTS conduiront à une amélioration significative des vitesses de transmission avec des débits supérieurs à 384Kbp/s et pouvant aller jusqu'à 2Mbp/s (en zone urbaine, avec une mobilité réduite).

Les réseaux cellulaires peuvent être classés en 3 générations suivant leurs développement chronologique [68].

La première génération est analogique, nous citons à titre d'exemple les systèmes :

- AMPS (Advanced Mobile Phone System) : développé par Bell Labs en 1970, la première utilisation commerciale était au Etats Unis en 1983. Ll exploite la bande de fréquence de 800 MHz.

- NMT 450 (Nordic Mobile Telephones/450) : développé par Ericsson et Nokia. Travail sur la bande de 450 MHz.

- NMT 900 (Nordic Mobile Telephones/900) : l'utilisation de la bande de 900 MHz au lieu de 450 MHz.

- TACS (Total Access Communications System) : développé par Motorola. Il est similaire à AMPS. Il utilise la bande de 900 MHz. Il a été utilise la première fois au Royaume Uni en 1985.

- C-Netz : utilisé principalement en Allemagne et en Australie, utilise la bande de 450 MHz.

- RC2000 (RadioCom 2000) : Un system Français lancé en novembre 1985.
Les systèmes de radiotéléphonie cellulaire ont vu, après une première génération constituée uniquement de réseaux analogiques, l'arrivée des technologies numériques au début des années 1990 en Europe (GSM) et au Japon (PDC). Les Etats-Unis ont suivi quelques années plus tard (IS-136 et IS-95).

Il existe une génération 2+ (dite aussi 2.5 G) avec le GPRS, évolution du système GSM. Celui-ci permet la commutation de paquets et des débits de 105 KBits/s. Dans ce cas, IPV4 est obligatoire.

La troisième génération permet d'utiliser IP dans un contexte multimédia. Les terminaux peuvent avoir une adresse fixe, certaines normes comme l'UMTS impose IPv6. Le nom générique pour les différentes normes 3G est IMT-2000.

Tableau 3 : Générations des réseaux cellulaires.

<table>
<tr><th>1ère génération (1 G)</th><th>2ème génération (2 G)</th><th>génération 2+ (2.5 G)</th><th>3ème génération - IMT-2000 (3 G)</th></tr>
<tr><td rowspan="4">AMPS
NMT 450
NMT 900
TACS
C-Netz
RC 2000</td><td>GSM (Global System for Mobile Communi-cation)
Utilisé en Europe et en Amérique.</td><td rowspan="4">GPRS (General Packet Radio Service) extension du GSM pour les communications de données</td><td>EDGE (UWC-136) (Enhanced Data rates for Global Evolution) Une évolution du GSM/GPRS et du TDMA.EDGE est aussi parfois appelé E-GPRS (Enhaced GPRS)</td></tr>
<tr><td>D-AMPS (IS-136) (Digital Advance Mobile Phone System) plus connu sous le nom TDMA (ANSI-136)
Utilisé en Amérique et Asie Pacifique.</td><td>UMTS (Universal Mobile tele - communications System) Une évolution du GSM</td></tr>
<tr><td>N-CDMA (IS-95) de Qualcomm (Code Division Mulitiple Access) Utilisé en Amérique</td><td>cdma2000 (Code Division Multiple Access) Une évolution du N-CDMA</td></tr>
<tr><td>PDC (Personal Digital Cellular)
Utilisé au Japon</td><td></td></tr>
</table>

Chapitre 8

Algorithmes de reprise dans les réseaux mobiles.

8.1 La sûreté de fonctionnement mobile plus qu'un objectif, un besoin !

Durant la dernière décennie, le monde de l'environnement mobile a été témoin d'une énorme effervescence. Les progrès technologiques, surtout dans le domaine des communications, ont vu l'émergence d'un nouveau défi dans le monde informatique : la communication sans fil, grâce à laquelle nous sommes passés des réseaux fixes vers les réseaux sans fils interconnectant des machines mobiles. Cette Informatique mobile prend un rôle et une place chaque jour plus important dans tous les aspects des activités humaines, mais l'actualité nous montre bien leur vulnérabilité. Il devient donc indispensable de savoir et définir leur sûreté.

La prise en compte de la mobilité a rendu l'implantation de la sûreté de fonctionnement plus complexe, et comme on dit cerner le problème c'est en partie trouver la solution, et les problèmes des systèmes répartis existent depuis le temps et ont été étudiés au chapitre II. Mais les caractéristiques des réseaux mobiles ont donné naissance à de nouvelles fautes, pour ne pas dire que parfois, ces caractéristiques sont elles même ces fautes.

8.2 Quelques caractéristiques des environnements mobiles

L'émergence de la technologie sans fil permet à l'utilisateur de se libérer des contraintes temporelles et spatiales. En effet, l'utilisateur peut continuer à accéder aux données fournies par une infrastructure répartie, quelque soit son emplacement. Malheureusement, comme signalé précédemment, l'environnement mobile se distingue de l'environnement statique par :

- La connexion sans fil dispose d'une bande passante très variable en terme de performance et de fiabilité (Par exemple, un WAN peut avoir une bande passante allant de 2 à 20 MB/S, par contre un WAN utilisant des cellules numériques de transmission de données peut avoir une bande passante de 19.2 KB/S).

- Contrairement aux environnements distribués statiques, l'environnement mobile est sujet à des déconnexions fréquentes (volontaires ou non).

- Les dispositifs mobiles sont plus pauvres en ressources telles que la

capacité de la mémoire et la vitesse d'exécution. Ces caractéristiques sont dues à la mobilité élevée qui impose au terminal mobile d'être petit et léger.

- Les dispositifs mobiles dépendent de la charge de la batterie, qui est une source d'énergie à capacité limitée.

La figure 16 nous montre un résumé sur ces caractéristiques :

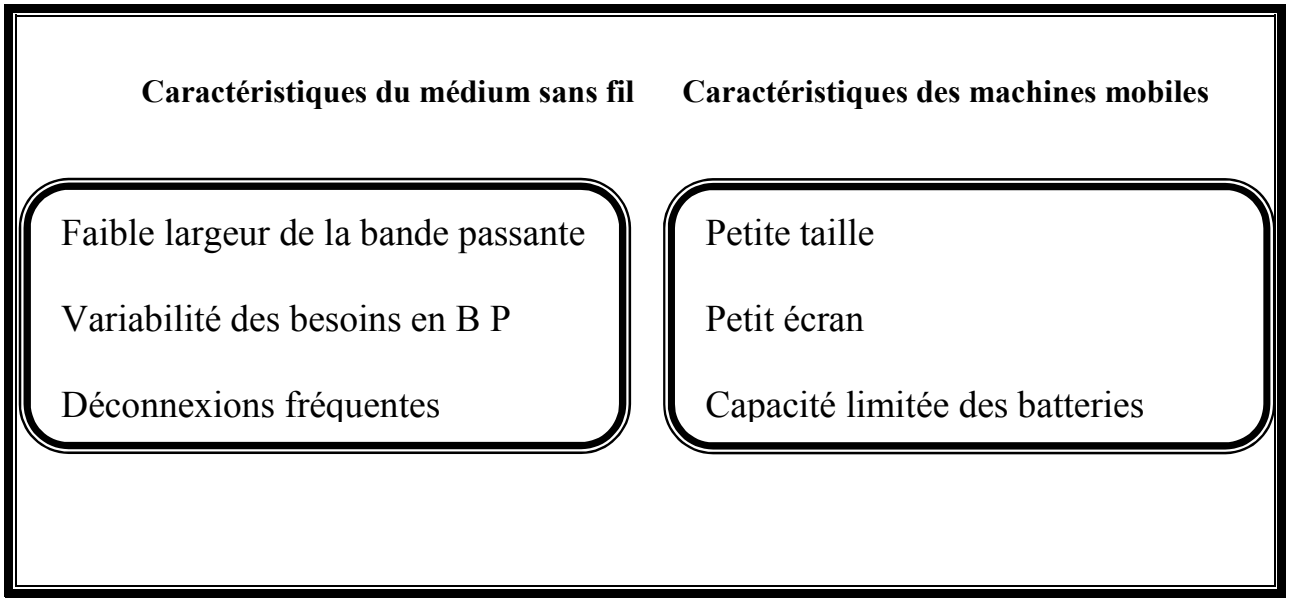


Figure16: Caractéristiques des environnements mobiles.

8.3 La classification des fautes

Bien évidemment, pour chaque nouvelle technologie, correspond une nouvelle spécification. L'environnement mobile est menacé par plusieurs types de fautes qui peuvent être :

8.3.1 Fautes du Logiciel

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

8.3.2 Fautes de l'Environnement

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

8.3.3 Contraintes Architecturales

À cause des conditions de mobilité, les appareils mobiles sont petits et légers et ont une faible capacité de ressources i.e. la puissance de la batterie, la capacité mémoire...etc. Le tableau ci-dessous présente la classification des fautes.

Tableau 4 : La classification des fautes en environnement mobile.

Faute	**Nature de la faute**	**Présence de la faute**
La réduction de la largeur de Bande Passante	Environnementale	transitoire
La capacité mémoire	Architecturale	Permanente
Les problèmes de la batterie	Architecturale	Permanente
Perte de paquets	Environnementale	transitoire
Hors de portée	Environnementale	transitoire
Arrêt du système	Logicielle	transitoire
Panne de transaction	Logicielle	transitoire

Le manque en ressources, représente lui-même une faute, parce que il peut perturber l'exécution d'une application. Contrairement aux fautes de logiciel et fautes d'environnement, les fautes architecturales sont toujours présentes et de nature différente.

Remarque

Il est à prévoir que les contraintes architecturales seront améliorées en fonction de l'évolution technologique. Des recherches actives dans ce domaines envisagent l'extension de la durée de vie de la batterie, la puissance des CPU et aussi l'accroissement de la taille de la mémoire.

8.4 La tolérance aux fautes et les réseaux mobiles

Aujourd'hui, on peut présenter plusieurs mécanismes de tolérance aux fautes. Chacun supporte des fautes particulières. Ce réservoir de mécanismes permet de trouver un mécanisme bien adapté pour une application particulière, mais en même temps, ce choix est très difficile à réaliser car les applications ont des attributs différents. Mais déjà ce choix est l'objet d'une étude approfondie. Nous optons pour le recouvrement arrière pour les mêmes raisons évoquées dans l'étude précédente.

8.5 Implémentation du recouvrement arrière dans les réseaux mobiles

8.5.1 Modèle du système

Les systèmes distribués sont une collection de processus qui communiquent les uns aux autres à travers des messages. Un système informatique mobile est un système distribué où quelques processus s'exécutent sur des noeuds mobiles ou mobile hosts (MHs) qui peuvent se déplacer. Ils n'existent entre eux ni mémoire ni horloge commune.

En fait, ce système représente la liaison entre deux réseaux : l'un statique connectant des sites statiques appelés stations de support mobiles (MSSs) ou encore stations de base, l'autre mobile ou sans fils connectant des nœuds mobiles.

Pour communiquer avec les MHs, une MSS est liée aux autres MSSs avec une liaison filaire, tandis qu'avec un MH elle est reliée à travers le réseau sans fils.

On suppose que les canaux utilisent une stratégie FIFO (First In, First Out) pour ordonner les messages. A chaque station de base correspond une cellule à partir de laquelle des unités mobiles peuvent émettre et recevoir des messages.

8.5.2 Prise en compte des caractéristiques des réseaux mobiles

Les contraintes qu'imposent les réseaux sans fils, sont les nouvelles considérations à prendre dans la conception des algorithmes de reprise.

8.5.2.1 La limitation des ressources

A cause du déplacement des sites, ces derniers sont soumis à des contraintes de minimisation de poids et de volume.

8.5.2.1.1 Faible capacité de stockage

Les MHs disposent d'une mémoire (vive ou morte) très limitée. De ce fait, ces mémoires ne peuvent pas être surchargée par trop d'informations. Il serait plus convenable alors d'utiliser des algorithmes n'effectuant pas trop de sauvegardes, et ne manipulant pas des structures de données trop volumineuses.

8.5.2.1.2 Faible énergie de la batterie

Pendant toutes les activités d'un MH, les batteries sont présentes pour alimenter les MHs, mais malheureusement leur durée de vie est limitée par leur taille et l'énergie qu'elles renferment. Dans ce cas, il faut opter pour des algorithmes qui minimisent l'effort du MHs, utilisé dans la création des points de reprise, leur stockage en lieu sûr et la collecte de ces points proprement dite.

8.5.2.1.3 Largeur de la Bande passante limitée

C'est une contrainte imposée par le réseau lui-même, le remède consiste à choisir des algorithmes ne générant pas trop de trafic sur le réseau, par l'échange de messages de synchronisation, de sauvegarde sur le support stable ou encore l'utilisation de messages énormes. Aussi n'appelant pas trop de sites à faire un retour arrière.

8.5.2.2 Vulnérabilité des réseaux mobiles face aux pannes catastrophiques

Les MHs sont exposés aux pertes, vols, pannes physiques…De ce fait, le disque sur un MH ne peut être considéré comme un moyen de stockage ni fiable ni encore moins stable. Une solution raisonnable est d'utiliser un stockage

stable dans les MSSs [73], pour sauvegarder les points de reprise des MHs. Mais pour prendre ces derniers, un MH doit transférer une énorme quantité d'information à son MSS à travers le réseau sans fils. Et puisque les réseaux sans fils ont une faible largeur de la bande passante et les MHs ont relativement une faible capacité de traitement, les algorithmes de reprise doivent concerner un nombre limité de processus pour la procédure d'établissement de points de reprise.

8.5.2.3 La mobilité

A un certain moment, un MH change de cellule, donc automatiquement il change de MSS. Quand un MH change de cellule on dit qu'il a quitté son MSS. Quand ce dernier revient vers son ancienne MSS ou s'y rattache à une nouvelle, on dit qu'il a rejoint cette MSS (voire la figure 17).

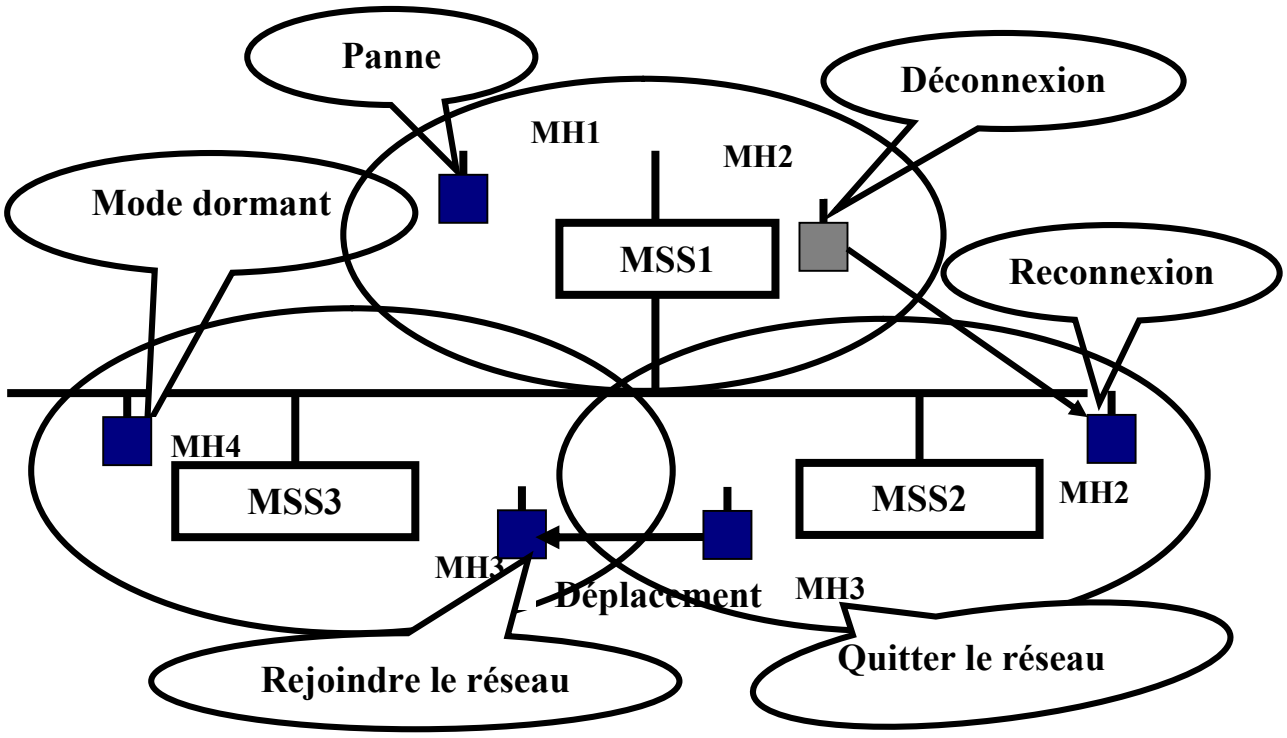


Figure17 : Exemple d'un système informatique mobile.

Le changement d'emplacement d'un MH complique le routage des messages. Le message envoyé par un MH à un autre MH peut être routé à nouveau si le MH a changé de cellule. Malgré les différentes techniques d'adressage de mobilité utilisées dans les protocoles de routage, localiser les nœuds, augmente le temps de communication et la complexité des messages.

Dans les algorithmes de reprise classiques dédiés à l'environnement réparti fixe, nous devons parfois sauvegarder l'histoire répartie de l'exécution de l'application. Mais dans le cas où un MH enregistre un point de reprise dans sa station de base locale MSS1, puis change de cellule, et fait la même chose avec MSS2, …, MSSn, rassembler les points de reprise sera une tâche très lourde à réaliser.

Pour que la mobilité soit transparente, les algorithmes de checkpointing déjà proposés ont opté pour deux solutions :

8.5.2.3.1 L'enregistrement des déplacements d'un MH

Cela consiste en la sauvegarde de l'itinéraire emprunté [74] par n'importe quel MH, en associant un graphe des MSSs visitées par ce noeud.

8.5.2.3.2 L'avertissement du site lui-même

Un MH lui-même doit informer le système de sa position. Par exemple dans [75], le protocole de checkpointing est doté de deux routines :

8.5.2.3.2.1 Un MH quitte la MSS

Quand un MHi décide de quitter une MSSp vers une MSSq ou quitter sa cellule courante, il envoie à MSSp un message l'avisant qu'il va la quitter (**Leave-Message**) ce message portera l'identité de ce MH.

8.5.2.3.2.2 Un MH rejoint son ancienne ou une nouvelle MSS

De la même façon que lorsqu'il veut quitter une MSS, le MH envoie un message de rejoindre (**Join-Message**) à la nouvelle MSS en même temps qu'il envoie le message de quitter à l'ancienne. Ce message encapsulera la valeur de l'ancienne MSS.

8.5.3 Prise en compte des modes de fonctionnement d'un MH

8.5.3.1 Les modes de fonctionnement d'un MH

Un mobile peut opérer dans trois modes :

8.5.3.1.1 Le mode connecté

Dans ce cas, le mobile dispose d'une connexion normale au réseau comme une station classique. La connexion peut alors être réalisée soit par une liaison filaire, soit par une interface de communication sans fil, qui fournit en général des débits plus faibles qu'une liaison câblée.

8.5.3.1.2 Le mode partiellement connecté

Dans ce cas, le mobile ne dispose plus, pour communiquer avec le réseau, que d'un lien à faible largeur de bande (connexion faible ou déconnexion partielle). Cette perte de capacité de la bande passante peut être due soit à des perturbations ou à des surcharges de la station de base qui gère les communications des mobiles se trouvant dans sa cellule, soit à l'éloignement du mobile de la station de base.

8.5.3.1.3 Le mode dormant

Pour économiser de l'énergie, un MH peut désactiver les composants individuels pendant la période de basse activité [71]. Cette stratégie est appelée le mode dormant (doze mode). Le MH dormant est réveillé par la réception d'un message.

8.5.3.1.4 Le mode déconnecté

Dans ce cas, le mobile est totalement déconnecté du réseau. Soit volontairement ou comme conséquence de la mobilité même des utilisateurs [77].

La figure 17 illustre un exemple des modes de fonctionnement d'un MH dans un réseau mobile. Si un MH entre dans la cellule d'une autre MSS, le canal sans fil lié à l'ancienne MSS est déconnecté et un nouveau canal est alloué à la nouvelle MSS. Un MH peut, volontairement se déconnecter du réseau, et pendant le temps de déconnection, ce MH ne reçoit aucun message. Comme il peut se reconnecter à nouveau à n'importe quelle MSS.

8.5.3.2 Détection des déconnexions

Ordinairement, dans une application répartie localisée sur plusieurs hôtes mobiles, un hôte est considéré défaillant à cause soit d'un arrêt soit d'une déconnexion. Quand un hôte mobile est défaillant, son état volatile est perdu. Jusque là, on ne peut pas faire la distinction entre une défaillance et une déconnexion involontaire, qui est le résultat de coupures intempestives des connexions physiques dues par exemple à un passage de l'utilisateur dans une zone d'ombre radio. Et une déconnexion voulue, décidée par l'utilisateur depuis son terminal mobile et justifiée par les bénéfices attendus sur le coût financier des communications, le niveau d'énergie de la batterie, la disponibilité du service applicatif. Cette dernière peut être traitée comme une interruption de service planifiée, anticipée et préparée. La déconnexion involontaire continue à être traitée comme une défaillance. Cependant, l'existence de détecteurs de connectivité offre la possibilité de prévoir les déconnexions involontaires, donc de les anticiper et de les préparer [78].

L'ajout du détecteur de connectivité permet d'avoir l'information liée au contexte d'un nœud mobile tel que le niveau de batterie, le pourcentage de la bande passante et sa capacité à maintenir une connexion vers une autre machine [79]. La machine mobile peut prévoir une déconnexion proche et ainsi disposer de suffisamment de temps pour effectuer des traitements. Suivant les besoins de l'application, il est possible de transférer les données nécessaires pour travailler en mode déconnecté, de charger un mandataire pour effectuer certaines opérations et aussi d'avertir les autres participants de la prochaine déconnexion. Les autres noeuds avertis d'une déconnexion d'un hôte mobile s'abstiennent de lui envoyer d'autres messages et ce jusqu'au prochain avertissement d'une reconnexion.

8.5.3.3 Gestion des déconnexions

Les déconnexions étant très fréquentes, une machine mobile doit être capable de continuer ses exécutions même avec une faible connexion ou sans connexion du tout [72].

Le problème de la déconnexion a un impact important sur le déroulement d'une procédure de reprise, la question qui se pose est comment recueillir les informations participant à une reprise quand ces dernières appartiennent à un

MH déconnecté. La solution est d'imposer que toutes les transactions concernant un MH, se terminent avant que ce dernier ne quitte le réseau [75].
Deux cas peuvent se présenter :

8.5.3.3.1 Déconnexion d'un MH d'une MSS

Quand un MHi décide de se déconnecter d'une MSSi il prend un point de reprise local qui est sauvegardé dans la mémoire de la MSSi à laquelle il est relié. Ce dernier est appelé point de reprise de déconnexion (**disconnect-checkpoint**).

8.5.3.3.2 Reconnexion d'un MH à une MSS

L'intervalle de déconnexion se termine quand un MHi déconnecté est reliée à l'ancienne MSSi ou à une nouvelle. Il exécute une routine de reconnexion. Supposons que MHi garde de sa mémoire stable l'identité de la dernière MSS à laquelle il était relié soit MSSp. En étant relié avec MSSq, la routine de reconnexion envoie une requête, à travers MSSq, à MSSp.
Si le MHi n'a pas gardé l'identité de son dernier MSS pour une raison qui soit, alors la requête est diffusée sur le réseau. En recevant la requête, MSSp exécute les pas suivants:

- Si MSSp avait traité une demande de checkpointing concernant MHi pendant l'intervalle de déconnexion, le point de reprise (**disconnect-checkpoint**) est envoyé à MHi.

- Si aucune demande de checkpointing impliquant MHi n'a été reçue par MSSp pendant l'intervalle de déconnexion, seulement les messages reçus à son insu lui sont envoyés.

Quand les données envoyées par MSSp arrivent à MHi, celui-ci exécute une des actions suivantes :

- ✓ Si les données reçues contiennent un **disconnect-checkpoint,** MHi l'enregistre comme étant son dernier point de reprise local, puis reprend ses activités.

- ✓ sinon MHi reprend une communication normale avec les autres noeuds du système.

8.6 Conception d'algorithmes de reprise dans les réseaux mobiles

En plus de la difficulté de conception des algorithmes de reprise, déjà signalée dans le cadre des réseaux statiques qui visent à éviter certains phénomènes (effet Domino, perte de messages …..), un deuxième constat s'impose également. Il est évident que les modèles classiques proposés pour les systèmes répartis statiques sont inappropriés, ils doivent adopter de nouvelles spécifications et solutions adaptées à la mobilité du système.

Les contraintes qu'imposent les réseaux sans fils (manque de mémoire stable, faible largeur de bande passante, mobilité élevée, et la limitation de vie de la batterie…) sont la considération la plus sollicitée dans la conception des algorithmes de reprise les plus complexes.

On ne pourrait pas avancer qu'il n'y a pas eu d'algorithmes de checkpointing proposés spécialement pour les réseaux mobiles, mais du moins on pourrait confirmer que tous ces algorithmes ne sont que des versions améliorées des algorithmes répartis fixes.

8.6.1 Algorithmes de reprise dans les réseaux mobiles

D'après notre étude, nous concluons que certains algorithmes ne peuvent pas être portés en environnement mobile, en raison de leur éloignement des objectifs attendus par ces derniers. Car en plus des objectifs déjà cités dans la deuxième partie de cette étude, ces algorithmes doivent respecter les contraintes imposées par le réseau sans fils.

L'idée de porter l'étude du fixe vers le mobile a surgit en 1993 dans [74], et avait pour but le traitement de la mobilité, en sauvegardant en plus des points de reprise l'itinéraire emprunté par le site.

8.6.1.1 Points de reprise indépendants

Les algorithmes d'établissement de point de reprise non coordonnés, ont été les premiers algorithmes proprement dits proposés pour le mobile. Comme déjà évoqué dans le cadre des systèmes répartis fixes, cette méthode consiste à faire prendre aux différents processus du système des points de reprise de manière totalement indépendante [73], [80], [81]. Cela implique la sauvegarde de plusieurs points de reprise. Si ces points devaient être transférés sur un

support de stockage stable, le coût en messages de transfert serait très élevé. Le calcul d'une ligne de recouvrement, via un graphe de dépendance entre les points de reprise locaux, n'est fait que pendant le recouvrement, ce qui peut s'avérer très coûteux en énergie. Le ramassage des points, est compliqué. Mais l'inconvénient majeur de cette technique est l'effet Domino, ce qui rend cette technique peu tentante, nous verrons par la suite que la méthode non coordonnée est combinée avec d'autres méthodes pour faire face à certaines contraintes du mobile.

8.6.1.2 Points de reprise coordonnés

8.6.1.2.1 Coordination explicite

C'est la technique vers laquelle beaucoup de chercheurs se sont penchés, sans doute parce qu'elle garantie la cohérence de la ligne de reprise.

Cette approche repose sur une coordination et une synchronisation globale des processus de l'application. Un coordinateur sauvegarde son point de reprise et diffuse une requête aux autres processus de l'application pour établir leur point de reprise, ce qui implique une latence importante et un coût considérable en messages de synchronisation.

La nouvelle idée est de construire des algorithmes utilisant la coordination, mais en minimisant le nombre de processus participant à la reprise, par conséquent minimiser le coût de la synchronisation, et visant la transparence de la procédure de checkpointing par rapport à l'exécution de l'application.
La technique de sauvegarde coordonnée de points de reprise est classée dans deux catégories : les algorithmes bloquants [84] et les algorithmes non bloquants [76], [83], [84], [85]. Les premiers forcent les processus à geler leur travail courant lors de l'algorithme, l'autre permet aux processus de continuer leur tache courante en parallèle avec l'algorithme, en attribuant des numéros de séquence aux messages, pour éviter les messages orphelins.

Le checkpointing coordonné, ne génère pas d'effet domino, stocke un nombre minimum de checkpoints (2 au maximum) en contre partie il utilise un surplus de messages de synchronisation et peut bloquer l 'application pendant une période de temps qui peut avoir un impacte néfaste sur son rendement. Sinon

elle utiliserait des structures de données énormes pour éviter le blocage. Le recouvrement est très simple.

8.6.1.2.2 Coordination réalisée par des horloges (time based)

Dans cette approche, on utilise les horloges synchronisées ou des temporisateurs pour coordonner l'établissement des points de reprise [51], [55], [86]. On remarque que la notion de coordinateur n'a pas totalement disparue, mais est incluse implicitement dans le protocole. On remarque une moindre utilisation de toutes les ressources, mais le problème est la déviation temporelle entre les horloges qui pourrait être très grande.

8.6.1.3 Points de reprise induits par les communications

Cette technique utilise les biens faits des deux techniques citées auparavant : coordonné et indépendant. La synchronisation est assurée par les messages de l'application, on appelle cela la phase "paresseuse" [33], [56], [89]. Chaque processus prend régulièrement des points de reprise de manière indépendante. On distingue deux familles de protocole :

8.6.1.3.1 Les protocoles model-based

Ces protocoles sont basés sur les notions de chemins et de cycles en "zig-zag" pour déterminer les états globaux cohérents. Les communications et les prises de checkpoints doivent respecter un certain motif.

3.6.1.3.2 les protocoles index-based

Ce type de protocole utilise une méthode basée sur la suppression des dépendances causales entre les checkpoints. Chaque message porte l'index du dernier point de reprise de l'émetteur.

Aucun message spécifique de coordination n'est envoyé, mais les points de reprise sont réalisés lors d'évènements particuliers, donc nous sommes sûr de former un état global cohérent.

8.6.1.4 Journalisation pessimiste

Tous les messages (journal) émis doivent être conservés de façon synchrone et sur un support stable [87], [88], au cas où il serait nécessaire de les rejouer. Cette méthode garantie une reprise très sûre après une panne et n 'implique aucune autre station MH car il faut tout simplement rejouer les

messages sauvegardés. Aussi on a besoin d'aucune coordination, mais elle est très coûteuse en espace mémoire et en message de sauvegarde sur le support stable.

8.6.1.5 Journalisation optimiste

À la différence de la journalisation pessimiste [80], le journal n'est pas écrit directement en mémoire stable mais est d'abord écrit en mémoire volatile pour être vidé périodiquement vers la mémoire stable. Donc la fréquence de l'accès à la mémoire est limitée mais peut conduire à impliquer d'autres MH lors de la reprise en cas de journalisation en retard et donc beaucoup de nœuds vont faire un retour arrière.

8.6.1.6 Journalisation causale

Cette technique combine les avantages de la journalisation optimiste et de la journalisation pessimiste [65].

Le journal est sauvegardé en mémoire volatile mais la décision de l'écrire en mémoire stable est prise à partir des dépendances causales entre les évènements. Ceci implique l'ajout d'informations additionnelles aux messages (dépendances causales) et par conséquent l'augmentation de la taille du message à sauvegarder et à transmettre vue la largeur de la bande passante limitée.

8.6.1.7 Les techniques hybrides

Ces techniques combinent entre deux ou plusieurs méthodes entre celles citées auparavant. Comme par exemple la L'algorithme de Prakach-Singhal [75], qui a combiné les deux techniques : bloquante et non bloquante, l 'algorithme de Hirogak-Takiwaza [92] qui combine entre les algorithmes synchrones et indépendants, ou encore celle de Lin et dow [93] qui combine entre l'approche coordonnée et celle induite par les communications. Leur but est de faire bon usage des avantages de chaque méthode qu'elle combine

8.6.2 Comparaison entre ces différentes techniques de sauvegarde

La comparaison cette fois ci va se jouer sur des éléments autres que dans le cas des réseaux fixes. Cela ne veut pas dire que la comparaison déjà citée n 'est plus à considérer.

Mais tout simplement, ajouter d'autres aspects qui ont surgi avec la mobilité :

- Consommation d'énergie en fonction des messages échangés et des MHs impliqués, exploitation de la bande passante… etc.

- La consommation de mémoire stable.

8.6.2.1 La consommation d'énergie, exploitation de la bande passante

Les algorithmes coordonnés ne sont pas trop convenables dans le cadre de l'environnement mobile si on voulait optimiser sur l'énergie. Lors de la synchronisation, il y' a un échange considérable de massage donc une consommation d'énergie élevée. Dans les algorithmes induits par messages, et non coordonnés, et par journalisation, la fréquence de la prise d'un point de reprise dépend des communications des MHs et n'est pas contrôlée par ces dernières. Dans le pire des cas, un point de reprise est transféré par chaque message sortant ce qui génère un trafic énorme sur le réseau ce qui diverge avec la largeur de la bande passante.

8.6.2.2 La consommation de mémoire stable

Les algorithmes coordonnés ne consomment pas trop de mémoire, ils effectuent au pire des cas deux points de reprise comme dans le cas des points de reprise mutables [76]. Les algorithmes non coordonnés et induits par message, consomment trop de mémoire et comme déjà dit on peut avoir autant de points de reprise qu'il y a de messages dans l'application.

Dans ce cas, il n'est pas nécessaire d'évoquer les algorithmes par journalisation parce que ces derniers ne font pas qu'enregistrer des points de reprise mais aussi les messages échangés, aussi les informations rajoutées aux messages vont faire augmenter la taille de celui ci.

8.6.3 Etude de quelques propositions d'algorithmes

Beaucoup d'algorithmes ont été proposés, surtout durant la dernière décennie, nous en avons choisi quelques uns en nous basant sur les deux facteurs suivants :

- Le premier réside dans l'originalité des idées apportées, car ces algorithmes ont servis de points de départ pour de nouvelles propositions (certaines propositions n'ont fait que reformuler les anciennes).

- Le deuxième concerne la primeur des propositions émises faisant ainsi de leurs auteurs les pionniers du contexte mobile.

8.6.3.1 Proposition de Krishna, Pradhan et Vaidya (1993)

Dans leur article [74], Krishna, Pradhan et Vaidya ont, pour la première fois, remis en cause, l'adaptabilité des algorithmes de checkpointing classiques aux caractéristiques spécifiques mobiles.

L'idée introduite, est qu'il fallait aussi sauvegarder l'itinéraire des hôtes MH, pour minimiser le temps de recherche et de routage des messages vers ces sites. Cet algorithme, n'impose aucune méthode de checkpointing et est doté d'un mécanisme de sauvegarde de l'itinéraire pour chaque MH. Pour ce faire, deux manières sont possibles :

1. À chaque MH est associée une liste contenant l'historique des MSSs visitées par ce dernier. Cette liste est enregistrée au niveau de la MSS courante, est transférée et modifiée chaque fois que le MH se lie à une nouvelle MSS. Cette stratégie est appelée la stratégie *Lazy*.

2. Chaque fois qu'un MH se déplace, son point de reprise sauvegardé dans la MSS courante, est transféré vers la nouvelle MSS, ceci est appelée l'approche *pessimiste*.

Discussion :

Cet algorithme travaille en combinant deux algorithmes, un pour la sauvegarde des points de reprises et l'autre pour la sauvegarde des déplacements, sans proposer au préalable un moyen de les coordonner. Cela pourrait s'avérer très coûteux en messages si on choisissait par exemple la méthode non coordonnée pour le checkpointing et l'approche pessimiste. Ce qui impliquerait que plusieurs points de reprise seront transférés sur le support de stockage des MSSs.

8.6.3.2 Proposition de Arup Acharya et B.R .Bardinath (1994)

Pour la première fois, Acharya et Bardinath proposaient dans [73], un algorithme prenant en considération les caractéristiques imposées par la mobilité, et ayant pour but la réduction du coût de recherche des sites impliqués.

Cet algorithme s'exécute en deux phases :

- La première est l'établissement des points de reprises par les MHs de façon asynchrone indépendamment des autres, en utilisant les informations additionnelles encapsulés dans les messages de l'application, et leur transfert vers les supports de stockage stables des MSSs courantes.

- La deuxième est que le recouvrement est laissé à la responsabilité des MSS pour déterminer une ligne de reprise cohérente.

Discussion :

Nous nous verrons entrain de boucler dans les inconvénients des stratégies non coordonnées, comme le risque d 'effet Domino, aussi le nombre de points de reprise à sauvegarder avant la détermination d'un point de reprise global cohérent est élevé ce qui implique un coût de stockage et de transfert élevé.

8.6.3.3 La proposition de Ravi Prakash et Mukesh Singhal 1996

Prakash et Singhal sont partis du principe qu'un bon algorithme d'établissement de points de reprise devrait être, non bloquant (non intrusif) et efficace.

Un l'algorithme non bloquant, ne force pas les noeuds dans le système à geler leurs calculs pendant la procédure de checkpointing. Un algorithme efficace, utilise un effort minimum dans cette même procédure.

Cela peut être accompli en forçant un sous-ensemble minimal de nœuds du réseau, à prendre leurs point de reprise locaux, et en employant une structure de données qui ne soit pas trop encombrante.

Cet algorithme [75], se base sur la coordination non bloquante, alors si un site initie le checkpointing, seulement les sites ayant communiqué directement ou transitivement avec lui, depuis son dernier point de reprise y seront impliqués.
Un des aspects intéressants de l'algorithme est qu'il est réalisé en deux phases :
Une phase paresseuse (Lazy phase) qui permet aux noeuds de prendre des points de reprise temporaires localement, dans un mode quasi asynchrone en se basant sur les informations rajoutées dans les messages de calcul, et une phase coordonnée dite la phase agressive, qui permet de rendre permanents, les points provisoires déjà créés.

Les sites impliqués vont établir leurs point de reprise provisoires, ensuite, ils vont confirmer cette action à l'initiateur, qui une fois tous les messages d'acquittement reçus, enverra des messages de validation de point de reprise à tous les sites impliqués pour rendre permanents leurs points de reprise provisoires.

Discussion :

Nous remarquons que la conservation d'énergie et la contrainte de la bande passante ont prises en compte, la quantité de travail à refaire concerne seulement les sites participants à l'opération qui a causé l'échec. De plus, la phase paresseuse fait avancer le processus de checkpointing lentement. Cela évite la concurrence pour les canaux à basse bande passante. Tandis que l'utilisation complexe de beaucoup de structures de données le support de stockage des checkpoints provisoires ne sont pas sûrs.

Mais la raison la plus fondée c'est que les mêmes auteurs affirment que cet algorithme a généré des états d'incohérence dans [90] et dans [82]. Ces mêmes auteurs prouvent qu'il n'existe pas d'algorithmes qui soient bloquant et impliquant un nombre minimal de site en même temps.

8.6.3.4 Proposition de Nuno Neves et W Kent Fuchs 1997

Partis de l'idée d'adapter la création des points de reprise à la qualité de service du réseau, les auteurs ont nommé cette approche « adaptative » [91]. L'algorithme utilise la coordination indirecte, tout en introduisant dans l'ancienne architecture des réseaux mobiles la notion de station originale Home Station (HS) et station étrangère Foreign Station (FS).

A chaque site est associée une HS avec laquelle il interagit comme dans un réseau fixe. Quand il se déplace vers une nouvelle station de base, sa station de base originale le représente, elle reçoit les messages qui lui sont destinés et les expédie vers la station étrangère à laquelle il est actuellement rattaché. Aussi cette HS est informée de tous les déplacements et les points de reprise pris par ce dernier y sont sauvegardés.

Les sites sauvegardent périodiquement leur état local à l'aide d'un temporisateur local (coordination basée sur le temps). Les états peuvent être sauvegardés localement dans la mémoire sur MH dans ce cas nous avons des points de reprise soft, leur création n'encombre pas la bande passante, ou transférés vers la mémoire stable de la HS, et là

nous allons avoir des points de reprise hards. Il adapte le comportement des points de reprise à la qualité de service du réseau, donc pour un réseau a une QOS minime, alors l'algorithme prend beaucoup de points soft avant de les valider en un seul hard.

Discussion :

On remarque qu'il y a une diminution de la consommation d'énergie surtout dans le cas de la coordination (messages de synchronisation), aussi il y a moins d'accès au support de stockage de la MSS. L'inconvénient réside dans la détermination d'une horloge commune (synchronisation des horloges réparties). Cet algorithme a motivé plusieurs études, comme par exemple la proposition de Lin, Wang, Kuo et Chen en 2002 dans [94] qui vise à réduire au maximum le nombre de points de reprise transmis sur le réseau sans fils.

8.6.3.5 la proposition de Cuohong Cao et Mukesh Singhal mai 1998

Dans leur article [83], les auteurs présentent une nouvelle perception des points de reprise : ce sont les points de reprise "mutable ". A un instant donné, les nouveaux points de reprise créés, ne sont ni provisoires, ni permanents, et qui peuvent être sauvegardés n 'importe où, en évitant de cette façon le coût de transfert de leur stockage dans la mémoire des MSSs.

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

Lorsqu'un MH envoie un message, il y encapsule la valeur du dernier point de reprise **csn (chekpoint séquence number)** pris sur ce dernier. A la réception de ce message par le site destinataire celui-ci agit de deux façons :

- Si la valeur csn reçue est plus petite que la valeur du csn sur le MH récepteur alors traiter normalement ce message.
- Sinon, il doit prendre un point de reprise mutable et le sauvegarder localement, évitant ainsi le coût de son transfert vers les MSSs, avant de traiter le message.

Si par la suite, le site reçoit une requête de checkointing, soit :

- Il transformera son checkpoint mutable en point provisoire.
- En réponse à la requête de checkpointing, il prend un checkpoint provisoire.

Dans les deux cas le checkpoint provisoire est transféré vers le support stable de la MSS courante, et la requête de checkpointing reçue, est circulée vers tous les sites dépendants, les invitant à faire la même chose.

Discussion :

L'algorithme a réalisé les buts assignés. Seulement, on remarque que la quantité d'information à ajouter est énorme et ne converge ni avec la capacité ni encore avec l 'instabilité du support de stockage des sites mobiles. La suite de cette idée, des checkpoints mutables, est implémentée par les mêmes auteurs en 2001 dans [76] et a servi de base pour des propositions récentes, comme en janvier 2005 où Kummar, Chauhan et Gupta, dans [84], trouvent une technique pour garder le même numéro de séquence pour les points de reprise permanents les plus récents (cns) sur le réseau, dans le but d'uniformiser les informations de dépendance, et alléger les structures de données encapsulées dans les messages

8.6.3.6 la proposition de Cuohong Cao et Mukesh Singhal août 1998

Cet article [82], ne ressemble pas aux autres, car dans son état de l'art, ces auteurs ne se sont pas contentés de critiquer les algorithmes déjà faits, mais il sont allés jusqu'à prouver qu'il n 'existe pas d'algorithmes bloquants et impliquant un nombre minimal de sites. Et ce qui est plus surprenant, c'est que l'article critiqué [75], est conçu par l'un des auteurs en question.

L'algorithme est réalisé en deux phase : la première est la prise de checkpoints provisoires et la deuxième consiste en la transformations de ces derniers en points de reprise permanents.

Chaque site est muni d'un vecteur de dépendance de N éléments, ce vecteur est maintenu par la station de base courante de ce site. Lorsqu'un MH initie un checkpointing il prend un point de reprise provisoire et envoie sa requête de

checkpointing à sa MSS qui va s'occuper du reste pour le compte de ce MH. Ne disposant pas des informations sur les dépendances, cette dernière envoie une demande de vecteurs de dépendances à toutes les MSS du réseau et se bloque. A la réception des demandes, chaque MSS lui envoie les vecteurs de dépendances de tous les sites qui sont dans sa cellule, et se bloque à leur tour.

Tous les vecteurs reçus, la MSS initiatrice calcule la matrice de dépendance (NxN), qui servira à établir la liste des sites impliqués par ce checkpointing. Cette liste sera envoyée avec une requête de checkpointing à toutes les MSSs.

Chaque MSS vérifie d'abord si un site qui lui est lié, figure dans cette liste, si oui elle lui transmet la requête reçue. Les opérations d'acquittement et de validation vont se faire via les MSSs.

Discussion :

Cet algorithme a relativement satisfait les contraintes des réseaux mobiles ou du moins les a allégé, par le transfert de la procédure de checkpointing sur le réseau fixe. Le seul désagrément, est le temps de blocage des MSSs en l'attente des vecteurs de dépendances, qui peut être très contraignant pour certaines applications. Cet inconvénient a fait l'objet de plusieurs études d'optimisation dont la dernière est celle de Proposition de Li-Shu en juillet 2005 dans [86] qui vise à réduire le temps nécessaire pour sauvegarder les points de reprise en réduisant le temps de blocage des sites en envoyant directement des requêtes de checkpointing aux sites dépendants détectés par l'encapsulation d'informations additionnelles dans les messages de l'application.

8.6.3.7 la proposition de Hiroaki Higaki et Makoto Takiwaza octobre 1998

Nous pouvons voir dans cet article [92], les bienfaits de la combinaison des deux stratégies coordonnée et non coordonnée. La première utilisée pour la sauvegarde d'un état global cohérent au niveau des sites fixes (MSS) et la deuxième pour la sauvegarde au niveau des sites mobiles (MH). Les auteurs utilisent la technique de journalisation des messages pour coordonner entre les deux méthodes, de cette façon une sauvegarde de points de reprise effectuée sur les MH pourrait rattraper une sauvegarde établie au niveau des MSSs.

L'inconvénient majeur de cette proposition est la complexité élevée au niveau de la technique de journalisation proposée, surtout dans le cas ou l'application est à échange de messages intensif.

8.6.3.8 Proposition de Cheng Min Lin et Chyi Ren Dow 2001

Dans leur article [93], les auteurs présentent une stratégie hybride, mais différente de celle de Higaki et Takiwaza [92] par la combinaison cette fois de l'approche coordonnée et la sauvegarde induite par les communications.

Les sites fixes prennent leurs points de reprise de façon coordonnée et les sites mobiles le font de façon induite par les messages de communication en ajoutant des informations additionnelles.

L'idée passe par trois phases :

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

Discussion :

Par l'utilisation de la coordination, cet algorithme non bloquant, assure un état cohérent global et un coût d'exécution minimum, mais il dévie vers une complexité énorme lorsque l'application exige un échange important en messages.

Dans le cadre des algorithmes hybrides on pourrait citer la proposition de Tantikul, Manivannan, en décembre 2005 dans [89] qui combine entre la méthode non coordonnée et celle induite par les communications.

8.6.4 Constat

D'après ce qui vient de vu, nous pouvons conclure qu'il n' y a pas eu d'algorithmes de checkpointing conçus spécialement pour les réseaux mobiles, mais ce sont des algorithmes dédiés à l'environnement fixe, qu'on a essayé de remodeler pour cacher les désagréments de vulnérabilité, manque de stockage stable, faible capacité de la batterie, déconnexions… nés avec la mobilité.

En optimisant soit sur la mémoire, soit sur l'effort fourni, soit sur l'inhibition de l 'application pendant la procédure d'établissement des points de reprise, on est sûr de satisfaire un ou deux critères au maximum, ou donner naissance à un nouveau problème. Par exemple, on pourrait prendre l'algorithme de Acharya et Bardinath [73] et essayer d'optimiser sur la consommation de l'espace de stockage sûr et le surcoût en messages de sauvegarde, ou encore essayer d'optimiser sur la consommation d'energie dans l'algorithme de Cao et Singhal [83] en proposant des structures de données moins volumineuses, etc.…

En proposant seulement des petits changements dans une structure de données, un petit indice, en combinant entre plusieurs méthodes ou enfin en imposant des hypothèses supportées par une application particulière, ces propositions sont restées jusque là non prouvées et parfois inadéquates.

Nous constatons aussi qu'à partir de ce point, décider de l'efficacité d'un algorithme de reprise dans l'environnement mobile revient à prouver, sa capacité à produire en fin d'exécution un état global cohérent, capable de relancer l'application sans aucune inconsistance, ce qui nous pousse à adopter l'approche coordonnée, et pour d'autres raisons :

- Les algorithmes coordonnés ont fait l'unanimité de presque toutes les études récentes, sans doute pour les avantages qu'elle procure.
- Le maintien de un ou deux (au maximum) points de reprise en support de stockage stable.
- Le ramassage des points de reprise locaux est simple, il suffit de prendre le dernier point de reprise.

- L'accès au support stable n'est pas fréquent, lors du stockage des points de reprise.

- Pas besoin d'intégrer un mécanisme de ramasse miette, car la création d'un nouveau point est rendu permanent en supprimant l'ancien.

Son inconvénient majeur réside dans le coût en messages de synchronisation encouru.

8.6.5 Idée d'optimisation

Nous choisissons pour notre optimisation, l'algorithme de Cao-singhal (1998) [82], car en fait, c'est l'algorithme le plus satisfaisant et répondant aux exigences des réseaux mobiles. Bien qu'il implique seulement les sites ayant communiqué directement ou indirectement avec le site initiateur, il bloque l'application, et parfois, ce blocage peut devenir inacceptable.

L'idée est de choisir une station fixe (deux au moins, cela dépend des moyens disponibles) ou en créer une nouvelle, à la différence des stations fixes, elle relie les MSSs entre elles, qu'on appellera « serveur de dépendance ». Cette station gère tous les messages circulants entre les MSS et les MH, si un MHi lié à MSSp, envoie un message m à MHj situé sur MSSp ou MSSq, alors ce message est automatiquement transféré vers le serveur de dépendance, qui change le vecteur de dépendance de MHi puis ré achemine ce message vers sa destination. Le but de l'ajout du serveur de dépendances et de minimiser le temps de blocage des MSS et MH.

L'algorithme de Cao et Singhal [82], pourrait être diviser en deux étapes :

1- La première est la collecte d'informations de dépendances :

- Un MHi prend un point de reprise provisoire et envoie une demande d'établissement de point de reprise, qui est transférée à la MSSp courante.

- Celle-ci envoie une demande à toutes les stations de base pour rassembler les informations de dépendance et se bloque, en attendant les réponses.

- À la réception des demandes de dépendances, les MSSs envoient les vecteurs de dépendances de tous les sites qui leur sont liés puis, se bloquent.

- MSSp reçoit les vecteurs, et calcule une matrice de dépendance, qui va contenir l'identité des dépendances directes et indirectes, et dégager la liste des sites impliqués. Elle envoie par la suite une requête s'établissement d'un point de reprise provisoire accompagnée de la liste des sites impliqués, à toutes les MSSs.

2- Le lancement de la procédure de checkpointing

- À la réception des demandes et des listes, chaque MSSs vérifie si un ou plusieurs sites figurants dans la liste, lui est lié. Si oui, elle lui achemine la demande d'établir un point de reprise provisoire.

- Les sites impliqués vont établir leur point de reprise provisoire, ensuite, ils vont le confirmer à la MSS courante.

- Une fois toutes les confirmations reçues par la MSS courante, elle les envoie à la MSS initiatrice.

- La MSS initiatrice, continue d'attendre, jusqu'à l'obtention de toutes les confirmations pour enfin envoyer des messages de validation de point de reprise à tous les sites impliqués pour rendre permanent leur point provisoire.

Notre proposition vise à minimiser le temps de blocage des sites en jouant sur le temps de l'obtention des dépendances. Et l'algorithme serait le suivant :

- Un MHi prend un point de reprise provisoire et envoie une demande d'établissement de point de reprise, qui est transférée à la MSSp courante.

- La demande est directement envoyée au serveur de dépendances.

- À la réception de messages, ce serveur réagira de deux façons :

- Si ce message est une requête de chechpointing, alors il fait le calcul de sa matrice de dépendance et diffuse cette requête et la liste des sites élus, à toutes les MSS.

- Si ce message est un message de l'application, reçu après une requête de checkpointing, il le garde dans un journal (pour éviter de créer de nouvelles dépendances), sinon il l'achemine vers sa destination.

2- Le lancement de la procédure de checkpointing

- À la réception des demandes et des listes, chaque MSSs vérifie si un ou plusieurs sites figurants dans la liste, lui est lié. Si oui, elle lui achemine la demande d'établir un point de reprise provisoire.

- Les sites impliqués vont établir leur point de reprise provisoire, en suite, ils vont le confirmer à la MSS courante.

- Une fois toutes les confirmations reçues par la MSS courante, elle les envoie à la MSS initiatrice.

- La MSS initiatrice, continue d'attendre, jusqu'à l'obtention de toutes les confirmations pour enfin envoyer des messages de validation de point de reprise à tous les sites impliqués pour permaniser leur point provisoire.

Avantages et inconvénients

Cette politique a l'avantage de : premièrement rendre le blocage implicite car tous les messages ne sont acheminés que par l'ordre du serveur de dépendance, deuxièmement elle minimise le temps de collecte des informations de dépendance. Mais son inconvénient est la centralisation de l'opération de la collecte, il faut prévoir une autre station qui pourra jouer en cas de panne le même rôle que celle-ci.

Conclusion

La large utilisation des ordinateurs portables et les progrès technologiques dans les réseaux sans fil ont fait des applications mobiles une réalité. De nombreuses applications et interfaces fournissent déjà divers services dédiés aux dispositifs mobiles. La communication sans fil, le traitement d'information personnelles et les services d'information réparties auront une importance stratégique dans l'avenir proche. Malheureusement, le luxe qu'a fourni la mobilité, reste sans mots devant les limitations en mémoire, bande passante, énergie de la batterie et les déconnexions fréquentes, imposées aux dispositifs mobiles. En dépit de tous les problèmes mentionnés ci-dessus, l'utilisation des machines mobiles devient de plus en plus courante. Il est donc souhaitable de fournir les éléments nécessaires pour que leur utilisation devienne aussi naturelle que possible. Un utilisateur mobile souhaite se déplacer librement et continuer à travailler le plus normalement possible avec son terminal mobile sans se préoccuper de sa position. Il convient alors de trouver des solutions pour régler ces désagréments ou du moins les minimiser en adaptant les applications répartis aux environnements mobiles.

Dans chaque système, la faute est devenue naturelle. Quelles que soient les précautions prises pour éviter l'introduction de fautes, qui provoquent la défaillance du système, et les efforts fournis pour les détecter et les corriger, elles se produisent quand même. Dans une application répartie, la défaillance de l'un des processus fait défaillir l'ensemble de l'application. Pour être sûr de fonctionnement, un système doit donc comporter des mécanismes de tolérance aux fautes qui lui permettent de rester opérationnel dans le cas de manifestation de fautes.

Dans notre étude, nous avons choisi le recouvrement arrière par reprise, pour réaliser un système tolérant aux fautes, qui consiste à substituer à l'état erroné par un état existant, présumé correct, avant l'occurrence de la faute. L'analyse des propositions existantes (à notre connaissance), nous a permis de dégager les aspects suivants :

- Chacun de ces algorithmes optimise avec la prise en considération d'un critère ou deux liés à la mobilité, mais pas la totalité, l'évitement de quelques phénomènes (effet Domino, perte de messages,…) mais pas l'ensemble, donc les solutions

proposées donnent une meilleure exécution seulement dans un environnement particulier visant les objectifs déterminés au préalable par le concepteur.

- Chacune des solutions ne considèrent pas les autres fautes que l'application pourrait rencontrer en même temps. La raison est comme déjà constatée en étudiant les type de fautes. Chacune, possède ses propres caractéristiques. Donc ce n'est pas possible de satisfaire tous les besoins d'utilisateur.

- L'environnement ou les ressources disponibles peuvent changer avec le temps, donc ces propositions convergent avec l'évolution des réseaux mobiles dans l'axe du temps. Ceci impliquera que l'algorithme ne va plus être aussi convenable qu'il l'était lors de sa conception, ou peut être deviendra-t-il pas convenable du tout.

Donc la conception ou juste la proposition d'un algorithme de reprise dans le cadre des réseaux mobiles n'a été jusque là qu'une adaptation ou une reprise du fixe vers le mobile en considérant les contraintes de ce dernier, mais avec le temps nous nous verrons forcés de témoigner de leur non convenance.

Références bibliographiques

[1] M. Satyanarayanan. “Pervasive Computing: Vision and Challenges”. IEEE Personal Communications, August 2001.

[2] J. C. Laprie. “Sûreté de fonctionnement des systèmes : concepts de base et terminologie”. Rapport LAAS N°04520, 2004

[3] L. Gelders, L. Pintelon. “Reliability and Maintenance”. International Encyclopedia of Robotics : Applications and Automation. R.C. Dorf and S.Y. Nof (eds). pp. 1305-1316. John Wiley, New York, 1988.

[4] B. Dhillon. Robot Reliability and Safety. Springer-Verlag, 1991.

[5] I. Walker, J. Cavallaro. “The Use of Fault Trees for the Design of Robots for Hazardous Environments”. Annual Reliability and Maintainability Symposium, pp. 229-235. 1996.

[6] M. Visinsky, J. Cavallaro, I. Walker. “A Dynamic Fault Tolerance Framework for Remote Robots”. IEEE Transactions on Robotics and Automation, Vol. 11, no. 4, pp. 477-490. August 1995.

[7] L. Laprie. “Sûreté de fonctionnement des systèmes informatiques : concepts de base et terminologie”. Rapport technique 94448, LAAS, Toulouse, France, Septembrer 1994.

[8] A. Avizienis, J. C. Laprie and B. Randell. “Fundamental Concepts of Dependability”. In 3rd Information Survivability Workshop, (Boston, MA, USA), pp.7-12. 2000.

[9] CENA/NT968623/SdF/S. METGE.

[10] L. S. Momo. “Réplication et Durabilité dans les systèmes répartis”. 19 février 2001.

[11] C. Baron. “Définition de la tolérance aux fautes Sûreté de fonctionnement des systèmes”. Dependability of system Claude.Baron@insa-tlse.fr.

[12] G. Zwingelstein. “Sûreté de fonctionnement des systèmes industriels complexes ”. Techniques de l’ingénieur, traité informatique industrielle.

[13] J. C. Laprie, B.Courtois, M. Gaudel et D. Powell. “ Sûreté de fonctionnement des systèmes informatiques logiciels et matériels ”. ISBN 2604601694263 Novembre 1987.

[14] W. Heimerdinger, C. B. Weinstock. “A conceptual framework for system fault tolerance”. Technical report. CMU/SEI-92-TR-033, Software Ingeneering Institute, Carnegie Mellon University, October 1992.

[15] F. Tiaini. “La Réflexivité dans les architectures multi niveaux : application aux systèmes tolérants les fautes”. Janvier 2004.

[16] L. Lamport, R. Shostak and M. Pease. “The Byzantine Generals Problem”. ACM Transactions on Programming Languages and Systems, 4 (3), pp.382-401, July 1982.

[17] J. C. Laprie, J. Arlat, H. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac, P. Thévenod. “Guide de la sûreté de fonctionnement”. Cépaduès (ISBN: 2.85428.382.1). Laboratoire d’Ingénierie de la Sûreté de Fonctionnement (LIS : Matra-Marconi Space, LAAS-CNRS, Technicatome), 1995-1996.

[18] N. Mahmoud, S. Attelly. “ La tolérance aux fautes dans les systèmes informatiques ”. Instia-dess quassi-génie logiciel , Février 2002.

[19] T. Anderson, P. A. Lee. “Fault-Tolerance Principles and Practice”. ISBN 0-13-308254-7 ? Prentice –Hall International, Inc., 1981.

[20] J. B. Stefani, S. Krakowiak. “Tolérance aux fautes: Principes et Mécanismes ”.

[21] B. Randell, P. A. Lee, P. C. Treleaven. “Reability issues in computing system design: Computing Surveys”. Vol.10,N°2, pp.123-165.1978.

[22] M. Aliouat. “Recovery in distributed system from solid fault “. Université de Constantine, Algérie, SAFECOM 92.

[23] C. Morin, A. Kermarrec, M.Banatre. “An efficient and scalable approach for implementing fault tolerant DSM architectures “. IRISA publication interne 1997.

[24] P. Ramanathan, K .G. Shin. ”Use of common time base for checkpointing and rollback recovery in distributed system”. IEEE,Trans.SoftwareEng,vol.SE_19,n°=6,PP.571-583. [25] R. Baldoni, J. M.Helary, A. Mostefaoui, M. Raynal. “A communication induced checkpointing protocol that insures rollback-dependency trackability “. IRISA publication interne, n°1076 January 1997.

[25] R. Baldoni, J. M.Helary, A. Mostefaoui, M. Raynal. “A communication induced checkpointing protocol that insures rollback-dependency trackability “. IRISA publication interne, n°1076 January 1997.

[26] D. L. Russel. “State restoration in systems of communicating processes”. IEEE Trans, software .Eng,vol.SE-6 ,n°2,PP.183-194 , March 1980.

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

[28] K. M. Chandy. ”A survey of analytic models of rollback and recovery strategies”, University of Texas at Austin .

[29] B. Randell. ”System structure for software fault tolerance”.IEEE, Trans.Software Eng, vol. SE -1, n°=2, pp. 22-230, June 1975.

[30] K. M. Chandy, L. Lamport. “Distributed snapshots: Determining global states of distributed systems”. In ACM Transactions on Computer Systems, pp. 63-75, 1985.

[31] L. Lamport. “Time, clocks, and the ordering of events in a distributed system”. In Communications of the ACM, vol. 21, pp 558-565, 1978.

[32] R. H. B. Netzer, J. Xu. “Necessary and sufficient conditions for consistent global snapshots“. In IEEE Transactions on Parallel and Distributed Systems, Vol. 6, pp 165-169, 1995.

[33] D. Manivannan, M. Singhal. ” Quasi-synchronous checkpointing: models, characterization, and classification”. IEEE Transactions on Parallel and distributed Systems, Vol. 10, No. 7, pp. 703-713, 1999.

[34] R. H. B. Netzer, J. M. Hélary, A. Mostefaoui and M. Raynal. “Communication-based prevention of useless checkpoints in distributed computations”. In Proc. 16th IEEE Symposium on Reliable Distributed Systems, pp. 183-190, 1997.

[35] R. H. B. Netzer, J. M. Hélary and M. Raynal. “Consistency issues in distributed checkpoints”. In IEEE Transactions On Software Engineering, Vol 25, 1999.

[36] G. Deconinck, J. Vounckx, R. Cuyers, and R. Lauwereins. “Survey of Checkpointing and Rollback Techniques”. Technical Reports of Esprit Project 6731 (FTMPS) 03.1.8 and 03.1.12, ESAT-ACCA Laboratory, Katholieke Universiteit Belgium, June 1993.

[37] A. Bouteiller, F.Cappello, Thomas Hérault, G.Krawezik, P. Lemarinier, and F. Magniette. “A fault tolerant MPI for volatile nodes based on pessimistic sender based message logging”. In High Performance Networking and Computing (SC2003), Phoenix USA. IEEE/ACM, , November 2003.

[38] R. F. Storm, S. Yemini. “Optimistic recovery in distributed system”. ACM Trans. On computer System Vol.3, N°2 , pp. 204-226. 1985.

[39] K. Kim, J. You, A. Abounadja. “A scheme for coordinated execution of independently designed recoverable distributed processes”. Proc 16th IEEE Symp. Fault Tolerant Comput. pp. 130-135. 1986.

[40] B. Bhargava, S. R. Lian.” Independent Checkpointing and concurrent rollback for recovery in distributed system: an optimistic approach”. Symp.Reliable Distributed Systems SRDS 488, 1988.

[41] Z. Tong, R. Y. Kain, W. T. Tsai. “Rollback recovery in distributed systems using loosely synchronized clocks”. IEEE Trans. On Parallel and distributed Systems, Vol. 3, N°.2, pp. 246-251. 1992.

[42] M. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson. “A survey of rollback-recovery protocols in message passing systems”. Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, October 1996.

[43] K. H. Kim. “Programmer-Transparent coordination of recovering concurrent processes: Philosophy and rules for efficient implementation”. IEEE Trans .on Software Eng .Vol. 14, N°6, pp. 810-821. 1988.

[44] J. L .Kim. “An efficient protocol for checkpointing recovery in distributed systems”. IEEE Trans .On Parallel and distributed Systems .Vol. 4, N °8, pp 955-960. 1993.

[45] G. Barigazzi, L. Strigini. “Application-Transparent Setting of Recovery Points“. In Proc. 13th IEEE Symposium on Fault Tolerant Computing, Milano (Italy), June 1983.

[46] J. L. Kim, T. Park. “An efficient algorithm for check pointing recovery in distributed systems”. IEEE Transactions on Parallel and distributed systems, vol.8, pp. 59-96, 1993.

[47] L. M. Silva, J. G. Silva. “Global Checkpointing for distributed programs”. Proc. 11th Symp. Reliable Distributed Systems. pp. 382-388, 1976.

[48] E. N. Elnozahy, D.B. Jhonson. “The performance of consistent check pointing”. Proc. 11th Symp. Reliable Distributed Systems. pp. 86-95, 1992.

[49] F. Cristian, F. Jahanian. “ A timestamp- based checkpointing protocol for long-lived distributed computations”. In proceeding of the IEEE International Computer Performance and Dependability Symposium. pp. 12-20. September 1991.

[50] Z. Tong, R. Y. Kain AND W. T. Tsai. “A low overhead and rollback recovery scheme for distributed systems”. In proceeding of the 8th Symposium on Reliable Distributed Systems. pp. 12-20, October 1989.

[51] N. Neves, W. K. Fuchs. “Coordinated checkpointing without direct coordination”. In proceeding of the IEEE International Computer Performance and Dependability Symposium. pp. 23-31. September 1998.

[52] C. Briatico and Simoncini. “A distributed domino-effect free recovery algorithm”. In IEEE International Symposium on Reliability, Distributed Software, and Databases, 1984.

[53] L. Alvisi, E. Elnozahy, S. S. Rao, S. A. Husain, and A. Del Mel. “An analysis of communication-induced checkpointing”. Technical report, 1999.

[54] K. Venkatech, T. Radhaktishan and F. Li. “Optimal checkpointing end local recording for Domino-free rollback recovery”. Information processing letters, vol. 25, pp. 295-303, 1987.

[55] Y. M. Wang , W. K. Fuchs. “Lazy checkpoint coordination for bounding rollback propagation”. In proceeding of the 12th I International Symposium on Reliable Distributed Systems. pp. 78-85, 1993.

[56] D. Manivannan, M. Singhal.”A low-overhead recovery technique using quasisynchronous checkpointing”. In Proceedings of the 16th ICDCS, 1996.

[57] D. B. Johnson, W. Zwaenepoel. “Sender-based message logging“. In The 7th annual international symposium on fault-tolerant computing. IEEE Computer Society, 1987.

[58] M. L. Powell, D. L. Presotto. Publishing. “A reliable broadcast communication mechanism”. In Proc. 9th ACM Symposium on Operating Systems Principles, BrettonWoods (USA), October 1983.

[59] A. Borg, J. Baumbach, and S. Glazer. “A Message System Supporting Fault Tolerance”. In Proc. 9th ACM Symposium on Operating Systems Principles, BrettonWoods (USA), October 1983.

[60] D. B. Johnson, W. Zwaenepoel. “Recovery in distributed systems using message logging and checkpointing”. Journal of Algorithms, Vol. 11, N°3. pp. 462-491, 1990.

[61] T. Chandra, V. Hadzilacos, S. Toueng. “The weakest failure detector solving consensus”. Journal of the ACM, 43(4), pp. 685-722, July 1996.

[62] P. A. Lee, T. Anderson. ”Fault Tolerance: principles and practice”. Springer-Verlag, 1990.

[63] M. Ben-Or. “Another advantage for free choice: completely asynchronous agreement protocols “. I proc, of second annual ACM Symposium of Distributed Computing (Poc’83), pp. 27-30, August 1983.

[64] E. N. Elnahazy, W.Zwaenapoel. “Transparent rollback–recovery with low overhead, limited rollback and fast output commit”. IEEE Trans. Comp. Vol 41, n°. 5. May 1992.

[65] L. Alvisi, B. Hoppe, and K. Marzullo. “Nonblocking and Orphan-Free M Message Logging Protocols”. In Proc. 23th IEEE Symposium on Fault Tolerant Computing, Toulouse (France), June 1993.

[66] E. N. Elnozahy. Manetho: “Fault Tolerance in Distributed Systems Using Rollback-Recovery and Process Replication”. PhD thesis, Rice University (USA), October 1993.

[67] M. Fetré. “ Intégration d’un mécanisme de reprise d’applications parallèles dans un système d’exploitation “. Rapport de stage Institut de Recherche en Informatique et Systèmes Aléatoires.

[68] K. Al Agha, G. Pujolle et G. Vivier. "Réseaux de mobiles & réseaux sans fil". 2001.

[68] A. M. Noll. “Introduction to Telephones and Telephone System“. Third edition Artech House Telecommunications Library. 1998.

[70] H. Chen, L .Huang, S. Kumar and C. C. J. Kuo. “Radio resources management for Multimedia QoS Support in Wireless Networks”. Kluwer Academic Publishers, 2004.

[71] G. H. Forman, J. Zahorjan, “The Challenges of Mobile Computing”. pp. 38-47, April 1994.

[72] D. Conan, S. Chabridon, O. Villin, and G.Bernard. “Domint: A Platform for Weak Connectivity and Disconnected CORBA Objects on Hand-Held Device”. Tecknical Report, INT, March 2003.

[73] A. Acharya, B. R. Badrinath. “Checkpointing Distributed applications on Mobil Computers”. Proc. Third Int'l Conf. Parallel and Distributed information Systems, September 1994.

[74] P. Krishna, N. H. Vaidya and D. K. Pradhan. “Recovery in distributed mobile environnement”. IEEE Workshop on advances in Parallel and distributed systems. October 1993.

[75] R. Prakash, M. Singhal. “Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems”. IEEE Trans. Parallel and Distributed Systems, pp. 1035-1048, October 1996.

[76] G. Cao, M. Singhal. “Mutable Checkpoints: A New checkpointing Approach for Mobile Computing Systems”. IEEE Trans. Parallel and Distributed Systems, pp. 157-172, February 2001.

[77] E. Pitoura, B. Bhargava. “Building Information Systems for Mobile environments”. In Third ACM International Conference on Information and Knowledge Management, pp, 371_378, Gaithersburg, Maryland, USA, November 1994.

[78] L. Temal. “Gestion des déconnexions et tolérance aux faute “. Rapport de Stage, DEA Méthodes Informatiques des Systèmes Industriels (M.I.S.I), laboratoire Systèmes répartis du département Informatique de l’Institut National des Télécommunications, Septembre 2003

[79] J. Jing, A. Helal, and A. Elmagarmid. “Client-Server Computing in Mobile Environments”. ACM Computing Surveys, 31(2), June 1999.

[80] D. B. Jhonson, S. W. Smith and J. D. Tygar. “ Completely asynchronous optimistic recovery with minimal rollbacks”. In proceeding of the 25th International Symposium on Fault Tolerance Computing Systems. pp. 361-370. 1987

[81] O. P. Damani, V. K. garg. “A low to recover efficiently and asynchronously when optimism fails ». In proceeding of the 16th International conference:Distributed Computing Systems. pp. 108-115, 1996.

[82] G. Cao, M. Singhal. “On the impossibility of min-process non-blocking checkpointing and an efficient checkpointing algorithm for mobile computing systems”. In proceeding of the 27th I conf. On Parallel processing, pp. 37-44, August 1998.

[83] G. Cao, M. Singhal. “Low cost checkpointing with mutable checkpoint in Mobile Computing Systems” In proceeding of the 18th International conference:Distributed Computing Systems. pp. 464-471, May 1998.

[84] P. Kumar, L. Kumar, R. K. Chauhan and V. K. Gupta.“A non intrusive minimum process synchronous checkpointing protocol for mobile computing systems“. Personnel wireless communications, IEEEE International conference. Pp. 23-25. January 2005.

[85] G. Li, L. Shu.” A low latency checkpointing for mobile computing systems”. Computer software and application conference, COMPSAC, 29th annual International. July 2005.

[86] C. Y. Lin, S. C. Wang “an efficient time based checkpointing protocol for mobile computing over mobile IP”. Mobil network and applications 8.pp. 687-697. 2003.

[87] P. Krishna, N. H. Vaidya and D. K. Pradhan. “Recoverable mobile environnement : Design and trade off analysis”. In proceeding of the 26th International Symposium on Fault Tolerance Computing . Sendai, Japan,pp. 16-25. June 1996.

[88] T.Park, N. Woo and Y. Yeom. “An efficient recovery scheme for fault tolerant mobile computing systems”. Future Generation computer systems. Vol 19, n° 1. pp. 37-53. December 2002.

[89] T. Tantikul, D. A. Manivannan.” Communication induced checkpointing and asynchronous recovery protocol for mobile computing systems”. Parallel and distributed computing, applications and technologies, 6th International conference. Pp. 05-08. December 2005.

[90] G. Cao, M. Singhal. “On Coordinated Checkpointing in Distributed Systems”. IEEE Trans. Parallel and Distributed System pp. 1213-1225, December 1998.

[91] N. Neves, W. K. Fuchs. “Adaptative recovery for mobile environnements”. Communication of the ACM, vol 40, n° 1,pp 68-74, January 1997.

[92] H. Hirogaki, M. Takiwaza. “Checkpoint recovery protocol for reliable mobile systems ”. In proceeding of the 7th IEEE Symposium on Reliable Distributed Systems.pp. 93-99. October 1998.

[93] C. M. Lin, C. R. Dow. “Efficient checkpointing based failure recovery techniques in mobile computing systems”. Journal of information science and engineering. pp. 549-573. May 2001.

[94] C. L. Yin, S. C. Wang, S. Yen. Kuo and I. Y. Chen. “ A low overhead checkpointing protocol for mobile computing systems”. Dependable computing proceedings. Pacific Rim International symposium. pp. 16-18. December 2002.

Résumé

Le travail présenté dans ce mémoire consiste à étudier la gestion des points de reprise (checkpoints) dans les applications réparties s'exécutants sur des réseaux fixes ou des réseaux mobiles.

L'étude des protocoles d'établissement de points de reprise dédiés à l'environnement réparti, nous a permis de classer ces protocoles selon des critères qui peuvent jouer sur la performance du système, voire la quantité du travail à refaire, le nombre de points de reprise à stocker, le nombre de site impliqués dans cette application et invoqués dans la procédure de reprise.

Tandis que l'étude faite sur l'environnement mobile dévoile des points cruciaux à prendre en considération lors de l'établissement des points de reprise. En effet, les caractéristiques du support mobile (largeur de la bande passante limitée, capacité da stockage restreinte, faible source d'énergie…) remettent en cause l'étude précédente et conclue que les protocoles dédiés aux réseaux fixes deviennent inadaptés une fois portés sur l'environnement mobile.

Mots clés : Applications distribuées, environnement mobile, tolérance aux fautes, Protocoles de checkpointing.

ملخص

العمل المدرج في هذه المذكرة, يخص تسيير نقاط الاستعادة في التطبيقات الموزعة و التي تنفذ في شبكات الاتصال الثابتة و النقالة.

دراسة بروتوكولات الاستعادة المخصصة للمحيط الموزع, مكنتنا من تصنيف هذه الأخيرة حسب خصائص تدخل في نجاعة النظام نذكر على سبيل المثال : كمية العمل الذي يجب إعادته, عدد نقاط الاستعادة التي يجب حفظها, عدد المواقع التي تساهم في عملية الاستعادة.

بخلاف الدراسة المخصصة للشبكات النقالة, التي نزعت القناع على نقاط هامة يجب أخذها بعين الاعتبار عند تشكيل بروتوكولات الاستعادة. خصائص الهيكل النقال (صغر عرض الشريط النافذ, قدرة تخزين نقط الاستعادة محدودة, مصدر طاقة ضعيف...), تعيد النظر في نتائج الدراسة السابقة وتجزم عدم نجاعة و لياقة بروتوكولات الاستعادة, عند نقلها من الثابت نحو النقال .

كلمات مفتاحيه: التطبيقات الموزعة, المحيط النقال, تجاهل الأخطاء, بروتوكولات الاستعادة.

Abstract

This work, consist of study the management of checkpoints, in distributed applications executed on fixed or mobile networks.

The study of check pointing protocols, dedicated for distributed applications, allow us to classify this protocols according to some facts, such as: the quantity of redo works, the number of checkpoints that we have to store, the number of hosts, implicated to participate to the recovery.

Mobile computing raises many new issues, such as: lack of stable storage, low bandwidth of wireless channel, high mobility and limited battery life…These new issues make traditional check pointing algorithms unsuitable for mobile computing.

Key words: distributed applications, mobile computing, fault tolerance, check pointing protocols.