UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

ALGORITHMES DE COMMUNICATION DANS LES RÉSEAUX EN PRÉSENCE DE PANNES BYZANTINES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

MICHEL PAQUETTE, ING.

SUPERVISEUR PROFESSEUR ANDRZEJ PELC

NOVEMBRE 2004

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Ce mémoire intitulé :

ALGORITHMES DE COMMUNICATION DANS LES RÉSEAUX EN PRÉSENCE DE PANNES BYZANTINES

présenté par
Michel Paquette, ing.

pour l'obtention du grade de maître ès science (M.Sc.)

a été évalué par un jury composé des personnes suivantes :

Andrzej Pelc ..... Directeur de recherche
Jurek Czyzowicz ..... Président du jury
Ilham Benyahia ..... Membre du jury

Mémoire accepté le : 3 novembre 2004

À ma conjointe Virginie.

À mon frère et à mes parents.

Résumé

La diffusion et l'échange d'information sont deux opérations primordiales pour les réseaux de communication et les systèmes distribués. Le fonctionnement correct de ces systèmes dépend de la fiabilité et de l'efficacité des opérations de transfert d'information. Alors, nos modèles d'analyse de ces systèmes doivent inclure la possibilité de panne des équipements impliqués dans ces processus. Ce mémoire répond à quelques questions fondamentales de tolérance aux pannes byzantines dans le cadre d'un modèle probabiliste de distribution de pannes. Nous proposons et validons d'abord un algorithme rapide, économique, et presque certain pour la diffusion dans un réseau de connexité logarithmique. Nous simulons ensuite cet algorithme et appuyons ainsi son fonctionnement. Finalement, nous présentons une stratégie de décision optimale pour la réception de messages diffusés dans des réseaux de connexité quelconque.

Mots clés : diffusion, échange d'information, pannes byzantines, modèle probabiliste, liens, noeuds, stratégie de vote.

Abstract

Broadcasting and gossiping are two essential operations for communication networks and distributed systems. The correct operation of these systems depends on the reliability and efficiency of information transfer functions. Thus, we must include the possibility of communication failures in our system analysis models. This thesis answers fundamental Byzantine fault tolerance questions in a probabilistic fault distribution model. We first propose and validate a fast, economical and almost safe broadcasting algorithm for use in a logarithmic-degree network. We then simulate this algorithm in order to further support its correctness. Finally, we present an optimal decision strategy for the reception of messages broadcasted in networks of arbitrary connectivity.

Keywords : broadcasting, gossiping, byzantine faults, probabilistic model, links, nodes, voting strategy.

Table des matières

Liste des figures

Liste des tableaux

Avant-propos

Je tiens à remercier Andrzej Pelc pour la précieuse collaboration offerte tout au cours de mes études de maîtrise.

J'aimerais tout particulièrement remercier ma conjointe Virginie pour le soutien moral, l'amour et l'aide qui m'ont permis de mener ce mémoire à terme.

Chapitre 1

Introduction et formulation du problème

Le consensus est un processus par lequel plusieurs entités se mettent en accord sur une action ou une information. Quoique simple à première vue, ce processus se complique dans les environnements à communication non fiable. Pour illustrer, il est simple de faire le consensus entre deux personnes en contact direct. Mais dès que le lien entre les deux partis consiste d'un ou plusieurs intermédiaires, le processus devient plus complexe. En effet, un intermédiaire pourrait brouiller les messages. Ce problème s'est présenté jadis avec ampleur lorsque l'Empire Byzantin était aux prises avec la corruption [16]. Lorsque différents généraux de l'Empire devaient coordonner leurs attaques, plusieurs d'entre eux, corrompus, brouillaient les messages à retransmettre aux généraux loyaux avec l'intention d'affecter la coordination des attaques. Le manque de coordination s'ensuivant causait la défaite par les corps loyaux de l'armée byzantine. Afin de résoudre le problème, il fallait trouver un moyen fiable de communiquer et interpréter les informations. Ce problème complexe, portant le nom de *consensus byzantin*, a dû être résolu par une méthode de communication impliquant un haut niveau de redondance des messages.

Aujourd'hui, ce même problème de communication existe dans différents contextes informatiques. Considérons un système dans lequel plusieurs processeurs communiquent pour faire le consensus sur un certain état. Dans un tel réseau, chaque composant peut tomber en panne. Mais peu importe la localisation et la gravité des pannes, l'ensemble des décisions prises par les bons processeurs doit être juste avec très grande probabilité [4]. Donc, nous voulons assurer la communication fiable entre deux partis fonctionnels dans un environnement où les chemins de communication peuvent être en panne. Il est

alors nécessaire d'implanter des algorithmes de communication résistants aux pannes. De plus, les systèmes d'information comportent, en général, un nombre toujours croissant de processeurs. Il importe donc de proposer des solutions conçues afin de ne pas pénaliser les réseaux de grande taille. Ceci est le premier problème étudié dans ce mémoire.

La diffusion de messages étant un ingrédient de base dans chaque processus de communication, nous élaborons un algorithme de diffusion fiable, rapide et économique qui utilise un nombre minimal de liens dans un réseau complet. Pour la communication, notre algorithme utilise un sous graphe, de connexité logarithmique, du réseau complet. Nous considérons uniquement le cas où les communications sont synchrones.

Au point de vue des investissements matériels et monétaires, les réseaux de connexité logarithmique sont relativement coûteux. Nous étudions donc aussi la communication fiable dans les réseaux dont la connexité est d'ordre inférieur. La seconde question considérée dans le cadre de ce mémoire est la suivante : trouver, pour un certain réseau de connexité k, la stratégie optimale de réception des messages si chaque noeud connaît la fiabilité de ses chemins entrants. Dans ce mémoire, nous étudions un sous problème fondamental de la question précédente : étant donné deux noeuds, S et D, liés par k chemins disjoints de fiabilité connue, trouver une stratégie optimale de réception des messages provenant à D à partir de S.

Comme domaine d'application de notre recherche, nous trouvons, par exemple, les systèmes réseaux reliant les bases de données réparties des succursales bancaires [18]. En fait, un grand nombre de systèmes nécessitant des transmissions d'information fiables dans un réseau sujet à des pannes matérielles ou logicielles fait partie du domaine d'application de notre mémoire. Notons que tous les réseaux existants sont sujets à des pannes matérielles.

Lorsqu'un noeud ou un lien est affecté par une panne byzantine, son comportement devient imprévisible [17]: il est possible que ce composant transmette l'information correctement, mais il peut aussi omettre des transmissions ou même transmettre des messages erronés. En général, on considère qu'un composant atteint d'une panne byzantine agit en pire cas; il est un adversaire à la transmission correcte des messages. Il s'ensuit que si un algorithme de communication est tolérant aux pannes byzantines permanentes, il est aussi capable de contrer tout autre type de pannes. L'avantage du scénario des pannes byzantines considéré dans ce mémoire est donc sa généralité: nos algorithmes pourront aussi être utilisés pour chaque autre type de pannes qui peut affecter les systèmes informatiques.

1.1 Composition du mémoire

Le mémoire est composé comme suit.

Le chapitre 2 contient la revue de la littérature concernant la communication en présence des pannes.

Le chapitre 3 résume les résultats obtenus au cours de nos recherches.

Le chapitre 4 porte sur la création d'un algorithme presque certain de diffusion rapide dans un réseau de notre construction.

Le chapitre 5 porte sur les simulations de l'algorithme développé au chapitre 4. Ces dernières furent effectuées à l'aide d'un algorithme codé dans le langage de programmation C++.

Le chapitre 6 traite du développement d'une stratégie de vote optimale sur les messages entrants dans un noeud par k chemins disjoints de fiabilités connues localement.

Le mémoire se conclue avec un résumé des points importants mis en évidence au cours des recherches, ainsi que différentes questions proposées pour des travaux futurs.

Chapitre 2

État de l'art

2.1 Introduction et définitions

Dans les réseaux de communication, les deux opérations de communication les plus importantes sont la diffusion et l'échange d'information. La diffusion est définie [17] comme la transmission d'un message contenu dans un noeud appelé *source* vers tous les autres noeuds du réseau. À la fin d'une diffusion correcte, tous les noeuds (fonctionnels) d'un réseau doivent connaître le message de la source. L'échange d'information est défini [17] comme la diffusion d'un message à partir de tous les noeuds vers tous les autres noeuds du réseau. À la fin de l'échange d'information correct, tous les noeuds (fonctionnels) du réseau doivent connaître les messages de chaque autre noeud (fonctionnel) du réseau. Dans certains problèmes, on considère que même les noeuds qui sont en panne doivent recevoir le(s) message(s).

Des algorithmes de transmission de messages sans tolérance aux pannes ont été développés pour tous les types de réseaux. Par contre, les algorithmes de transmission permettant la diffusion correcte des informations dans les mêmes réseaux en présence de pannes sont plus complexes et moins connus. Généralement, deux types de pannes sont considérées: les pannes arrêt et les pannes byzantines.

Deux modèles décrivent les noeuds en panne arrêt [9]. Ces deux modèles ont un point commun : un noeud *arrêté* ne peut émettre aucun message. Par contre, un modèle dicte qu'un noeud arrêté peut recevoir un message tandis que l'autre dicte le contraire. Pour effectuer une opération de communication correcte selon le premier modèle, les noeuds arrêtés doivent recevoir le message; ceci ne peut être exigé selon le second modèle. Pour

ces modèles de pannes, si seuls les noeuds sont affectés par les pannes, il suffit d'envoyer le message avec succès une fois à chaque noeud pour accomplir une diffusion correcte. Dès qu'on doit transmettre les messages à l'aide d'intermédiaires, la solution augmente en complexité.

Une panne arrêt sur un lien cause la perte de tous les messages envoyés par ce dernier. Alors, lorsqu'on tente de diffuser un message en présence de pannes arrêt permanentes, il faut généralement envoyer les messages à chaque noeud par plusieurs chemins disjoints. On envoie plutôt un message à un processeur plusieurs fois sur un même lien si les pannes sont transitoires.

Les pannes byzantines sont caractérisées par un comportement arbitraire et même malicieux [1]. Un noeud ou un lien affecté par une panne de ce type peut imiter un élément sans panne en se comportant correctement. Il peut aussi modifier les messages pour tromper le système, générer des messages erronés, ou omettre des transmissions. Les pannes byzantines sont les plus difficiles à contrer à cause de leur généralité et de leur caractère imprévisible. En général, on peut contrer ces pannes en appliquant une stratégie de vote sur un ensemble de plusieurs copies du message à recevoir. Nous utilisons les stratégies de vote aux chapitres 4 et 6.

L'apparition des pannes dans un réseau peut suivre différents modèles [17]. Dans le modèle probabiliste, les pannes apparaissent avec des probabilités $0 \leq p,q < 1$ sur les liens et sur les noeuds respectivement. Dans ce modèle on considère souvent les algorithmes ϵ-certains (ϵ-*safe*). Un algorithme ϵ-certain résout le problème de diffusion dans un réseau de n noeuds - le message se rend partout où il le doit - avec une probabilité de $1-n^{-\epsilon}$, avec $\epsilon \geq 1$. Donc la fiabilité d'un algorithme ϵ-certain converge vers l'unité avec l'augmentation de la taille du réseau. Dans ce mémoire, nous présentons un algorithme qui satisfait cette condition.

Dans un modèle de pannes bornées, le nombre de pannes dans un réseau est borné par un entier k [17]. Un algorithme qui résout correctement un problème de communication dans un système suivant ce modèle est dit k-tolérant. L'algorithme est correct si et seulement si les informations se rendent à tous les noeuds (fonctionnels) du système, pour n'importe quelle configuration d'au plus k pannes.

Les pannes sont dites transitoires si elles ne durent qu'un certain temps; elles sont dites permanentes si elles ne se rétablissent pas avant la fin d'une exécution. Un modèle intéressant sera vu à la section 2.2.2 : un modèle avec un nombre de défaillances transitoires borné linéairement dans le temps.

La recherche en référence est consacrée strictement à des problèmes de communication synchrones, c'est-à-dire des communications dans lesquelles les émetteurs et récepteurs envoient et reçoivent des messages à des temps prédéterminés selon une horloge globale. Un algorithme asynchrone est caractérisé par une absence de synchronisation entre les processus; les événements se produisent alors à des temps imprévisibles.

Afin de mieux modéliser les systèmes d'information, deux modes de transmission de messages sont discutés: *shouting (ou ports multiples)* et *whispering (ou port unique)*. Dans le mode de communication *ports multiples*, un noeud peut communiquer sur tous ses ports à chaque unité de temps. Par exemple, un processeur sur un réseau radio envoie ses messages à tous les processeurs qui sont en portée d'écoute. En mode *port unique* un noeud peut communiquer sur un seul lien à chaque unité de temps; il ne peut utiliser plus d'un port à la fois. Par exemple, un processeur qui aurait un lien vers un démultiplexeur qui, ensuite aurait dix liens vers dix processeurs différents pourrait être modélisé par un noeud de 10 ports en mode port unique.

Une distinction supplémentaire par rapport aux liens de communication est le mode bidirectionnel (*full-duplex*) vs le mode unidirectionnel (*half-duplex*). Un lien bidirectionnel permet aux noeuds d'envoyer et de recevoir des messages simultanément sur ce dernier. Le lien unidirectionnel ne permet pas aux processeurs d'envoyer et de recevoir des messages simultanément. Par exemple, les processeurs sur des réseaux *bus* ne peuvent qu'envoyer ou recevoir à tout moment tandis que la plupart des systèmes de communications point-à-point modernes permettent l'envoi et la réception simultanés. Le modèle le plus étudié dans les articles passés en revue est celui des communications *bidirectionnel port unique*. Nous utilisons aussi ces modes dans le présent mémoire afin de modéliser des systèmes modernes et économiques: des systèmes avec seulement les équipements essentiels.

La performance d'un algorithme de communication dans les réseaux est principalement évaluée en fonction de deux mesures : le nombre de messages que l'algorithme doit utiliser (nombre d'appels) et le temps nécessaire pour compléter son exécution. Nous désirons obtenir un ordre de grandeur minimal pour ces deux mesures. Un autre paramètre de performance, souvent implicite, est la connexité minimale du réseau qui est requise pour exécuter l'algorithme de transmission avec succès. Plus cette connexité minimale est faible, plus l'algorithme est portable [3].

Les algorithmes de communication sont séparés en deux catégories par rapport à leur fonctionnement: les algorithmes adaptatifs et les algorithmes non adaptatifs. Les algorithmes adaptatifs choisissent les actions à prendre au cours de l'exécution en fonction

des connaissances localement accumulées. Les algorithmes non adaptatifs dictent toutes les actions à prendre avant le début de l'exécution. Pour un scénario donné, on peut souvent trouver un algorithme adaptatif plus rapide ou plus économique pour chaque algorithme non adaptatif. Par contre, les algorithmes adaptatifs exigent en général une infrastructure de réseau plus complexe. Nous développons un algorithme non adaptatif au chapitre 4.

Dans les sections suivantes nous explorons certaines solutions employées pour différents problèmes de diffusion dans les réseaux contenant des pannes. Les sections suivantes sont organisées comme suit: la section 2.2 porte sur les scénarios de pannes arrêt sur les liens, la section 2.3 introduit le problème des pannes byzantines sur les liens, la section 2.4 traite de problèmes avec des noeuds en panne arrêt, la section 2.5 introduit le problème réaliste des pannes arrêt sur les noeuds et sur les liens, la section 2.6 traite des stratégies de vote optimales. Le chapitre se conclura avec un résumé de quelques problèmes auparavant ouverts, traités dans ce mémoire.

2.2 Pannes arrêt des liens

Les pannes arrêt permanentes sur des liens peuvent correctement modéliser certains bris matériels des liens de communication. Les pannes arrêt transitoires modélisent bien les débordements de tampons. Aucun lien de communication des réseaux connus n'est immunisé aux bris matériels. Ce type de panne est donc pertinent pour modéliser les systèmes de communications existants.

2.2.1 Nombre borné de pannes permanentes

Lorsque des pannes arrêt affectent seulement les liens, on peut effectuer une diffusion certaine d'information si le nombre de pannes est borné par un entier inférieur à la connexité d'un réseau [8, 17]; si le nombre de pannes est supérieur à la connexité, il est possible de couper le réseau en deux parties disjointes en plaçant les pannes aux endroits appropriés, rendant ainsi toute communication impossible entre ces deux portions du réseau. Un algorithme qui effectue correctement une diffusion en présence d'un maximum de k pannes dans un réseau est dit k-tolérant. La diffusion est correcte lorsque tous les noeuds du réseau ont reçu ce message. Les analyses doivent toujours être faites en pire cas, c'est-à-dire avec les pannes localisées pour affecter le réseau de la pire manière possible.

Dans [8], les auteurs traitent d'un hypercube de dimension n affecté d'un nombre de pannes des liens borné par k, $0 \leq k < n-1$. La première phase de l'algorithme proposé consiste en l'application d'un algorithme zéro-tolérant constitué d'appels séquentiels suivant les dimensions $d = 0,1,...,m-1$ de l'hypercube. Pour la seconde phase, les noeuds procèdent en parallèle à $k+1$ appels supplémentaires sur les dimensions $d = 0,1,...,k$ de l'hypercube. Ces appels redondants acheminent l'information aux noeuds qui ne la possédaient pas par au moins k chemins différents de celui prévu dans l'algorithme zéro-tolérant. Puisque les appels se font toujours en phases qui correspondent aux m dimensions de l'hypercube, un noeud doit appeler seulement son voisin de la dimension $n \ mod \ m$ où n est la ronde courante de l'algorithme. Le temps d'exécution total de l'algorithme de diffusion pour l'hypercube est donc $\log n + k + 1$. Ce résultat est d'ordre optimal puisque dans [12], on prouve la borne inférieure sur le temps de diffusion $T \geq \lceil \log n \rceil + k$.

Le modèle des pannes bornées, utilisé dans cet article, n'est pas le modèle de pannes le plus réaliste [17]. En réalité, tous les liens et tous les noeuds d'un système de communication ont une probabilité de panne non nulle. Donc plus un réseau est grand, plus le nombre de pannes doit être important, d'où le modèle des pannes bornées linéairement. Ce modèle sera revu dans la section **??**.

2.2.2 Nombre de pannes transitoires borné linéairement dans le temps

Dans les réseaux réels, les liens de communication ne sont pas immunisés aux erreurs de transmission. On spécifie souvent la qualité d'un lien par le taux d'erreur sur les données causé par ce dernier. Le modèle des pannes arrêt transitoires des liens dont le nombre est borné linéairement dans le temps est considéré dans [11]. Dans cet article, on considère le mode de communication port unique.

On qualifie d'α-tolérant un algorithme qui peut effectuer correctement la diffusion de l'information dans le réseau malgré un nombre $\alpha \cdot t$ de pannes transitoires, où t est le nombre d'unités de temps écoulées et $0 < \alpha < 1$. Ces pannes doivent être considérées en pire cas, comme dans tous les modèles de pannes bornées. Le premier algorithme présenté dans [11] pour le cas du graphe en ligne permet la transmission α-tolérante pour $\alpha < 1/2$ (moins d'une faute par deux tours). Les liens portent des identificateurs comme suit: le lien le plus à gauche porte l'identificateur 0, puis les liens à sa droite suivent la séquence des nombres entiers positifs. L'algorithme consiste d'appels en alternance sur liens pairs d'abord et impairs ensuite. Cet algorithme ne peut fonctionner dès qu'$\alpha \geq 1/2$ puisque

alors on peut placer une panne sur le lien 1 à chaque tour impair, empêchant ainsi la diffusion. Un second algorithme est alors présenté pour résoudre le cas où $1/2 \leq \alpha < 1$. Cet algorithme effectue un nombre croissant d'appels consécutifs sur les liens pairs et ensuite impairs. De cette manière, on ne peut placer les pannes en pire cas et bloquer la progression du message dans la ligne. Les auteurs démontrent que grâce à la plus grande connexité de l'anneau, la diffusion peut se faire dans ce graphe avec le premier algorithme lorsque $\alpha \leq 1$.

Pour prouver l'exactitude et calculer la performance de cet algorithme, les auteurs de [11] emploient l'argument de l'adversaire. Un adversaire place toujours les pannes en pire cas, selon un nombre de pannes qu'il peut dépenser. L'adversaire détient une bourse de pannes qui ne peut devenir négative en aucun temps. Cette bourse se vide d'une quantité α^{-1} à chaque panne utilisée et se remplit d'une unité à chaque unité de temps. Cet argument peut être utilisé pour tous les algorithmes de diffusion en présence de pannes transitoires bornées linéairement dans le temps.

Le modèle des pannes transitoires linéairement bornées dans le temps apporte un élément supplémentaire de réalité dans la modélisation des réseaux. Par contre, rien en réalité n'impose une borne sur le nombre de pannes. C'est pourquoi les modèles probabilistes ont été utilisés dans le cadre de maintes recherches. C'est aussi pourquoi le présent mémoire traitera de problèmes dans ce cadre.

2.2.3 Modèle probabiliste

Dans [3], les auteurs considèrent un réseau de connexité logarithmique dont les communications s'effectuent en mode bidirectionnel port unique. Chaque lien de ce réseau a une probabilité positive d'être affecté par une panne arrêt. L'algorithme adaptatif présenté dans cet article peut effectuer la diffusion à travers ce réseau. L'algorithme décrit (*ABA*: Call Saving Adaptative Broadcasting Algorithm*) peut effectuer une diffusion en pire cas en temps dans $O(\log n)$ avec un nombre de messages dans $O(n)$.

Pour effectuer cette diffusion, on encadre les noeuds dans une structure semblable à celle d'un arbre binaire. En fait, des groupes de noeuds de taille dans $O(\log n)$ sont disposés dans une structure d'arbre binaire complet. Ces groupes de noeuds ne comportent pas de liens internes mais seulement des liens avec leurs groupes voisins dans l'arbre binaire; chaque paire de groupes voisins forme un graphe biparti complet. La source est un noeud qui fait partie du groupe à la racine de l'arbre. Dans une première phase, l'algorithme transmet l'information à un noeud par groupe avec grande probabilité; ce

noeud est appelé *chef*. Dans la seconde phase, chaque chef tente de transmettre l'information vers tous les noeuds de son groupe parent dans l'arbre binaire. Finalement, tous les noeuds qui n'ont pas reçu l'information appellent des noeuds dans leur groupe parent pour l'obtenir. Le processus est complété avec souci de ne pas faire participer un noeud à plus d'un appel à la fois. Une structure similaire est utilisée dans le présent mémoire.

Il est intéressant de noter qu'à la dernière étape de cet algorithme, les noeuds qui ignorent le message de la source font les appels, d'où l'aspect adaptatif. L'hypothèse est faite ici que les noeuds ignorant le message savent qu'ils auraient dû le recevoir. Cet algorithme est donc applicable à des situations où des transmissions sont effectuées selon un certain horaire connu des noeuds. Si on voulait effectuer une telle transmission selon un horaire inconnu des noeuds, il faudrait changer d'approche et augmenter le nombre d'appels nécessaires pour compléter l'algorithme.

2.3 Pannes byzantines des liens

Les composants atteints de pannes byzantines peuvent corrompre les informations communiquées dans les réseaux; on doit les considérer comme des adversaires. Étant donné la nature trompeuse de ces pannes, il est impossible d'effectuer les opérations de diffusion et d'échange d'information dans les réseaux de communication en appliquant les mêmes techniques que pour les pannes arrêt. Un modèle de pannes byzantines transitoires des liens représente bien l'influence des champs électromagnétiques sur les liens de communication.

Tel que vu dans [1], dans le cas des pannes byzantines, il est nécessaire de transmettre un message indirectement à travers un grand nombre de chemins disjoints afin d'assurer avec grande probabilité que la majorité des messages arrivés ($\lfloor m/2 \rfloor + 1$, m le nombre total de messages) à destination soient corrects. Alors, aucun algorithme de transmission pour réseaux affectés par des pannes byzantines ne peut fonctionner pour une probabilité de pannes de tous les chemins $p \geq 1/2$, puisque dans ce cas la majorité des messages seront faux avec forte probabilité. Notons que la probabilité de panne d'un chemin est la probabilité qu'au moins un noeud ou un lien de ce chemin soit en panne. Pour les pannes arrêt, un seul message arrivé à destination suffisait puisque les messages ne pouvaient pas être corrompus.

Dans [1], on décrit un algorithme de transmission fonctionnant en temps d'ordre optimal dans $O(\log n)$ et qui permet la diffusion presque certaine dans un réseau affecté par des pannes byzantines des liens. Cet algorithme fonctionne pour un réseau de

connexité au moins logarithmique avec des groupes logarithmiques de noeuds organisés dans une structure d'arbre binaire; cette organisation a aussi été utilisée dans [3]. La connexité logarithmique est nécessaire pour assurer la redondance du message permettant une transmission presque certaine. L'algorithme proposé dans cet article fonctionne en phases: d'abord la source envoie son message à tous les noeuds de son groupe appelé $ROOT$; ensuite, pour chaque niveau de l'arbre à tour de rôle, chaque noeud de chaque groupe envoie son message à son voisin de chaque groupe enfant; finalement, le message est communiqué à l'intérieur de chaque groupe, puis voté en parallèle pour chaque noeud. Chaque groupe constitue un sous graphe complet du grand graphe et les liens entre les groupes forment des couplages.

2.4 Pannes arrêt des noeuds

2.4.1 Nombre de pannes borné linéairement dans le nombre de noeuds

Ce modèle de pannes est plus réaliste que les modèles où le nombre de pannes est borné par une constante. En effet, il est naturel qu'avec l'augmentation du nombre de composants dans un système, le nombre de pannes augmente aussi.

Lorsque les noeuds sont affectés par des pannes, les solutions ne sont pas toujours les mêmes que lorsque les liens sont en panne. Pour les pannes arrêt [9], on cherche à transmettre l'information à tous les noeuds (fonctionnels). L'algorithme k-tolérant (avec $k = \beta \cdot n$,$0 < \beta < 1$) proposé dans cet article se déroule tel que décrit dans le paragraphe suivant.

Tout d'abord, une liste est formée pour utilisation ultérieure. Cette liste doublement ordonnée permet ensuite de faire la diffusion en temps dans $O(\log n)$ par une technique de sauts de pointeurs. L'algorithme de diffusion est ensuite appliqué sur cette liste. L'organisation de la diffusion se fait suivant la structure d'un arbre binaire. Cet algorithme est adaptatif puisque les appels sont déterminés au cours de l'algorithme même, lors de la formation de la liste de transmission. Le temps de fonctionnement de l'algorithme est $\in O(\log^2(n))$. Ensuite, un algorithme plus rapide travaillant en temps $O(\log n)$ a été proposé dans [5] pour ce modèle de pannes.

2.4.2 Pannes aléatoires sur les noeuds

Le modèle où les pannes se produisent aléatoirement seulement sur les noeuds n'a pas été exploré. Par contre, dans la prochaine section, nous explorons un modèle plus général et plus réaliste, celui des pannes arrêt distribuées aléatoirement sur les noeuds et les liens.

2.5 Pannes arrêt distribuées aléatoirement sur les noeuds et les liens

Tout composant d'un système peut tomber en panne avec une probabilité positive. Nous devons donc explorer le modèle des pannes aléatoirement distribuées sur les noeuds et les liens d'un réseau.

Ce modèle est présenté en [2], pour une application à un réseau à structure d'hypercube. On considère l'hypercube de dimension $m = \log n$ ($2^{\log n} = n$ noeuds). L'algorithme de diffusion proposé rappelle qu'il est nécessaire d'envoyer le message à travers plusieurs chemins disjoints pour pouvoir affirmer, avec grande probabilité, que tous les bons noeuds ont reçu ce message [1, 3].

La difficulté majeure dans l'élaboration de l'algorithme provient du fait qu'un noeud peut participer à un seul appel à la fois; certaines contraintes s'ensuivent. L'algorithme est exécuté en 4 phases: la phase 1 consiste à transmettre l'information au tiers des voisins directs de la source (voisins directs des dimensions $[2k/3,k]$). En phase 2, ces derniers transmettent à leurs voisins directs des dimensions $[0,k/3]$. À la phase 3, les récepteurs de la phase précédente envoient le message à leurs voisins directs des dimensions $[2k/3,k]$; ces derniers sont les voisins directs de la source dans les dimensions $[0,k/3]$. Ces noeuds reçoivent le message avec grande probabilité. À la phase finale les noeuds ne connaissant pas le message l'obtiennent des noeuds voisins de la source en dimensions $[0,k/3]$. Avec autant de transmissions parallèles, il faut prendre soin de ne pas faire participer un noeud à deux appels simultanés. L'algorithme est presque certain et fonctionne en temps logarithmique (dans $O(\log n)$) en pire cas.

2.6 Stratégies de vote optimales

L'exclusion mutuelle est une opération essentielle du partage des ressources dans les systèmes distribués. Si les noeuds et/ou les liens d'un système tombent en panne de façon aléatoire, l'exclusion mutuelle doit encore fonctionner. Alors, peu importe l'action exclusive qu'un segment de réseau permette à un processeur, il ne faut pas qu'un autre segment de réseau attribue le même droit à un autre processeur.

Pour un certain U (l'ensemble des noeuds faisant partie d'un système réparti) une fonction de répartition des votes v attribue à chaque noeud $u \in U$ un nombre positif et entier de votes $v(u)$. Un sous graphe de noeuds fonctionnels obtient la majorité s'il détient plus que la moitié de tous les votes du système. Le total des votes $T = \sum_{u \in U} v(u)$, donc la majorité $MAJ(v) = \lfloor \frac{T}{2} + 1 \rfloor$. On dit que le système est en halte à un quelconque moment s'il ne peut pas attribuer une ressource partagée. Ceci peut se produire si aucun des groupes de noeuds fonctionnels connexes ne détient la majorité. Une répartition de vote optimale est celle dont la probabilité de halte du système est minimale.

Pour résoudre ce problème d'optimisation, Garcia-Molina et Barbara ont introduit le concept de *coteries* [6, 7] comme suit: Soit U l'ensemble des noeuds composant un système. Un ensemble S de sous-ensembles de U est une *coterie* sous U si et seulement si

1. $G \in S$ implique $G \neq \{\}$, $G \subseteq U$.
2. Si G,$H \in S$, alors G et H doivent avoir au moins un noeud en commun.
3. Il n'y a pas de G,$H \in S$ tel que $G \subset H$.

Les éléments d'une coterie s'appellent *quorums*. Les propriétés des coteries garantissent que les actions permises par un quorum préservent l'exclusivité.

Pour chaque fonction de répartition de votes, l'ensemble des ensembles minimaux détenant la majorité des votes du système forme une coterie. La qualité d'une coterie est mesurée par la probabilité que l'ensemble des noeuds fonctionnels possède un sous graphe connexe contenant un élément de la coterie. Une coterie optimale sous V maximise cette probabilité parmi toutes les coteries sous U.

Garcia-Molina et Barbara ont proposé un algorithme d'énumération pour trouver la coterie optimale pour un ensemble de noeuds U et une répartition de votes v [6, 7]. Puisque le nombre de coteries à générer pour toutes les comparer est $\in \Omega(2^{2^{cn}})$ avec c une constante, l'approche d'énumération proposée n'est pas utilisable pour tout grand ensemble U. Une solution de rechange donnant des résultats suboptimaux est donc

proposée.

Dans [20], Tong et Kain ont démontré que le nombre d'éléments faisant partie d'une *coterie* optimale augmente exponentiellement selon N pour un réseau complètement connexe. Ce constat implique que l'utilisation de coteries est impossible pour les grands réseaux. Les auteurs proposent donc un algorithme pour trouver une bonne stratégie de vote (quoique suboptimale) travaillant en temps linéaire.

Soit un ensemble de noeuds $\overline{X} = U - X$ tel que U est l'ensemble de tous les noeuds d'un réseau. Tong et Kain [20] prouvent la suffisance des conditions suivantes pour l'obtention de coteries S optimales:

1. Si $S_i \in S$ alors $P(S_i) \geq P(\overline{S_i})$
2. $S_i \in S$ ou $\overline{S_i} \in S$, pour chaque S_i satisfaisant les conditions de cette coterie.

Tong et Kain proposent un algorithme générant une répartition de votes optimale avec un temps d'exécution $O(2^n), n = |U|$. Puisque pour de grands n, cet algorithme n'est pas utilisable, les auteurs présentent ensuite un autre algorithme fournissant une répartition de votes suboptimale et fonctionnant en temps linéaire.

Spasojevic et Berman utilisent les résultats de Tong et Kain dans [18] en modifiant l'algorithme de génération de répartition optimale de votes pour qu'il devienne dans $O(n)$. Ils obtiennent un algorithme générant des répartitions de votes proches d'optimales.

Le problème de trouver la stratégie de vote optimale permettant de faire le choix parmi les copies possiblement corrompues d'un message, reçues par un noeud à travers k chemins disjoints est semblable aux problèmes décrits ci-dessus. On connaît la fiabilité de ces k chemins et on cherche quelle stratégie de vote sur les messages reçus fournira une fiabilité optimale. Ce problème sera abordé au chapitre 6 du présent mémoire.

2.7 Conclusion

Les algorithmes de diffusion et d'échange d'information doivent être adaptés aux modèles de panne représentant le mieux leurs environnements d'utilisation. Dans ces contextes, certaines lignes directrices nous permettent de résoudre les problèmes qui nous sont posés.

Le problème suivant est proposé dans plusieurs articles [17] : Effectuer la conception d'algorithmes *rapides* de diffusion de données pour différents réseaux (complet, hyper-

cube, arbres binaires de groupes logarithmiques, etc.) atteints de pannes byzantines aux noeuds et aux liens avec probabilités respectives $q < 1/2$ et $p < 1/2$ (des restrictions supplémentaires devront être imposées par la suite). Ici, *rapide* reste un terme indéfini; il faudra découvrir la borne supérieure sur le temps de diffusion; l'efficacité en nombre de messages est un autre paramètre à considérer. Les pannes byzantines sont choisies à cause de leur généralité: les algorithmes de diffusion conçus pour ces pannes fonctionnent aussi bien pour n'importe quel autre type de pannes du système.

Ce problème important est le premier sujet du présent mémoire. Nous choisissons de traiter ce problème pour le réseau complet dans le contexte des communications synchrones. Comme nous le verrons plus tard, seulement une petite partie des liens du réseau complet sera utilisée par l'algorithme.

Les stratégies de vote optimales sont des outils importants pour la réception fiable des messages dans les réseaux de connexité arbitraire. Puisque les réseaux comportent rarement autant de liens que nécessitent les algorithmes de communication presque certains, il faut trouver un autre moyen de résister aux pannes, le meilleur moyen possible. Cette question est le second sujet de ce mémoire.

Le scénario de pannes traité est le même que pour le premier problème. Par contre, le contexte est différent. Le processeur récepteur connaît la fiabilité de chaque chemin à travers lesquels les messages lui sont expédiés. De plus, le nombre de chemins d'expédition des messages est insuffisant pour permettre la réception presque certaine. Nous devons alors assurer la fiabilité optimale de réception.

Chapitre 3

Présentation des résultats de la recherche

3.1 Algorithme de diffusion rapide en présence de pannes byzantines

Les résultats de cette portion du travail de recherche ont été publiés dans l'article de conférence suivant: Michel Paquette et Andrzej Pelc, *Fast Broadcasting with Byzantine Faults*, Proc. 7th IFAC Symposium on Cost Oriented Automation, Gatineau, Québec, Juin 2004, pp. 311-316 (référence [15]).

Nous considérons les algorithmes non adaptatifs dans les réseaux de n noeuds avec des pannes byzantines distribuées aléatoirement sur les liens et les noeuds; ces pannes sont indépendantes. Les communications sont synchrones, bidirectionnelles et se font en mode port unique. La probabilité de panne d'un lien est p et la probabilité de panne d'un noeud est q. Nous supposons $(1-p)^2(1-q) > \frac{1}{2}$.

Notre but principal est de construire un algorithme de diffusion dont la fiabilité est au moins $1-\frac{1}{n}$ pour un réseau de n noeuds. Un algorithme qui atteint cette fiabilité sera appelé *presque certain*. Notons que la probabilité qu'il fonctionne correctement converge vers 1 lorsque le nombre de noeuds augmente.

Afin de créer un algorithme de diffusion rapide et presque certain, nous avons utilisé une structure similaire à celle de l'arbre binaire. Ce type de structure permet d'effectuer en parallèle un grand nombre de transmissions à chaque ronde, permettant ainsi d'optimiser le temps de diffusion.

Afin de contrer les pannes arrêt, l'ordre du nombre de messages optimal pour les algorithmes non adaptatifs est dans $\Theta(n \log n)$ [17]. Étant donné la nature plus complexe des pannes byzantines, il était attendu que cet ordre du nombre de messages constitue une borne inférieure pour un algorithme non adaptatif traitant ce cas. Nous avons réussi à développer un algorithme aussi performant à ce niveau: un algorithme d'ordre optimal.

En effet, la borne de Chernoff[1] permet de calculer que la transmission d'un message à travers un nombre logarithmique de chemins disjoints est nécessaire pour permettre une diffusion presque certaine. Alors, puisque chaque noeud doit recevoir le message $c \cdot \log n$ fois (c est une constante dépendant des probabilités de panne p et q), le nombre de messages devait être dans $O(n \log n)$.

Pour le temps d'exécution de l'algorithme, le temps de diffusion le plus rapide possible dans un réseau sans panne est dans $O(\log n)$ en mode de communication port unique[2]. Dans ce cas, une seule transmission est nécessaire pour chaque noeud. Puisque un nombre logarithmique de transmissions est nécessaire pour chaque noeud afin d'assurer une transmission presque certaine, notre but était la construction d'un algorithme de diffusion fiable, travaillant en temps dans $O(\log^2 n)$. Dans le chapitre 4, nous exposons l'algorithme DRPB qui atteint ce but.

Du point de vue méthodologique, l'analyse de l'algorithme a été effectuée en utilisant des outils combinatoires. L'analyse de fiabilité de l'algorithme a été faite en utilisant des méthodes probabilistes se basant sur la borne de Chernoff.

3.2 Simulation de l'algorithme de diffusion rapide

Afin de simuler le fonctionnement de l'algorithme dans un réseau avec des pannes, nous avons initialement proposé de générer un réseau virtuel, tel que spécifié dans la recherche, et d'y distribuer des pannes de manière aléatoire. À l'étape de la simulation, les noeuds et liens en panne byzantine devaient jouer le rôle d'adversaires de l'algorithme.

Par contre, la simulation du réseau qui exécute l'algorithme a été implantée d'une manière différente de celle planifiée. Afin de produire des résultats en des temps raisonnables (permettant de les inclure dans ce mémoire), la modélisation du réseau exécutant l'algorithme a été simplifiée. Comme montré plus tard, ces simplifications n'affectent pas

1. Borne de Chernoff: Soit X le nombre de succès dans une série de Bernoulli de longueur m avec probabilité de succès q. Alors $Prob(X \leq (1-\epsilon)mq) \leq e^{-\epsilon^2 mq/2}$.

2. Initialement, seulement la source connaît le message. Puis, à chaque tour on peut au plus doubler le nombre de noeuds qui connaissent le message. $\sum_{i=0}^{(t-1)} 2^i = 2^t - 1 = n$. Alors $t \in O(\log n)$.

de manière négative les résultats de simulation.

Tel qu'attendu, l'algorithme fonctionne mieux dans les simulations que les prévisions des calculs analytiques. Cette attente provient du fait que l'outil utilisé pour estimer la fiabilité de l'algorithme, la borne de Chernoff, est une borne supérieure sur les probabilités. Le nombre d'essais effectués pour chaque réseau considéré a été assez grand pour obtenir des statistiques significatives.

3.3 Stratégie de vote optimale

Les résultats de cette portion du travail de recherche ont été publiés dans l'article de conférence suivant: Michel Paquette et Andrzej Pelc, *Optimal Decision Strategies in Byzantine Environments,* Proc. 11th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Smolenice Castle, Slowakia, Juin 2004, pp. 245-254 (Référence [14]).

Nous considérons un noeud dont la tâche est de déterminer avec la plus grande probabilité d'exactitude possible quel message lui a été envoyé par une source. Ce noeud reçoit le message à travers k chemins disjoints dont les fiabilités sont connues localement.

Pour les valeurs de k d'ordre inférieur au logarithmique, il serait impossible d'effectuer des diffusions presque certaines. Nous avons donc cherché à optimiser la probabilité d'exactitude des décisions prises sur la base des messages reçus; ceci est fait à travers une stratégie de vote optimale, c'est-à-dire une stratégie dont la fiabilité est la plus élevée parmi toutes les stratégies.

Étant donné que toutes les données pertinentes sont connues par les noeuds au moment du vote, il n'est pas nécessaire d'implanter une liste de décisions dans l'algorithme. Il suffit plutôt d'implanter une règle de calcul pour générer les décisions. L'utilisation d'une règle a comme avantage l'économie d'espace mémoire dans les processeurs.

Chapitre 4

Diffusion rapide en présence de pannes byzantines

SOMMAIRE

La communication en présence de pannes byzantines est un problème général auquel il faut trouver des solutions pour les réseaux de connexité la plus faible possible. Ce chapitre s'attaque à la transmission presque certaine dans un réseau pouvant comporter ces pannes sur les liens et sur les noeuds. Nous proposons un algorithme presque certain pour effectuer la diffusion dans un réseau de degré logarithmique de notre construction.

4.1 Introduction

La diffusion est une opération importante pour beaucoup de systèmes de communication. Il est important de garantir le fonctionnement de cette opération malgré les diverses pannes qui peuvent perturber le processus de transmission de messages.

En effet, les pannes matérielles et logicielles peuvent affecter tous les systèmes d'information. Que ces pannes se produisent sur les processeurs ou sur les liens de communications, il est impossible de prévoir leur nature. Alors, il est important de développer des algorithmes de communications tolérants à tous les cas de pannes possibles.

Le modèle des pannes byzantines est très général; une panne byzantine peut causer l'envoi de messages erronés, la corruption des messages, l'omission d'envois, ainsi que l'apparence d'un comportement normal. En fait, nous attribuons aux éléments atteints

de telles pannes un caractère malicieux; ils agissent en pire cas pour faire échouer les algorithmes de communication. En effet, on suppose que ces éléments connaissent les algorithmes de communication et ils peuvent même coordonner leurs attaques sur le système de communication. Nos algorithmes, conçus pour ce modèle de pannes le plus général, fonctionnent donc avec autant d'efficacité pour tout autre type de pannes de système.

La distribution des pannes est un facteur important dans la modélisation des réseaux. En effet, il est impossible de prévoir où une panne se produira. De plus, dans la majorité des cas, les pannes se produiront de manière indépendante. Le modèle de distribution aléatoire des pannes semble alors approprié pour modéliser une grande partie des réseaux réels, ce qui motive notre choix.

Les résultats de ce chapitre ont été publiés dans l'article de conférence suivant: Michel Paquette et Andrzej Pelc, *Fast Broadcasting with Byzantine Faults*, Proc. 7th IFAC Symposium on Cost Oriented Automation, Gatineau, Québec, Juin 2004, pp. 311-316 (Référence [15]).

Le restant du chapitre se divise ainsi: La section 4.2 détaille le modèle considéré dans la résolution du problème de diffusion. Ensuite, la section 4.3 expose la structure du réseau proposée pour effectuer la diffusion. Puis, la section 4.4 détaille l'algorithme de diffusion rapide. Dans la section 4.5, nous prouvons que l'algorithme proposé est presque certain, puis dans la section 4.6 nous prouvons sa performance aux niveaux du temps et du nombre de messages. À la section 4.7, nous présentons un ensemble de considérations et notes d'application. Le chapitre se conclue à la section 4.8 avec quelques problèmes ouverts.

4.2 Le modèle

Considérons le réseau suivant:

1. Le réseau est complet et comporte n noeuds, donc chaque noeud a un degré $n-1$. Nous verrons plus tard que seulement une petite partie des liens est utilisée par l'algorithme de communication.
2. Chaque noeud comporte un identificateur unique entre 0 et $n-1$; le noeud source porte l'identificateur 0.
3. Les processeurs communiquent de façon synchrone.
4. Les processeurs communiquent sur des liens bidirectionnels en mode port unique.

C'est-à-dire que chaque processeur peut appeler au plus un seul autre processeur à chaque ronde et que les noeuds communicants peuvent chacun envoyer et recevoir de l'information dans cette ronde.

5. Les pannes sont permanentes et de nature byzantine. Elles se produisent sur les liens et sur les noeuds de façon indépendante, avec des probabilités respectives $p > 0$ et $q > 0$. Nous supposons que les probabilités de pannes respectent la condition $(1-p)^2(1-q) > 0{,}5$ [1]. Nous supposons aussi que la source est fonctionnelle, sinon aucune diffusion n'est possible.

4.2.1 Terminologie et simplifications

Notons les conventions terminologiques et hypothèses qui furent utilisées au courant du chapitre:

1. Le log représente $\log_2$ tout au long des développements de ce chapitre.
2. Pour un événement E, $\overline{E}$ dénote son complément. Les symboles p_E et $\overline{p_E}$ dénotent respectivement les probabilités de E et $\overline{E}$.
3. Sans perte de généralité, pour alléger les formules, nous posons $n \, mod \lceil c \log n \rceil = 0$. Nous montrerons comment fonctionner sans cette supposition à la fin de ce chapitre.

4.3 Construction du réseau

Le réseau que nous utilisons pour effectuer la diffusion est construit comme suit:

1. Posons la constante $c = \frac{40q' \ln 2}{(2q'-1)^2}$ avec $q' = (1-q)(1-p)^2$ où q et p sont respectivement les probabilités que chaque noeud et chaque lien du graphe tombe en panne.
2. Nous séparons le graphe en groupes composés de $\lceil c \log n \rceil$ noeuds. Le noeud i appartient au groupe $\lfloor \frac{i}{\lceil c \log n \rceil} \rfloor$. Nous appellerons dorénavant les groupes des *supernoeuds*.
3. Afin de simplifier l'énoncé de l'algorithme et sans perte de généralité, chaque noeud i porte l'identificateur $i \, mod(\lceil c \log n \rceil)$ à l'intérieur de son supernoeud.

1. Cette condition n'est pas très restrictive. Elle est satisfaite, par exemple, pour $p = 0{,}1$ et $q = 0{,}2$. Les composants des réseaux réels sont en général plus fiables. Notre algorithme est donc applicable à des conditions pratiques.

4. Similairement à [3], nous disposons les supernoeuds dans une structure d'arbre binaire comportant $L+1$ niveaux: $l = 0,1,...,L$.
5. Le supernoeud à la base de l'arbre binaire porte le nom *RACINE*. Ce supernoeud contient le noeud SOURCE qui porte l'identificateur 0. Ce noeud est la source du message.
6. Chaque couple d'un supernoeud parent et d'un de ses supernoeuds enfants forme un graphe biparti complet. Alors, chaque noeud d'un supernoeud parent est lié à chaque noeud de ses supernoeuds enfants.
7. *RACINE* est un sous graphe complet du réseau. Aucun autre supernoeud ne comporte de lien interne.
8. *RACINE* est situé au niveau $l = 0$ dans la structure d'arbre binaire.
9. Les supernoeuds au niveau $l = 1$ de l'arbre binaire sont adjacents à *RACINE*. Les supernoeuds du niveau $l > 1$ sont adjacents à leurs supernoeuds parents du niveau $l - 1$.
10. Les supernoeuds sont ordonnés dans l'arbre binaire suivant leur identificateur, de la manière suivante :
    (a) *RACINE* porte l'identificateur 0;
    (b) Le fils gauche d'un supernoeud i porte l'identificateur $2i + 1$;
    (c) Le fils droite d'un supernoeud j porte l'identificateur $2j + 2$.

Pour chaque supernoeud dont l'identificateur est $k > 0$, l'identificateur de son supernoeud parent est $\lceil \frac{k-2}{2} \rceil$.

Le réseau construit ci-dessus sera dorénavant appelé *l'arbre binaire épais*.

4.4 Énoncé de l'algorithme

L'algorithme de diffusion proposé dans le présent chapitre fonctionne comme suit:

Initialement, le noeud $SOURCE$ informe le supernoeud $RACINE$ du message à diffuser.

RÉPETER : Si un supernoeud n'est pas une feuille et est informé du message à diffuser, sans l'avoir encore transmis à ses enfants:

1. ce supernoeud transmet le message à son fils gauche
2. ce supernoeud transmet le message à son fils droit

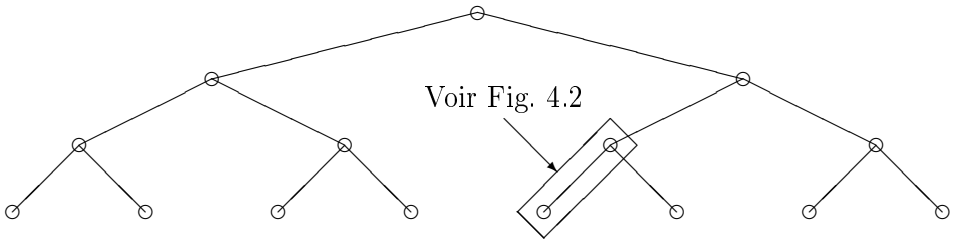

FIG. 4.1 – *Arbre binaire épais*

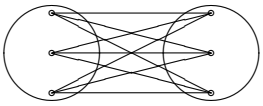

FIG. 4.2 – *Graphe biparti complet*

JUSQU'À ce que toutes les feuilles de l'arbre binaire épais connaissent le message.

Dans l'énoncé de l'algorithme qui suit, l'expression *id* représente l'identificateur de chaque processeur. Au moment de l'exécution, chaque processeur remplacera donc ce terme par son propre identificateur. Le mode de communication choisi dicte qu'à chaque appel, les noeuds échangent les messages qui leur sont connus. Les noeuds placent les messages reçus dans un tampon afin de conserver tous les messages en mémoire.

Les routines de communication de l'algorithme DRPB s'énoncent tel que détaillées dans les sous-sections suivantes

4.4.1 Le vote

La fonction de vote permet aux processeurs d'interpréter les messages qui leur parviennent. La méthode utilisée dans notre algorithme est le *vote majoritaire*. Le vote majoritaire s'énonce comme suit:

Si, sur l'ensemble des valeurs reçues par un processeur, on retrouve une valeur X plus de la moitié des fois, le processeur choisit cette valeur. Autrement, le processeur choisit la valeur par défaut.

La fonction VOTE s'énonce alors comme suit:

Fonction VOTE()
  Choisir la valeur majoritaire dans son tampon. Si aucune
    valeur n'est majoritaire, choisir la valeur par défaut

4.4.2 Diffusion dans le supernoeud RACINE

La routine RACINE-Propagation de l'algorithme effectue la diffusion à partir du noeud $SOURCE\,(0)$ à tous les autres noeuds de $RACINE$. Cette diffusion est faite en utilisant tous les noeuds de $RACINE$ comme intermédiaires.

Routine RACINE-Propagation()
  Pour tous les noeuds dans RACINE, en parallèle
    Pour i de 1 à $2\lceil c \log n \rceil$
      si $i - id > id$ ET $i - id \leq \lceil c \log n \rceil$ alors
        appeler le noeud $(i - id)$
        % 0-1; 0-2; 0-3,1-2; 0-4,1-3; 0-5,1-4,2-3;
        % 0-6,1-5,2-4; etc.
    VOTE()
    assigner $MESSAGE = 1$; %j'ai un message à transmettre

4.4.3 Transmission entre les supernoeuds

Pour transmettre l'information d'un supernoeud i à un supernoeud j, tous les noeuds de i envoient leur information à tous les noeuds de j.

Routine GROUPE-Transmission(i,j)
  Pour tous les noeuds du supernoeud i, en parallèle
    Pour k de 1 à $\lceil c \log n \rceil$
      appeler le noeud $id + k$ du supernoeud j
  Parallèlement, tous les noeuds du supernoeud j

VOTE()
assigner $MESSAGE = 1$; %j'ai un message à transmettre

4.4.4 Propagation dans l'arbre binaire épais

Afin de compléter une étape de la diffusion du message dans l'arbre, un supernoeud parent i qui est informé du message transmet ce message à ses supernoeuds enfants $2i+1$ et $2i+1$. La routine présentée ci bas transmet donc le message d'un niveau x au niveau $x+1$ de l'arbre binaire épais.

Routine ARBRE-Propagation
Pour tous les supernoeuds i, parallèlement
Si ((i n'est pas une feuille de l'arbre binaire épais) ET ($MESSAGE = 1$))
GROUPE-Transmission($i, 2i+1$); %supernoeud parent à son fils gauche
GROUPE-Transmission($i, 2i+2$); %supernoeud parent à son fils droit
Parallèlement, tous les noeuds du supernoeud i
assignent $MESSAGE = 0$; %je n'ai pas de message à transmettre

4.4.5 Algorithme DRPB

Le programme principal de l'algorithme consiste à propager le message dans $RACINE$ et ensuite à transmettre le message à travers les L niveaux suivants de l'arbre binaire épais. Nous allons démontrer que tous les noeuds de l'arbre binaire épais ont recevront le message correctement avec grande probabilité.

Algorithme DRPB
RACINE-Propagation();
Répéter L fois
ARBRE-Propagation()
% L est le nombre de niveaux dans l'arbre binaire
% épais, excluant le niveau de RACINE

4.5 Analyse de l'algorithme DRPB

Le théorème suivant énonce les caractéristiques de performance de l'algorithme DRPB.

Théorème DRPB : L'algorithme DRPB effectue la diffusion presque certaine dans l'arbre binaire épais de degré $O(\log n)$ en temps $O(\log^2(n))$ et avec un nombre d'appels $O(n \log n)$.

4.5.1 Degré de l'arbre binaire épais

Dans l'arbre binaire épais, chaque supernoeud comporte $\lceil c \log n \rceil$ noeuds. Chaque noeud est lié à tous les noeuds des supernoeuds adjacents. De plus, toutes les paires de noeuds de RACINE ont un lien entre eux. Pour chaque noeud (sauf ceux des supernoeuds feuilles), nous comptons donc 3 ensembles de $\lceil c \log n \rceil$ liens vers leurs noeuds voisins. Les noeuds des supernoeuds feuilles de l'arbre binaire épais comportent seulement des liens vers les noeuds de leur supernoeud parent respectif. C'est-à-dire que ces noeuds ont $\lceil c \log n \rceil$ liens vers leurs noeuds voisins. Le degré de l'arbre binaire épais est donc $O(\log n)$.

4.5.2 Preuve de la validité de l'algorithme DRPB

Soit M le message que le noeud SOURCE tente de diffuser par l'algorithme DRPB. Soit l'événement C décrit comme suit : la transmission du message M de la source vers tous les bons noeuds du graphe se fait correctement, c'est-à-dire qu'à la fin de l'algorithme, tous les bons noeuds concluent qu'ils ont reçu M. Nous pouvons reformuler l'événement C comme suit : étant donné un message M du noeud SOURCE, à la fin de l'algorithme, tous les bons noeuds dans chaque supernoeud S_i de l'arbre binaire épais concluent qu'ils ont reçu le message M. Soit $p_C = P(C)$ la probabilité de cet événement.

Soit l'événement CS_i énoncé comme suit : À la fin d'une transmission d'un supernoeud parent vers son supernoeud enfant S_i, tous les bons noeuds de S_i concluent qu'ils ont reçu le message M.

Soit l'événement C_{racine} s'énonçant comme suit : tous les bons noeuds du supernoeud RACINE concluent qu'ils ont reçu le message M. Soit $p_{C_{racine}} = P(C_{racine})$.

Soit l'événement $C_{source,i}$ dénotant que le noeud ayant l'identificateur i, faisant partie du supernoeud RACINE, conclue avoir reçu le message M du noeud SOURCE. Soit

$p_{C_{source,i}} = P(C_{source,i})$.

Soit la probabilité $p_{CS_i} = P(CS_i|CS_{(parent\,de\,i)})$. Étant donné une branche de l'arbre binaire épais dans lequel sont disposés les supernoeuds, nous calculons la probabilité que tous les événements CS_i pour les supernoeuds S_i de cette branche soient vrais:

Soit l'événement B_x énoncé comme suit : tous les bons noeuds de la branche x de l'arbre binaire épais reçoivent correctement le message M provenant du noeud SOURCE.

Soit le nombre de niveaux L de l'arbre binaire épais:

$p_{B_x} = P(B_x) = p_{C_{racine}} p_{CS_i}^L$

$L = \lceil \log((\frac{n}{\lceil c \log n \rceil} + 1)\frac{1}{2}) \rceil$

$L = \lceil \log(\frac{n+\lceil c \log n \rceil}{\lceil c \log n \rceil}\frac{1}{2}) \rceil$

$L = \lceil \log(n + \lceil c \log n \rceil) - \log(\lceil c \log n \rceil) - 1 \rceil$

Nous savons que $(x+y)/y \leq x, \forall x,y \geq 2$. Puisque $\lceil c \log n \rceil \geq 2$ et $\log n \geq 2$,

$\log(n + \lceil c \log n \rceil) - \log(\lceil c \log n \rceil) - 1 \leq L \leq \log(n + \lceil c \log n \rceil) - \log(\lceil c \log n \rceil)$

$L \leq \log(\frac{n+\lceil c \log n \rceil}{\lceil c \log n \rceil})$

$L < \log n$

$p_{C_{racine}} < p_{CS_i}$ puisque la transmission entre le noeud SOURCE et les bons noeuds de RACINE passe par des chemins comportant un noeud et deux liens potentiellement mauvais tandis que les transmissions entre noeuds d'un supernoeud et les bons noeuds du supernoeud fils passent par des chemins comportant un noeud et un lien potentiellement mauvais, c'est-à-dire $(1-p)(1-q) > (1-p)^2(1-q)$.

Donc, puisque les chemins de transmission de chaque étape de transmission sont indépendants, nous devons avoir

$p_{B_x} > p_{C_{racine}}^{L+1}$

$p_{B_x} > p_{C_{racine}}^{\log n+1}$

$p_{B_x} > p_{C_{racine}}^{\log(2n)}$

Nous avons $\overline{C} \subset \cup_{x-branche} \overline{B_x}$

Puisque le niveau L de l'arbre binaire épais comporte au plus 2^L supernoeuds, le nombre de branches dans l'arbre binaire épais est au plus 2^L. On a donc

$\overline{p_C} \leq 2^L \overline{p_{B_x}}$

$\overline{p_C} < 2^{\log n}\overline{p_{B_x}}$

$\overline{p_C} < n\overline{p_{B_x}}$

$\overline{p_C} < n(1 - p_{B_x})$

$\overline{p_C} < n(1 - p_{C_{racine}}^{\log(2n)})$

Il suffit donc de garantir, à partir d'un certain n,

$n(1 - p_{C_{racine}}^{\log(2n)}) \leq 1/n$, ou, de façon équivalente,

$1 - p_{C_{racine}}^{\log(2n)} \leq 1/n^2$

$p_{C_{racine}}^{\log(2n)} \geq 1 - 1/n^2$

$p_{C_{racine}} \geq (1 - 1/n^2)^{1/\log(2n)}$

Nous savons que $\overline{p_{C_{racine}}} \leq \sum_{i=1}^{\lceil c \log n \rceil} \overline{p_{C_{source,i}}}$ puisque la conjonction de tous les $C_{source,i}$ est nécessaire pour que C_{racine} soit vrai

$\lceil c \log n \rceil \overline{p_{C_{source,i}}} \geq \overline{p_{C_{racine}}}$

$1 - \lceil c \log n \rceil \overline{p_{C_{source,i}}} \leq p_{C_{racine}}$

Puisque $x > \log_2(x) + 1, \forall x > 2$, $\log(2n) = \log n + 1 < n$

Il suffit donc de garantir

$1 - n\overline{p_{C_{source,i}}} > (1 - 1/n^2)^{1/n}$

Nous avons

$(1 - 1/n^4)^{n^3} \to 1$ et

$(1 - 1/n^2)^{n^2} \to 1/e$

Donc, à partir d'un certain n,

$(1 - 1/n^4)^{n^3} > (1 - 1/n^2)^{n^2}$

C'est à dire

$1 - 1/n^4 > (1 - 1/n^2)^{1/n}$ est satisfait

Supposons que $\overline{p_{C_{source,i}}} \leq \frac{1}{n^5}$

Alors $1 - n\frac{1}{n^5} > (1 - 1/n^2)^{1/n}$

$1 - \frac{1}{n^4} > (1 - 1/n^2)^{1/n}$, à partir d'un certain n. Nous avons prouvé ceci dans les étapes précédentes.

Il suffit donc de garantir $\overline{p_{C_{source,i}}} \leq \frac{1}{n^5}$

Dénotons $p_e = \overline{p_{C_{source,i}}}$

Par la borne de Chernoff:

$p_e \leq e^{\frac{-(2q'-1)^2 mq'}{8q'^2}}$

où $m = \lceil c \log n \rceil$

Si l'on prend $c = \frac{40 \ln(2) q'}{(2q'-1)^2}$, on a

$p_e \leq e^{\frac{5 \ln(2) \log n (-(2q'-1)^2}{(2q'-1)^2/8q'} \frac{q'}{8q'^2}} = e^{-5 \ln(n)} = 1/n^5,$

ce qu'il fallait garantir.

Le critère de presque certitude est donc satisfait avec $c = \frac{40q' \ln 2}{(2q'-1)^2}$.

À la section suivante, nous démontrons que l'algorithme DRPB atteint bien les performances énoncées dans le théorème DRPB.

4.6 Performances

4.6.1 Temps requis

Le temps requis pour compléter l'exécution de l'algorithme DRPB est $O(\log^2(n))$.

Initialement, $2\lceil c \log n \rceil$ rondes sont nécessaires pour compléter l'exécution de la routine RACINE-Propagation. Ensuite, pour chaque niveau de l'arbre binaire épais, la routine GROUPE-Transmission est exécutée deux fois, à raison de $c \log n$ rondes chaque fois. Donc, le temps total de fonctionnement de DRPB est $2\lceil c \log n \rceil + 2L\lceil c \log n \rceil = 2(L+1)\lceil c \log n \rceil$. $L+1$ est le nombre total de niveaux dans l'arbre binaire épais étant donné le nombre de supernoeuds $\lceil n/\lceil c \log n \rceil \rceil$

$L + 1 = \lceil \log((n/\lceil c \log n \rceil)) \rceil = \lceil \log n - \log(\lceil c \log n \rceil) \rceil \leq \log n$ puisque la taille des supernoeuds $\lceil c \log n \rceil > 2$

Le temps total de fonctionnement de notre algorithme est donc, au plus, $2\lceil c \log n \rceil \log n$, avec c constant.

Donc, le temps total est dans $O(\log^2(n))$. L'ordre de temps optimal est inconnu pour cette classe de problèmes. $\Omega(\log n)$ est une borne inférieure triviale pour la diffusion, même sans pannes.

4.6.2 Nombre de messages requis

Initialement, le noeud SOURCE envoie $\lceil c \log n \rceil - 1$ copies du message (une copie pour chaque autre membre de RACINE). Chaque membre du supernoeud RACINE sauf SOURCE envoie à son tour $\lceil c \log n \rceil - 1$ messages (un pour chaque membre de son supernoeud). Un total de $\lceil c \log n \rceil (\lceil c \log n \rceil - 1)$ messages sont expédiés dans RACINE.

De plus, l'algorithme prévoit l'envoi du message à chaque noeud hors du supernoeud RACINE par $\lceil c \log n \rceil$ noeuds différents. Nous savons alors que $\lceil c \log n \rceil (n - \lceil c \log n \rceil)$ messages peuvent leur être envoyés.

Donc, au total, $\lceil c \log n \rceil (\lceil c \log n \rceil - 1) + \lceil c \log n \rceil (n - \lceil c \log n \rceil) = \lceil c \log n \rceil (n - \lceil c \log n \rceil + \lceil c \log n \rceil - 1)$ messages. Le nombre de messages est donc $O(n \log n)$.

4.7 Notes d'application

4.7.1 Nombre de messages et congestion du réseau

L'algorithme de diffusion DRPB a été développé pour les réseaux de communication et les portions de réseaux de communications dédiés, temporairement ou en permanence, à la transmission d'information en quantité connue. Par exemple, il serait approprié d'utiliser le réseau du type de l'arbre binaire épais avec cet algorithme pour la diffusion de l'information à travers un réseau industriel supportant les opérations d'une chaîne de production critique à la survie d'une entreprise. Ce réseau pourrait être dédié aux opérations de production le jour tout en transmettant les informations critiques à la chaîne d'approvisionnement la nuit.

Par contre, le grand nombre de messages généré dans l'exécution de DRPB pourrait congestionner un réseau de communications conventionnel. Donc, le praticien désirant ajouter de la tolérance aux pannes dans ce type de réseaux devrait choisir le meilleur compromis entre la fiabilité et le taux de congestion tolérable. Afin de réduire la densité du réseau, et le nombre de messages, il lui suffira de multiplier la constante de densité du réseau c par le facteur voulu plus petit que 1. La pénalité entraînée par une diminution de la densité est l'augmentation du taux d'échec des diffusions. En effet, une variation linéaire de la densité du réseau devrait entraîner une variation approximativement exponentielle du taux d'erreur dans la diffusion.

4.7.2 Topologie

Il est possible d'appliquer directement les résultats de ce chapitre dans tous les réseaux complets. Les réseaux structurés de connexité logarithmique peuvent aussi utiliser une version de DRPB adaptée à leur géométrie. En effet, il est possible d'appliquer les mêmes principes utilisés dans DRPB afin de permettre la diffusion tolérante aux pannes dans un hypercube, par exemple, ou d'autres réseaux structurés où des groupes peuvent être définis à l'aide de règles mathématiques et de nombres déterminés de bonds.

Il n'est pas possible d'appliquer les principes de l'algorithme DRPB sur les réseaux en étoile. Ces derniers sont, par définition, intolérants aux pannes permanentes.

4.7.3 Application de l'algorithme DRPB aux réseaux asynchrones

L'algorithme DRPB ne peut être appliqué intégralement aux réseaux asynchrones. En effet, on ne connaît pas le moment d'arrivée des messages dans ces réseaux. Alors, contrairement aux réseaux synchrones, il est impossible de connaître à l'avance le moment où tous les messages provenant des noeuds parents devraient normalement être reçus par un noeud enfant. De plus, puisque les composants en panne byzantine peuvent omettre d'envoyer des messages, un algorithme qui demande la réception de tous les messages avant le vote n'aura aucune chance de succès pour la diffusion en présence de pannes byzantines. Il faut donc imposer un décompte de temps limite pour la réception des messages, T_{limite}, qui commencerait au moment de la réception d'un nombre de messages minimal $N_{m,min}$. Le nombre $N_{m,min}$ peut être fixé à $\lceil m/2 \rceil + 1$ car, avec forte probabilité, parmi tant de messages au moins un doit être envoyé par un chemin sans pannes, ce qui élimine le risque de déclencher le décompte T_{limite} uniquement par des composants en panne byzantine. D'autre part le temps T_{limite} peut être fixé expérimentalement en prenant une borne raisonnable sur la différence des temps de réception des messages passant par les bon chemins

4.8 Conclusion

Dans le cas des algorithmes non adaptatifs (même pour contrer les pannes arrêt), la borne inférieure du nombre de messages nécessaires pour compléter l'exécution d'un algorithme est dans $O(n \log n)$[17]. L'algorithme DRPB performe aussi bien. Il est donc optimal, en ce qui concerne le nombre de messages.

Par contre, il est possible qu'il existe un moyen d'effectuer la retransmission du message M avant de passer au vote majoritaire. Ceci pourrait diminuer le temps nécessaire pour la diffusion dans des cas de faibles probabilités de pannes.

Notre algorithme fonctionne en temps $O(\log^2 n)$. La meilleure borne inférieure sur le temps est $\Omega(\log n)$. Il demeure ouvert de trouver quel est le temps optimal de diffusion dans notre scénario de pannes byzantines affectant les liens et les noeuds.

Chapitre 5

Diffusion rapide en présence de pannes byzantines - simulations

SOMMAIRE

Suite au développement de l'algorithme DRPB dans le chapitre précédent, il est important de supporter la validité de cet algorithme à l'aide de simulations. Ces simulations nous ont permis de constater que l'algorithme fonctionne mieux qu'estimé dans les calculs analytiques.

5.1 Introduction

Afin d'appuyer la preuve analytique exposée au chapitre précédent, nous avons développé un algorithme de simulation. Ainsi, nous avons fait exécuter un ensemble de simulations, appuyant la validité de l'algorithme de diffusion rapide en présence de pannes byzantines. Pour ce faire, nous avons codé le programme de simulation en utilisant le langage C++. Selon les paramètres de pannes et du réseau lui étant fournis à chaque exécution, ce programme effectue un nombre prédéterminé de simulations de diffusions dans le réseau comportant des pannes distribuées aléatoirement sur les liens et sur les noeuds. À chaque simulation, les pannes sont redistribuées aléatoirement et le processus de communication recommence.

Étant donné le grand nombre d'opérations nécessaire pour effectuer la simulation de grands réseaux, nous avons dû simplifier le modèle des simulations le plus possible. Les

simplifications utilisées ont réduit le degré de complexité du programme et ont eu comme effets positifs d'augmenter la cadence des simulations et de faciliter la modélisation.

Le restant du chapitre se divise ainsi: La section 5.2 explique les simplifications que nous avons utilisées dans la simulation de l'algorithme. Ensuite, la section 5.3 détaille l'algorithme simplifié que nous avons utilisé pour la simulation. Puis la section 5.4 expose les résultats des simulations. Le chapitre se conclue à la section 5.6 avec une avenue de recherche à considérer.

5.2 Simplifications

Dans les sous-sections suivantes, nous présentons chaque simplification utilisée dans la simulation de l'algorithme de diffusion presque certaine.

5.2.1 Sérialisation de l'exécution

L'algorithme de diffusion rapide et presque certaine est composé de plusieurs grandes étapes :

1. diffusion de $SOURCE$ aux noeuds de $RACINE$
2. transmissions entre les noeuds de $RACINE$
3. transmission de chaque supernoeud parent à son (ses deux) supernoeud(s) enfant(s)

La diffusion de $SOURCE$ aux autres noeuds de $RACINE$ est une suite d'événements consécutifs. D'abord, $SOURCE$ transmet au noeud 1 de $RACINE$, puis au noeud 2 de $RACINE$ et ainsi de suite.

En second lieu, la transmission entre les noeuds de $RACINE$ se termine au moment où tous les noeuds décident, en parallèle et indépendamment, de la valeur qu'ils considèrent localement vraie. Cet événement est précédé, pour chaque noeud, de la réception du message de $SOURCE$ relayé par chaque autre noeud. Puisque chaque envoi-réception de cette étape n'a pas d'influence sur les autres envois-réceptions, il n'est pas nécessaire d'ordonner ces transmissions d'une manière spécifique. Donc, il est possible d'effectuer la simulation en ordonnant les opérations de sorte que les noeuds effectuent l'ensemble de leurs transmissions consécutivement un à la suite de l'autre: d'abord 1 envoie à tous, puis 2 envoie à tous et ainsi de suite.

Troisièmement, un raisonnement analogue s'applique à la transmission de chaque noeud parent à ses noeuds enfants.

De plus, la structure de l'arbre binaire épais isole "géographiquement" les étapes de réception de données. En effet, la réception d'un ensemble de valeurs par un noeud du supernoeud i de niveau l n'a aucune influence sur la réception de l'ensemble de valeurs par un noeud du supernoeud j ($i \neq j$) du même niveau l. Par contre, les valeurs reçues par les noeuds d'un supernoeud peuvent influencer les valeurs reçues par les noeuds de ses supernoeuds enfants; nous traitons cet aspect à la section 5.2.3.

La séparation géographique permet de simuler correctement le fonctionnement de l'algorithme dans l'arbre binaire épais sans l'exécution de processus concurrents. Les transmissions entre supernoeuds peuvent être faites une à la suite de l'autre.

Donc, afin de simuler l'algorithme développé au chapitre précédent, il est possible d'utiliser une suite d'événements sans ordonnance particulière à l'intérieur de chaque étape. Il est simplement nécessaire de respecter l'ordre des étapes afin de simuler correctement l'ordre des événements issus de l'algorithme. Ce constat permettra de poser plusieurs simplifications qui suivent.

5.2.2 Conservation de l'état des noeuds et des liens

Les affectations de valeurs sont des opérations qui consomment normalement peu de temps dans l'exécution d'un programme. Par contre, lorsque le nombre d'affectations est grand, ce temps peut devenir considérable. Dans le cas particulier de notre algorithme de communications, peu d'opérations de calcul sont faites par rapport aux opérations de transfert de données, cet aspect prend alors une importance considérable.

À la fin de l'exécution de l'algorithme de diffusion rapide, chaque lien du réseau aura été utilisé au plus une fois. Donc, puisque chaque lien a une probabilité de panne égale et que les pannes sont indépendantes, il suffit de générer chaque état de panne au moment de son utilisation sans conserver cet état en mémoire. Cet état de panne doit alors être généré au moment de chaque transmission de données entre deux noeuds du réseau.

De façon similaire, l'état de panne des noeuds n'est pas un paramètre à conserver en mémoire. La simulation de l'algorithme pouvant être sérialisée, il est possible de simuler l'ensemble des transmissions d'un supernoeud vers ses enfants en une suite d'événements consécutifs. On peut donc générer l'état d'un noeud au moment de choisir comment il effectuera l'ensemble de ses transmissions, sans conserver cet état en mémoire. Par

contre, dans le cas du supernoeud RACINE, il est nécessaire de conserver l'état de chaque noeud en mémoire puisque ces noeuds participent à deux ensembles d'envois non consécutifs (envois aux autres noeuds de $RACINE$ et aux supernoeuds enfants). Nous avons donc créé deux routines de simulation afin de représenter cette réalité.

5.2.3 Mémoire des décisions locales

L'algorithme de transmission de messages est une composition de plusieurs transmissions entre un supernoeud parent et ses supernoeuds enfants. En effet, chaque noeud d'un supernoeud parent doit envoyer des messages à tous les noeuds de ses supernoeuds enfants. Un noeud enfant ne pourra envoyer de message que lorsqu'il aura reçu le message de tous les noeuds de son supernoeud parent et qu'il aura décidé localement quelle valeur il considère vraie. La coupure entre les rôles de supernoeud enfant et de supernoeud parent se fait donc à la fin du vote.

Afin d'augmenter la vitesse de l'ensemble des simulations, il est correct d'arrêter une simulation dès qu'un bon noeud prend une mauvaise décision. Lors d'un tel vote, nous savons que l'algorithme a subi un échec. Alors, il devient inutile de conserver les messages contenus dans chacun des noeuds à la fin de leurs phases individuelles de vote; il faut seulement vérifier que tous les bons noeuds ont choisi le bon message afin de pouvoir continuer la simulation. Alors, dans un algorithme de simulation adoptant ce fonctionnement, le seul message important est le message original M. Il est donc possible de modéliser un supernoeud parent à l'aide d'un générateur de nombres aléatoires et du message original M. Lorsqu'on a généré l'état d'un noeud parent, il ne suffit plus que d'envoyer les messages à tous ses enfants, avant de passer au prochain noeud parent.

Dans l'élaboration de l'algorithme de simulation, le critère déterminant si une exécution de l'algorithme a été un succès est traduit par l'événement suivant : tous les bons noeuds d'un réseau ont choisi le bon message. Comme dans le chapitre précédent, appelons cet événement C. Appelons C' l'événement suivant : tous les noeuds d'un réseau ont choisi le bon message. Évidemment, $P(C) \geq P(C')$.

La probabilité de succès de l'algorithme est supérieure ou égale à la probabilité de succès calculée par le programme de simulations. Alors, nous pouvons être certains qu'une statistique de fonctionnement de l'algorithme DRPB calculée par notre algorithme de simulation sera une borne inférieure de sa performance réelle.

5.3 Algorithme de simulation

5.3.1 Distribution aléatoire du message originel

Sans perte de généralité, nous considérons que le message à envoyer par $SOURCE$ est 0.

5.3.2 Terminologie

Tel qu'expliqué dans la section précédente, voici l'algorithme de simulation qui a été utilisé pour calculer les résultats pratiques de ce chapitre (voir l'annexe A pour le code C++ complet). Les termes suivants seront employés dans l'algorithme de la sous-section suivante.

- $vote0$: Compteur des votes pour le message 0
- $vote1$: Compteur des votes pour le message 1
- $Échec$: Variable booléenne, vraie si l'algorithme simulé échoue
- $SOURCE$: Noeud qui envoie le message original (0)
- $RACINE$: Supernoeud qui contient $SOURCE$
- $Racine$: Ensemble des tampons de réception de messages des noeuds de $RACINE$. La première dimension du tableau est l'identificateur du noeud récepteur et la seconde dimension celui du noeud expéditeur.
- $BonNoeud$: Tableau de l'état de panne des noeuds employé pour les transmissions de $RACINE$
- $EnfantGauche$: Ensemble des tampons de réception de messages des noeuds du supernoeud enfant gauche. Les dimensions sont comme celles de $Racine$.
- $EnfantDroite$: Ensemble des tampons de réception de messages des noeuds du supernoeud enfant droite. Les dimensions sont comme celles de $Racine$.
- $L_{i,j,g}$: Lien entre le noeud parent i et le noeud j du supernoeud enfant gauche (substituer g par d pour droite).
- $L_{i,j}$: Lien entre le noeud i et le noeud j de $RACINE$.
- $NombreTotalNoeuds$: Nombre total de noeuds dans le réseau.
- $NombreGroupesRéseau$: Nombre total de supernoeuds dans le réseau.
- $NombreGroupes$: Nombre de supernoeuds ayant complété leur réception de messages dans la simulation courante.
- $NombreNoeudsParGroupe$: Nombre de noeuds par supernoeud.

– $NombreEssais$: Nombre de simulations à exécuter.

5.3.3 Algorithme

L'algorithme de simulation s'énonce comme suit:

VOTER(Tampons[i])
    %$Tampons[i]$ est le tampon du noeud i
    %0 est le bon message et 1 est le mauvais message

    Compter le nombre de votes pour 0 ($vote0$) et 1 ($vote1$) dans $Tampons[i]$
    si $vote0 > vote1$
        $Échec = faux$
    sinon
        $Échec = vrai$
    Retourner $Échec$

RACINE-Propagation($BonNoeud$)
    %$BonNoeud$ est le tableau des états des noeuds
    %$Racine$ est le tableau des tampons de réception des
    %noeuds du supernoeud RACINE
    %$NombreNoeudsParGroupe$ est le nombre de noeuds qui est
    %simulé dans chaque supernoeud
    %0 est le bon message et 1 est le mauvais message

    $Échec = faux$
    $BonNoeud[0] = vrai$ %le noeud $SOURCE$ est sans panne
    pour les noeuds $i \neq 0$ du supernoeud $RACINE$
        $BonNoeud[i]$ = état généré selon la probabilité de panne des noeuds
        si $L_{SOURCE,i}$ est bon (généré)
            $Racine[i][0] = 0$
        sinon
            $Racine[i][0] = 1$
    pour les noeuds $i \neq 0$ du supernoeud $RACINE$
        pour un décalage j de 0 à $NombreNoeudsParGroupe - 1$
            si le noeud i et $L_{i,i+j}$ sont sans panne

$Racine[i+j][i] = 0$
sinon
$Racine[i+j][i] = 1$
pour les noeuds i de 1 à $NombreNoeudsParGroupe - 1$
$\acute{E}chec = \acute{E}chec$ OU $VOTER(Racine[i])$
Retourner $\acute{E}chec$

RACINE-Transmission($BonNoeud$)
%$BonNoeud$ est le tableau des états des noeuds
%$EnfantGauche$ est le tableau des tampons de réception des
%noeuds du supernoeud enfant gauche
%$EnfantDroite$ est le tableau des tampons de réception des
%noeuds du supernoeud enfant droite
%$NombreNoeudsParGroupe$ est le nombre de noeuds qui est
%simulé dans chaque supernoeud
%0 est le bon message et 1 est le mauvais message

$\acute{E}chec = faux$
pour les noeuds i du supernoeud
pour un décalage j de 0 à $NombreNoeudsParGroupe - 1$
si le noeud i et le $L_{i,i+j,g}$ ne sont pas en panne
$EnfantGauche[i+j][i] = 0$
sinon
$EnfantGauche[i+j][i] = 1$
si le noeud i et $L_{i,i+j,d}$ ne sont pas en panne
$EnfantDroite[i+j][i] = 0$
sinon
$EnfantDroite[i+j][i] = 1$
pour un compteur k de 1 à $NombreNoeudsParGroupe$
$\acute{E}chec = \acute{E}chec$ OU $VOTER(EnfantGauche[k])$
$\acute{E}chec = \acute{E}chec$ OU $VOTER(EnfantDroite[k])$
Retourer $\acute{E}chec$

GROUPE-Transmission()

%"EnfantGauche" est le tableau des tampons de réception des
%noeuds du supernoeud enfant gauche
%"EnfantDroite" est le tableau des tampons de réception des
%noeuds du supernoeud enfant droite
%"NombreNoeudsParGroupe" est le nombre de noeuds qui est
%simulé dans chaque supernoeud
%0 est le bon message et 1 est le mauvais message

$\acute{E}chec = faux$
pour les noeuds i du supernoeud
    pour un décalage j de 0 à $NombreNoeudsParGroupe - 1$
        si le noeud i et le $L_{i,i+j,g}$ ne sont pas en panne
            $EnfantGauche[i+j][i] = 0$
        sinon
            $EnfantGauche[i+j][i] = 1$
        si le noeud i et le $L_{i,i+j,d}$ ne sont pas en panne
            $EnfantDroite[i+j][i] = 0$
        sinon
            EnfantDroite[i+j][i] = 1
pour un compteur k de 0 à $NombreNoeudsParGroupe - 1$
    $\acute{E}chec = \acute{E}chec$ OU $VOTER(EnfantGauche[k])$
    $\acute{E}chec = \acute{E}chec$ OU $VOTER(EnfantDroite[k])$
Retourner $\acute{E}chec$

ALGORITHME SIMULATION($argumentsd'exécution$)
    Saisie des arguments d'exécution
        $NombreTotalNoeuds$
        $NombreGroupesRéseau$
        $NombreNoeudsParGroupe$
        $NombreEssais$
        Fichier de sortie
        Probabilité de panne des noeuds
        Probabilité de panne des liens
    Calcul des valeurs manquantes dans les arguments d'exécution
    Répéter $NombreEssais$ fois

$\acute{E}chec = faux$
%dans l'instruction suivante BonNoeud est un tableau de valeurs de retour
$\acute{E}chec = \acute{E}chec$ OU $RACINE - Propagation(BonNoeud)$
%dans l'instruction suivante BonNoeud est un tableau en paramètres
$\acute{E}chec = \acute{E}chec$ OU $RACINE - Transmission(BonNoeud)$
$NombreGroupes = 3$
Tant que ($\acute{E}chec = faux$) et ($NombreGroupes < NombreGroupesRéseau$)
$\acute{E}chec = \acute{E}chec$ OU $GROUPE - Transmission()$
NombreGroupes =NombreGroupes+2;
Enregistrer le résultat des essais

Tel qu'expliqué à la section précédente, le fonctionnement simplifié de cet algorithme est conforme à l'algorithme DRPB. Les résultats de simulation sont présentés à la section suivante.

5.4 Résultats de simulation

Considérons un ensemble de processeurs opérant dans un environnement physique quelconque. Posons l'hypothèse que les probabilités de pannes sont constantes pour les liens et les processeurs. Alors, nous avons un réseau avec des probabilités de pannes ainsi qu'un nombre de noeuds fixes. Ceci est généralement le cas pour les problèmes d'architecture des réseaux. Alors afin d'optimiser les communications dans un tel réseau avec notre algorithme de communications, nous pouvons ajuster sa configuration physique. Ceci est la prémisse à la série de simulations effectuées dans cette section du mémoire.

5.4.1 Connexité requise

Le principal résultat des simulations est le constat suivant: l'algorithme DRPB peut effectuer des diffusions presque certaines avec moins de noeuds dans chaque supernoeud que le nombre ($\lceil c \log n \rceil$) prévu dans la section 4.3. Alors, les communications peuvent être presque certaines dans un réseau plus économique que prévu, dont la connexité est inférieure.

Cet écart entre les calculs théoriques et les simulations s'explique comme suit : plusieurs approximations ont dû être utilisées tout au long des calculs. Afin de garantir les

conditions de fiabilité voulues, il a été nécessaire, à chaque approximation, de considérer une probabilité d'erreur plus grande ou égale à la probabilité réelle. Notons, entre autres, que la borne de Chernoff, le principal outil employé au cours de ces calculs, est une borne supérieure sur une probabilité plutôt que sa valeur exacte.

Donc, afin de trouver la véritable connexité nécessaire pour effectuer une diffusion presque certaine dans un réseau avec l'algorithme DRPB, plusieurs séries de simulations ont été effectuées avec un nombre total de noeuds constant, pour diverses valeurs des probabilités de pannes. En variant la connexité du réseau, nous avons trouvé celle pour laquelle le taux d'erreur était le plus rapproché de $1/n$, et inférieur à ce nombre. Cet ensemble de simulations révèle le nombre minimal de noeuds par supernoeud pour assurer une diffusion presque certaine (la connexité minimale est trois (3) fois ce nombre). La figure 5.1 illustre une tendance du taux d'erreur en variant la connexité du réseau pour $p = 0{,}1$, $q = 0{,}05$ et $n = 10000$. Avec ces conditions, 55 noeuds par supernoeuds étaient suffisants pour garantir la fiabilité $1 - \frac{1}{10000}$ de la diffusion.

FIG. 5.1 – *Tendance du taux d'erreurs, p=0,1, q=0,05, n=10 000*

Le tableau 5.1 résume les résultats de recherche au volet du nombre de noeuds minimal par supernoeud pour assurer des communications presque certaines dans un réseau de 10000 noeuds. Chaque résultat a été établi pour des probabilités de pannes différentes.

p	q	# supernoeuds	# noeuds/supernoeud	clogn	$\frac{\#noeuds/supernoeud}{clogn}$
0.10	0.05	182	55	382	14%
0.10	0.10	134	75	529	14%
0.10	0.15	95	106	781	14%
0.10	0.20	65	154	1266	12%
0.10	0.25	30	334	2400	14%
0.10	0.30	13	770	6178	12%
0.05	0.10	223	45	285	16%
0.15	0.10	50	200	1229	16%
0.20	0.10	17	589	4801	12%
0.05	0.05	334	30	218	14%
0.15	0.05	80	125	799	16%
0.20	0.05	40	250	2378	11%

TAB. 5.1 – *Statistiques des simulations*

Dans tous les cas examinés, le rapport entre le nombre de noeuds par supernoeud permettant la diffusion presque certaine en pratique et le nombre $\lceil c \log n \rceil$ prévu théoriquement égale approximativement 15%. Alors, approximativement 15% du nombre $\lceil c \log n \rceil$ est nécessaire pour obtenir la fiabilité voulue. Nous notons $\lceil c' \log n \rceil = \lceil 15\% c \log n \rceil$ et donc $c' = \frac{6q' \ln 2}{(2q'-1)^2}$. Nous définissons l'algorithme DRPB* comme l'algorithme DRPB dont la constante c est substituée par la constante $c' = \frac{6q' \ln 2}{(2q'-1)^2}$.

Une conséquence de la diminution du nombre de noeuds par supernoeud est la diminution du nombre de liens nécessaires entre les noeuds et, par conséquent, la diminution des coûts matériels du réseau. De plus, ceci entraîne une diminution du nombre de messages nécessaire pour l'exécution de l'algorithme DRPB et donc une diminution dans le temps de calcul accordé par chaque processeur dans l'exécution de l'algorithme.

5.5 Fiabilité de l'algorithme en fonction du nombre de noeuds dans le réseau

Afin de justifier qu'un algorithme effectue la diffusion de manière presque certaine, il est nécessaire de simuler son fonctionnement pour plusieurs valeurs du nombre n de noeuds dans le réseau. Les sections suivantes présentent les résultats obtenus pour ce volet du travail de simulation.

5.5.1 Fiabilité de l'algorithme DRPB* en fonction de n

Dans la section 5.4.1, nous avons montré que l'algorithme DRPB peut être presque certain avec $\lceil c' \log n \rceil = \lceil 15\% c \log n \rceil$ noeuds par supernoeud, d'où la définition de l'algorithme DRPB*. Alors afin de trouver la relation entre la fiabilité de DRPB* et le nombre total de noeuds dans un réseau, nous avons effectué des simulations avec $\lceil c' \log n \rceil$ noeuds par supernoeud. Les résultats des simulations sont représentés dans la figure 5.2.

FIG. 5.2 – *Tendance du taux d'erreurs, p=0,1, q=0,1,* $\lceil c' \log n \rceil$ *noeuds par supernoeud*

Dans la figure 5.2, nous voyons que l'erreur réelle de DRPB* se situe sous la courbe $1/n$. Alors l'algorithme DRPB* est expérimentalement presque certain. Dans la même figure, nous notons que la pente de la courbe d'erreur réelle est beaucoup plus prononcée que celle de $1/n$ entre $n = 26$ et $n = 32$. En effet, lorsque $n = 26$, $c' \log n > 26$ pour $p = q = 10\%$. Alors pour ces petites valeurs de n, il est impossible de former même un supernoeud avec au moins $\lceil c' \log n \rceil$ noeuds. La figure 5.3 montre le point d'intersection entre les résultats de simulation et la courbe $1/n$ (fiabilité désirée). La valeur $n = 21$ à ce point représente la limite du fonctionnement correct de DRPB* pour les probabilités de panne $p = 10\%$ et $q = 10\%$.

FIG. 5.3 – *Limite de fonctionnement de DRPB, p=0,1, q=0,1, nombre maximal de noeuds par supernoeud*

5.5.2 Fiabilité de l'algorithme DRPB en fonction de n

Afin de déterminer la tolérance réelle aux pannes byzantines de l'algorithme DRPB, nous avons exécuté une série de simulations avec le nombre de noeuds par supernoeuds tel que défini dans la section 4.3. Les probabilités de panne considérées étaient $p = q = 10\%$.

D'après les résultats obtenus à l'issue de ces simulations, l'algorithme DRPB permet d'effectuer des diffusions d'information sans aucune erreur. Par contre, ceci n'est pas réaliste. Nous soupçonnons plutôt que les résultats obtenus sont dus aux limites du générateur de nombres aléatoires *rand*.

En effet [13], les générateurs de nombres aléatoires sont en réalité pseudo-aléatoires. Ils produisent mathématiquement des nombres qui simulent le hasard. Différents générateurs offrent différents degrés de qualité dans leurs simulation du hasard. Il serait intéressant de reprendre les simulations de la présente section avec différents générateurs pseudo-aléatoires afin de comparer les résultats.

Ce travail n'a pas été effectué puisqu'il sort du cadre de la recherche. En effet, le domaine des générateurs de nombres pseudo-aléatoires est complexe et très différent de

notre domaine d'intérêt. Pour notre recherche, il suffit de savoir que ces générateurs ne font qu'approximer un comportement aléatoire et qu'un autre générateur que *rand* aurait probablement permis l'obtention de résultats différents pour la présente section des simulations.

5.6 Conclusion

Dans le travail de simulation, nous avons vu que l'algorithme DRPB fonctionne mieux en réalité que dans les prévisions théoriques. Nous avons alors défini un algorithme DRPB* plus économique, qui utilise des supernoeuds avec $\lceil 15\% c \log n \rceil$ noeuds. Nous avons étudié expérimentalement la fiabilité de ce dernier algorithme en fonction de n. Il s'est avéré que pour $n \geq 21$, même cet algorithme plus économique assure la diffusion presque certaine, et ce, malgré des taux élevés de pannes ($p = q = 10\%$).

Chapitre 6

Stratégie de vote optimale

SOMMAIRE

Des solutions ont été développées pour la communication rapide et presque certaine dans les réseaux de connexité logarithmique affligés par des pannes byzantines. Ce chapitre traite d'une stratégie optimale de réception des messages pour un noeud dans un réseau de connexité arbitraire. Cette stratégie est optimale puisqu'elle a une probabilité de succès maximale.

6.1 Introduction

La communication en présence de pannes est possible dans les réseaux d'information avec l'implantation d'algorithmes robustes de transmission de données. Les pannes byzantines sont les pires à traiter. Les éléments d'un réseau affligés de celles-ci peuvent coordonner leurs efforts pour déjouer les algorithmes de communication. Au chapitre 4 une solution a été développée pour la communication presque certaine en présence de pannes byzantines dans un réseau de connexité logarithmique. Par contre, pour beaucoup de réseaux de grande taille, il est difficile, techniquement et financièrement, d'assurer une telle connexité. Il faut alors chercher des méthodes de transmission de données pour les réseaux dont la connexité est arbitraire. La *tore* et la *grille* sont deux exemples de réseaux dont la connexité est faible.

Les résultats de ce chapitre ont été publiés dans l'article de conférence suivant: Michel Paquette et Andrzej Pelc, *Optimal Decision Strategies in Byzantine Environments,* Proc. 11th International Colloquium on Structural Information and Communication

Complexity (SIROCCO), Smolenice Castle, Slowakia, Juin 2004, pp. 245-254 (Référence [14]).

Le restant du chapitre se divise ainsi: La section 6.2 détaille le modèle considéré dans la résolution du problème de stratégie optimale. Ensuite, la section 6.3 introduit les concepts préliminaires de la solution. Puis, la section 6.4 définit la stratégie de vote optimale et prouve son optimalité. À la section 6.5, nous considérons une généralisation de notre solution. Puis à la section 6.6, nous présentons quelques notes d'application. Le chapitre se conclue à la section 6.7 avec quelques problèmes ouverts.

6.2 Le modèle

Considérons un réseau quelconque dans lequel chaque noeud est relié à chaque autre noeud par k chemins disjoints. Deux chemins entre les noeuds s et d sont disjoints s'ils n'ont aucun lien ni noeud en commun sauf s et d. La Figure 6.2 illustre deux noeuds (en haut et en bas) reliés par trois chemins disjoints.

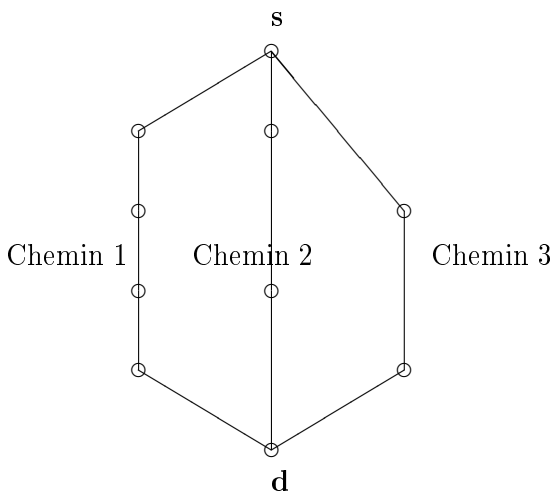

FIG. 6.1 – *Noeuds s et d, reliés par trois chemins disjoints*

Nous considérons le problème suivant de transmission entre deux noeuds du réseau: s, la source, et d, le destinataire. Le noeud s envoie un message m au noeud d à travers les k chemins disjoints qui les relient. Le noeud d connaît la fiabilité de chacun de ces k chemins[1]. Nous supposons que la fiabilité de chaque chemin est au moins $\frac{1}{2}$. Après

1. Par exemple, un chemin avec deux noeuds intermédiaires contient deux noeuds et trois liens. Si

avoir reçu les k versions du message envoyé par s, le noeud d applique une stratégie de décision pour trouver le message qu'il considère comme correct. Une stratégie optimale de décision en est une qui donne la probabilité maximale de trouver le message correct.

Dans le restant du chapitre nous développerons une stratégie optimale de décision pour la réception des messages et prouverons son optimalité.

6.3 Notions et propriétés préliminaires

Soit $E = \{0,1\}$ l'ensemble des messages pouvant être émis par une source s. Pour $v \in E$, $q(v)$ est la probabilité *a priori* de v, c'est-à-dire la probabilité que la source s émette v. Sans perte de généralité, posons que $q(0) \geq q(1)$ pour le restant de ce chapitre.

Soit $C = \{1,2,...,k\}$ l'ensemble des chemins disjoints entre la source s et le destinataire d. On associe à C l'ensemble $P = \{p_1,p_2,...,p_k\}$ tel que $\forall i = 1,2,...,k$, p_i est la probabilité de panne du chemin $i \in C$. Nous supposons que $p_i \leq \frac{1}{2}$ pour chaque $i \leq k$.

Notons par $R(i)$ la valeur binaire livrée au noeud d par un chemin i. Un rapport R est le vecteur de toutes les valeurs $R(i)$ livrées au noeud d par les chemins i, c'est-à-dire $R = [R(1)R(2)...R(k)]$. Notons par $\mathcal{R}$ l'univers des rapports R possibles, c'est-à-dire l'ensemble de tous les vecteurs binaires de longueur k.

6.3.1 Espace des probabilités

Définissons l'espace des événements élémentaires $U = \{(B,v) \mid B \subset C, v \in E\}$ tel que B est l'ensemble des bons chemins et v est le message envoyé par la source.

La probabilité que tous les chemins $i \in B$ soient des bons chemins et que les autres chemins soient en panne est donc :

$I(B) = \prod_{i \in B}(1 - p_i) \cdot \prod_{i \in C \setminus B} p_i$, car les événements qu'un chemin particulier est bon sont indépendants puisque les chemins sont disjoints.

Nous définissons une fonction de probabilité sur U par la formule $P(B,v) = q(v) \cdot I(B)$. Intuitivement, il s'agit de la probabilité que v a été envoyé et que les bons chemins sont exactement ceux dans B.

Un rapport R est compatible avec (B,v) si $\forall i \in B, R(i) = v$. Chaque rapport

q et p sont respectivement les probabilités que chaque noeud et que chaque lien tombe en panne, la fiabilité de ce chemin est $(1-p)^3(1-q)^2$.

compatible avec (B,v) est possible si l'événement (B,v) a lieu. Puisqu'il est impossible de prévoir le comportement des chemins affligés de pannes byzantines, les valeurs $R(i)$ pour les chemins $i \in C \setminus B$ peuvent être quelconques.

Notons $\rho(B,v) = \{R \mid \forall i \in B\, R(i) = v\}$ l'ensemble des rapports compatibles avec l'événement (B,v). Puisque $\mid E \mid = 2$ et $\mid C \setminus B \mid$ positions de R restent indéfinies, (B,v) est compatible avec $2^{|C\setminus B|}$ rapports R. Alors $\mid \rho(B,v) \mid = 2^{|C\setminus B|}$.

Une stratégie de décision D est une fonction qui fait correspondre à un rapport R une valeur $v \in E$, c'est-à-dire que $D : \mathcal{R} \rightarrow E$.

Définissions $Exact(D) = \{(B,v) \mid \forall R \in \rho(B,v), D(R) = v\}$ l'événement que la stratégie D donne une réponse correcte pour n'importe quel rapport possible, c'est-à-dire pour chaque comportement possiblement adopté par les chemins en panne.

6.3.2 $Fi(D)$, la fiabilité de D

Notons $Fi(D) = P(Exact(D))$ la probabilité que la stratégie D donne une réponse correcte (quels que soient les messages obtenus à travers les chemins affligés de pannes). Nous appelons ce nombre la fiabilité de la stratégie D.

$Fi(D) = P(\cup_{(B,v)\in Exact(D)}(B,v))$

$Fi(D) = \sum_{(B,v)\in Exact(D)} P(B,v)$.

Une stratégie D est appelée optimale, si $Fi(D) \geq Fi(D')$ pour n'importe quelle stratégie D'. Une stratégie optimale a donc la fiabilité maximale parmi toutes les stratégies de décision.

Nous notons $CD(B,v) = (C \setminus B, 1-v)$ le complément de (B,v) pour $E = \{0,1\}$.

Lemme 1 : Soit D une stratégie de décision. Si $(B,v) \in Exact(D)$ alors $CD(B,v) \notin Exact(D)$.

Preuve du lemme 1 : Supposons que le lemme 1 est faux. Alors pour un certain $(B,v) \in Exact(D)$, nous avons que $CD(B,v) \in Exact(D)$.

Considérons le rapport R suivant:

$$\mathrm{R}(i) = \begin{cases} v & \text{si } i \in B \\ 1-v & \text{sinon.} \end{cases}$$

Alors $R \in \rho(B,v) \cap \rho(CD(B,v))$.

Donc $(B,v) \in Exact(D)$ et $CD(B,v) \in Exact(D)$ implique

$D(R) = v = 1 - v$,

contradiction.

Fin de la preuve du lemme 1.

Lemme 2 : Si $B \subset B'$, alors $I(B) \leq I(B')$.

Preuve du lemme 2 : Il suffit de démontrer le lemme pour $B' = B \cup \{j\}$

$I(B) = \prod_{i \in B}(1 - p_i) \cdot \prod_{i \notin B} p_i$ et $I(B') = \prod_{i \in B'}(1 - p_i) \cdot \prod_{i \notin B'} p_i$.

$I(B') = \prod_{i \in B \cup \{j\}}(1 - p_i) \cdot \prod_{i \notin B \cup \{j\}} p_i$.

$\frac{I(B')}{I(B)} = \frac{\prod_{i \in (B \cup \{j\})}(1-p_i) \cdot \prod_{i \notin (B \cup \{j\})} p_i}{\prod_{i \in B}(1-p_i) \cdot \prod_{i \notin B} p_i}$

$\frac{I(B')}{I(B)} = \frac{(1-p_j)}{p_j} \geq 1$, car $p_j \leq \frac{1}{2}$. Donc $I(B) \leq I(B')$.

Fin de la preuve du lemme 2.

6.4 La stratégie optimale de réception de messages

Nous définissons $Z(R, v) = \{i \in C \mid R(i) = v\}$ l'ensemble des chemins entrants dans d qui rapportent la valeur v.

Définition : La stratégie de décision OPT est définie comme suit:

$$\mathrm{OPT}(R) = \begin{cases} 0 & \text{si } q(0) \cdot I(Z(R,0)) \geq q(1) \cdot I(Z(R,1)) \\ 1 & \text{sinon.} \end{cases}$$

Lemme 3 : $Exact(OPT) = \{(B,v) \mid q(v) \cdot I(B)) > q(1 - v) \cdot I(C \setminus B))\} \cup \{(B,0) \mid q(0) \cdot I(B) = q(1) \cdot I(C \setminus B)\}$

Preuve du lemme 3 :

Soit $(B,v) \in U$. Considérons les quatre cas possibles.

Cas 1: $q(v) \cdot I(B) > q(1 - v) \cdot I(C \setminus B)$

Soit $R \in \rho(B,v)$. Donc, pour chaque $i \in B$, nous avons $R(i) = v$. Donc, $B \subseteq Z(R,v)$, alors $I(B) \leq I(Z(R,v))$ et $I(C \setminus B) \leq I(Z(R,1 - v))$, par le lemme 2. Ceci implique $q(v) \cdot I(Z(R,v)) > q(1 - v) \cdot I(Z(R,1 - v))$, donc $OPT(R) = v$. On a donc $OPT(R) = v$ pour chaque $R \in \rho(B,v)$, ce qui implique que $(B,v) \in Exact(OPT)$.

Cas 2: $q(v) \cdot I(B) = q(1 - v) \cdot I(C \setminus B)$ et $v = 0$ Soit $R \in \rho(B,0)$. Donc, pour chaque $i \in B$ nous avons $R(i) = 0$. Donc, $B \subseteq Z(R,0)$ et alors $I(B) \leq I(Z(R,0))$. Ceci implique

$q(0) \cdot I(Z(R,0)) \geq q(1) \cdot I(Z(R,1))$, donc $OPT(R) = 0$. On a donc $OPT(R) = 0$ pour chaque $R \in \rho(B,0)$, ce qui implique que $(B,0) \in Exact(OPT)$.

Cas 3: $q(v) \cdot I(B) = q(1-v) \cdot I(C \setminus B)$ et $v = 1$ Par le Cas 2, nous avons $CD(B,v) \in Exact(OPT)$. Donc $(B,v) \notin Exact(OPT)$ par le lemme 1.

Cas 4: $q(v) \cdot I(B) < q(1-v) \cdot I(C \setminus B)$ Par le Cas 1, nous avons $CD(B,v) \in Exact(OPT)$. Donc $(B,v) \notin Exact(OPT)$ par le lemme 1.

Fin de la preuve du lemme 3.

Théorème 1 : La stratégie OPT est optimale dans un réseau où $P = \{p_1,p_2,...,p_k\}$ *tel que* $p_i \leq 0{,}5, \forall i = 1,2,...,k$

Preuve du théorème 1 :

Soit D une stratégie de décision. Il faut montrer $Fi(OPT) \geq Fi(D)$.

Soit $(B,v) \in U$. Par le lemme 1, au plus un des événements (B,v) et $CD(B,v)$ peut être élément de $Exact(D)$. Par le lemme 3, pour chaque couple (B,v) et $CD(B,v)$, celui avec la probabilité la plus grande ou égale est dans $Exact(OPT)$. Donc $Fi(OPT) = P(Exact(OPT)) \geq P(Exact(D)) = Fi(D)$.

Fin de la preuve du Théorème 1.

6.5 Généralisation des probabilités de pannes

Dans les sections précédentes, nous avons considéré le cas des probabilités de pannes des chemins inférieures à 50%. Le cas où les probabilités de pannes sont arbitraires a été traité dans un article de conférence que nous avons écrit [14]. En effet, il suffit d'utiliser la stratégie présentée dans ce chapitre, tout en ignorant les chemins dont la fiabilité est inférieure à 50%.

L'intuition derrière cette solution est que les informations reçues à travers les chemins dont la fiabilité est inférieure à 50% est plus (en cas du pire comportement de ces chemins) probablement une information mensongère que celle transmise. Cette information est donc simplement sans valeur.

6.6 Notes d'application

La stratégie de vote est conçue afin de fonctionner avec des paramètres de fiabilité fournis par des couches supérieures. De plus, l'exécution de OPT doit se faire au moment où tous les messages sont réputés reçus. Les aspects des temps limite de réception des messages, de réception des messages et de calcul des probabilités de pannes des processus sont gérés par les couches au dessus de la stratégie OPT.

Donc, définissons une couche supérieure à OPT, nommée OPT*. Les tâches devant être accomplies par OPT* sont:

1. réception des messages;
2. calcul des probabilités de panne de chaque processus au moment de la réception de son message;
3. gestion des temps limite de réception des messages;
4. exécution de la stratégie OPT.

6.7 Conclusion

Dans ce chapitre, nous avons développé une stratégie optimale de réception des messages pour des probabilités de pannes des chemins entrants au récepteur $p_i \leq 0{,}5$. Ensuite, nous avons référé à notre article [14] qui généralise cette stratégie pour le cas des probabilités arbitraires de pannes des chemins.

Nous proposons le problème suivant: Considérons un état d'alarme (0 = "pas d'alarme" et 1 = "alarme") envoyé à un processeur par k chemins disjoints dont les probabilités de pannes byzantines sont arbitraires et connues. Dans ce cas, la situation d'erreur de décision n'est pas symétrique. Il est bien moins dangereux de déclencher l'alarme si ce n'est pas nécessaire que de ne pas le faire si c'est nécessaire. On veut donc s'assurer que la nécessité de déclencher l'alarme sera négligée avec une très petite probabilité (par exemple au plus 0,01%) et trouver la stratégie optimale de décision seulement parmi ces stratégies ultra sécuritaires. Nous voulons donc trouver la stratégie optimale de décision (déclencher l'alarme ou pas) qui permet de bien interpréter l'état "alarme" dans au moins A% des cas (état d'alarme = 1).

Conclusion

Le but de ce mémoire a été de répondre à deux questions fondamentales de la tolérance aux pannes byzantines dans un modèle de distribution probabiliste où les pannes des composants sont indépendantes. Rappelons que le choix du modèle probabiliste a été fait car selon beaucoup d'auteurs il modélise le mieux la réalité des systèmes d'information.

Algorithme de diffusion

Au chapitre 4, nous avons développé l'algorithme de diffusion de messages DRPB qui est rapide et presque certain. Cet algorithme utilise un nombre de messages d'ordre optimal. Nous avons vu que pour effectuer une transmission de données presque certaine dans un environnement avec des pannes permanentes, il est nécessaire de transmettre le message à travers un nombre logarithmique de liens.

La question suivante n'a pas été considérée dans le cadre des recherches et est proposée pour la recherche future: Considérons un hypercube de n noeuds dont les liens et les noeuds tombent en panne avec probabilités respectives $p < 0{,}5$ et $q < 0{,}5$. Les communications dans l'hypercube se font de manière *synchrone*, en mode *port unique bidirectionnel*. Trouver un algorithme qui effectue la diffusion avec fiabilité au moins $1 - \frac{1}{n^\epsilon}$ dans ce réseau avec un nombre de messages et temps d'ordre optimal. De plus, trouver la relation entre ϵ et les probabilités de pannes p et q.

L'hypercube est une structure intéressante [2] qui permet d'effectuer la diffusion avec forte probabilité de succès. Dans l'hypercube, exactement $\log n$ liens relient chaque noeud à leurs $\log n$ voisins.

Stratégie de vote optimale

Au chapitre 6, nous avons développé une stratégie optimale de réception des messages pour des probabilités de pannes des chemins entrants au récepteur $p_i \leq 0{,}5$. Ensuite, nous avons référé à notre article [14] qui généralise cette stratégie au cas des probabilités de pannes arbitraires.

Le problème qui a été abordé dans ce chapitre était de trouver la stratégie de décision qui optimisait la probabilité d'une décision correcte. Par contre, dans certains contextes, le risque associé à l'interprétation optimiste des données est trop élevé. Par exemple, dans un réacteur nucléaire, il est plus dangereux d'ignorer une fausse alerte que d'appliquer les procédures d'urgence associées à cette alerte dans le cas où ce n'est pas nécessaire.

La question suivante n'a pas été considérée dans le cadre de ce mémoire et est proposée pour la recherche future: Considérons un état d'alarme (0 = "pas d'alarme" et 1 = "alarme") envoyé à un processeur par k chemins disjoints dont les probabilités de pannes byzantines sont arbitraires et connues. Nous voulons trouver la stratégie optimale parmi celles qui donnent une garantie d'au moins $A\%$ qu'une nécessité d'alarme ne soit pas négligée.

Annexe A

Algorithme de simulation DRPB

Le code qui suit a été utilisé pour simuler le fonctionnement de l'algorithme DRPB dans un réseau affligé de pannes.

```
// ArbreBinaireEpais.cpp : Defines the entry polong for the console application.

//pour effectuer une simulation où le nombre de simulations est respecté
//selon la structure du réseau tel que développée dans le mémoire, il
//est utile d'arrêter la simulation à l'instant où un noeud vote
//le mauvais message. Donc, si cet événement ne se produit pas, tous
//les noeuds ont le bon message en mémoire. Les bons noeuds envoient le
//bon message, les mauvais noeuds envoient le mauvais message. Les
//bons liens envoient le message tel que transmis par le noeud et les
//mauvais liens envoient le mauvais message.

//Après la routine RACINE PROPAGATION, définir trois supernoeuds
//Parent, EnfantGauche, EnfantDroite. On comptera le nombre
//de transmissions entre supernoeuds afin de déterminer quand arrêter les
//simulations.


#include "stdafx.h"
long NbGroupes = 100;
long NbNoeudsParGroupe = 100;
long NbEssais = -1;
double p = 0.1;
double q = 0.2;
char *NomFichier = "log.txt";
long NbNoeuds = -1;

/*struct ValeursNoeuds{
```

```
bool Valeur[NbNoeudsParGroupe];
};

struct SuperNoeudEnfant{
//Un supernoeud est composé de clogn noeuds
//qui ont chacun un tampon long de clogn positions
ValeursNoeuds Liste[NbNoeudsParGroupe];
};*/


bool VOTER(bool Message, bool **BUFF){
//effectuer un vote sur le tampon d'un seul noeud
bool echec = false;
long votefalse=0, votetrue=0;
for(long Compteur=0;Compteur<NbNoeudsParGroupe;Compteur++){
if (BUFF[0][Compteur]){
votetrue++;
}
else{
votefalse++;
}
}

if (Message){
if (votetrue > votefalse)
echec = false;
else
echec = true;
}
else{
if (votefalse > votetrue)
echec = false;
else
echec = true;
}
return echec;
}

bool RACINE_Transmission(bool message, bool *BonNoeud,
bool **EnfantGauche, bool **EnfantDroite){

bool ECHEC = false;

//l'envoi a partir du supernoeud racine a ses enfants doit
```

```
//être géré à part pour préserver l'état de faute des noeuds
for(long CompteurNoeuds=0;CompteurNoeuds<NbNoeudsParGroupe;CompteurNoeuds++){
for(long Decalage=0;Decalage<NbNoeudsParGroupe;Decalage++){
if (BonNoeud[CompteurNoeuds] && ((float)rand()/(float)RAND_MAX>p)){
//le noeud et le lien sont bons
EnfantGauche[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = message;
}
else{//sinon...
EnfantGauche[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = !message;
}
if (BonNoeud[CompteurNoeuds] && ((float)rand()/(float)RAND_MAX>p)){
//le noeud et le lien sont bons
EnfantDroite[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = message;
}
else{//sinon...
EnfantDroite[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = !message;
}
}
}
for(CompteurNoeuds=0;CompteurNoeuds<NbNoeudsParGroupe;CompteurNoeuds++){
ECHEC = (ECHEC || VOTER(message, &EnfantGauche[CompteurNoeuds]));
ECHEC = (ECHEC || VOTER(message, &EnfantDroite[CompteurNoeuds]));
}

return ECHEC;
}


bool GROUPE_Transmission(bool message, bool **EnfantGauche, bool **EnfantDroite){

bool BonNoeud;

bool ECHEC=false;

//l'envoi a partir du supernoeud racine a ses enfants doit
//être géré à part pour préserver l'état de faute des noeuds
for(long CompteurNoeuds=0;CompteurNoeuds<NbNoeudsParGroupe;CompteurNoeuds++){
BonNoeud = ((float)rand()/(float)RAND_MAX>q); //est-ce que ce noeud est bon?
for(long Decalage=0;Decalage<NbNoeudsParGroupe;Decalage++){
if (BonNoeud && ((float)rand()/(float)RAND_MAX>p)){
```

```
//le noeud et le lien sont bons
EnfantGauche[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = message;
}
else{//sinon...
EnfantGauche[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = !message;
}
if (BonNoeud && ((float)rand()/(float)RAND_MAX>p)){
//le noeud et le lien sont bons
EnfantDroite[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = message;
}
else{//sinon...
EnfantDroite[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = !message;
}
}
}
for(CompteurNoeuds=0;CompteurNoeuds<NbNoeudsParGroupe;CompteurNoeuds++){
ECHEC = (ECHEC || VOTER(message, &EnfantGauche[CompteurNoeuds]));
ECHEC = (ECHEC || VOTER(message, &EnfantDroite[CompteurNoeuds]));
}

return ECHEC;
}



bool RACINE_Propagation(bool message, bool *BonNoeud, bool **RACINE){
//Nous simulons ici la propagation à partir d'un noeud SOURCE
//qui contient le message jusqu'aux autres noeuds de RACINE
//le tableau BonNoeud contiendra l'état de faute des noeuds de
//RACINE simplement puisqu'il faudra simuler le transfert à partir
//de ces noeuds jusqu'aux noeuds des supernoeuds enfants par la suite


bool ECHEC = false; //si un échec survient, cette variable doit
//devenir vraie.
BonNoeud[0] = true; //SOURCE est un bon noeud par définition

//SuperNoeudEnfant RACINE;

for(long CompteurNoeuds=1;CompteurNoeuds<NbNoeudsParGroupe;CompteurNoeuds++){
```

```
//définir si les noeuds du supernoeud RACINE sont bons ou mauvais
BonNoeud[CompteurNoeuds] = ((float)rand()/(float)RAND_MAX>q);

//Transmissions de SOURCE aux autres noeuds de RACINE
if ((float)rand()/(float)RAND_MAX>p) //vérifier
RACINE[CompteurNoeuds][0] = message;
else
RACINE[CompteurNoeuds][0] = !message;
}//fin du FOR


//Transmissions à partir des noeuds de RACINE vers les
//noeuds de RACINE
for(CompteurNoeuds=1;CompteurNoeuds<NbNoeudsParGroupe;CompteurNoeuds++){
for(long Decalage=1;Decalage<NbNoeudsParGroupe;Decalage++){
if (BonNoeud[CompteurNoeuds] && ((float)rand()/(float)RAND_MAX>p))
//si le noeud et le lien sont bons
RACINE[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = RACINE[CompteurNoeuds][0];
else//sinon...
RACINE[(CompteurNoeuds+Decalage) % NbNoeudsParGroupe]
[CompteurNoeuds] = !message;
}
}

//effectuer tous les votes et conserver un échec
for(CompteurNoeuds=1;CompteurNoeuds<NbNoeudsParGroupe;CompteurNoeuds++)
ECHEC = (ECHEC || VOTER(message, RACINE));

return ECHEC;
}




long main(long argc, char* argv[])
{
srand((unsigned)time(NULL));
ofstream Fichier;
double FichierPresent;

char **arguments;
char **valeurs;
```

```
char *stopstring;
arguments = new char*[argc];
valeurs = new char*[argc];
unsigned long j;
for (long i=0; i<argc; i++){
arguments[i] = new char[4];
valeurs[i] = new char[strlen(argv[i])-3];
strncpy(arguments[i],argv[i],3);
arguments[i][3] = '\0';
for(j=0;j<strlen(argv[i])-3;j++){
valeurs[i][j]=argv[i][j+3];
}

if (strcmp(arguments[i],"-N:") == 0){
NbNoeuds = strtol(valeurs[i], &stopstring, 10);
}

else if (strcmp(arguments[i],"-n:") == 0){
NbNoeudsParGroupe = strtol(valeurs[i], &stopstring, 10);
}

else if (strcmp(arguments[i],"-g:") == 0){
NbGroupes = strtol(valeurs[i], &stopstring, 10);
}

else if (strcmp(arguments[i],"-e:") == 0){
NbEssais = strtol(valeurs[i], &stopstring, 10);
}

else if (strcmp(arguments[i],"-p:") == 0){
p = strtod(valeurs[i], &stopstring);
}
else if (strcmp(arguments[i],"-q:") == 0){
q = strtod(valeurs[i], &stopstring);
}
else if (strcmp(arguments[i],"-f:") == 0){
strcpy(NomFichier,valeurs[i]);
}
}
struct _finddata_t InfoFichier;
FichierPresent = _findfirst(NomFichier,&InfoFichier);
if (FichierPresent == -1){//le fichier est nouveau, le créer
//avec son entête
Fichier.open(NomFichier,ios::out,filebuf::sh_none);
```

```
Fichier<<"#essais\t#échecs\tp\tq\t#noeuds/groupe\t#groupes\tclogn"<<endl;
}
else{//le fichier existe, ajouter les résultats à la fin
Fichier.open(NomFichier,ios::app,filebuf::sh_none);
}

double qq = (1-q)*pow((1-p),2);
double c = 40*log(2)/pow((2*qq-1),2);
double c_log_n = ceil(c*log10(NbGroupes*NbNoeudsParGroupe));
if (NbNoeuds > 0){
NbNoeudsParGroupe = (long)ceil(c*log10(NbNoeuds));
c_log_n = NbNoeudsParGroupe;
NbGroupes = (long)ceil((double)NbNoeuds/(double)c_log_n);
}
else{
NbNoeuds = NbGroupes*NbNoeudsParGroupe;
}
if (NbEssais == -1){
NbEssais = NbNoeuds*2;
}
cout<<"Nb Noeuds "<<NbNoeuds<<endl;
cout<<"Nb Noeuds Par Groupe "<<NbNoeudsParGroupe<<endl;
cout<<"Nb Groupes "<<NbGroupes<<endl;
cout<<"Nb Essais "<<NbEssais<<endl;
cout<<"probabilite de panne des liens "<<p<<endl;
cout<<"probabilite de panne des noeuds "<<q<<endl;
cout<<"enregistrement dans le fichier "<<NomFichier<<endl;

//Le supernoeud parent ne doit pas être modélisé puisqu'à
//tout moment dans la simulation on sait que les valeurs dans
//ce noeud sont bonnes puisque si les valeurs dans les noeuds
//enfants sont mauvaises la simulation sera interrompue.
const grosseur = NbNoeudsParGroupe;

bool **Tableau1;
Tableau1 = new bool*[grosseur];
for (long index=0;index<grosseur;index++)
Tableau1[index]=new bool[grosseur];


bool **Tableau2;
Tableau2 = new bool*[grosseur];
for (index=0;index<grosseur;index++)
Tableau2[index]=new bool[grosseur];
```

```
long NombreGroupes;
long NombreEchecs = 0;

bool ECHEC = false; //échec de la simulation si un noeud vote !M

bool *BonNoeud;
BonNoeud = new bool[NbNoeudsParGroupe];
//choix du message initial
bool M;

for(long NombreEssais=0; NombreEssais<NbEssais;NombreEssais++){

printf("Essai %d\n",NombreEssais+1);

NombreGroupes=0;
ECHEC = false;
if ((float)rand()/(float)RAND_MAX < 0.5){
M=true;
}
else{
M=false;
}
//de SOURCE aux autres noeuds de RACINE
ECHEC = (ECHEC || RACINE_Propagation(M, BonNoeud, Tableau1));

//des noeuds de RACINE aux enfants de RACINE
ECHEC = (ECHEC || RACINE_Transmission(M, BonNoeud, Tableau1, Tableau2));

NombreGroupes = 3;

while (!ECHEC && NombreGroupes < NbGroupes){
//autres transmissions de groupe.
ECHEC = (ECHEC || GROUPE_Transmission(M, Tableau1, Tableau2));
NombreGroupes = NombreGroupes+2;

printf("\r%.1f%%",(float)NombreGroupes/NbGroupes*100);
}
if (ECHEC){
printf("\rECHEC: essai %d\n",NombreEssais+1);
NombreEchecs++;
}
else
```

```
printf("\rSUCCES: essai %d\n",NombreEssais+1);
}


Fichier<<NbEssais<<"\t"<<NombreEchecs<<"\t"<<p<<"\t"<<q<<"\t"<<NbNoeudsParGroupe<<"\t"
<<NbGroupes<<"\t"<<c_log_n<<endl;

//cout<<NombreEchecs<<" echecs pour "<<NbEssais<<" essais"<<endl;
printf("%i echecs pour %i essais", NombreEchecs, NbEssais);
Fichier.close();
return ECHEC;
}
```

Bibliographie

[1] P. Berman, K. Diks, A. Pelc, *Reliable Broadcasting in Logarithmic Time with Byzantine Link Failures.*, Journal of Algorithms 22, pp. 199-211, 1997.

[2] B. S. Chlebus, K. Diks, A. Pelc, *Reliable Broadcasting in Hypercubes with Random Link and Node Failures.*, Combinatorics, Probability and Computing 5, pp. 337-350, 1996.

[3] B. S. Chlebus, K. Diks, A. Pelc, *Sparse Networks Supporting Efficient Reliable Broadcasting.*, Nordic Journal of Computing 1, pp. 332-345, 1994.

[4] A. Dahbura, M. Barborak, M. Malek. *The consensus problem in fault-tolerant computing.* ACM Computer Surv., 25:pp. 171-220, 1993.

[5] K. Diks, A. Pelc. *Optimal adaptative broadcasting with a bounded fraction of faulty nodes.* Algorithmica 28:pp. 37-50, 2000.

[6] H. Garcia-Molina, D. Barbara. *Optimizing the reliability provided by voting mechanisms.* Proc. 4th Int. Conf. Distributed Computing Systems, pp. 340-346, 1984.

[7] H. Garcia-Molina, D. Barbara. *How to assign votes in a distributed system.* Journal of the ACM, Vol.32, No.4, pp. 841-860, October 1985.

[8] L. Gargano, A. L. Liestman, J. G. Peters, D. Richards, *Reliable Broadcasting.*, Discrete Applied Mathematics 53, pp. 135-148, 1994.

[9] L. Gąsieniec, A. Pelc, *Adaptive broadcasting with faulty nodes.*, Parallel Computing 22, pp. 903-912, 1996.

[10] L. Gąsieniec, A. Pelc, *Broadcasting with a Bounded Fraction of Faulty Nodes.*, Journal of Parallel and Distributed Computing 42, pp. 11-20, 1997.

[11] L. Gąsieniec, A. Pelc, *Broadcasting with linearly bounded transmission faults.*, Discrete Applied Mathematics 83, pp. 121-133, 1998.

[12] A. L. Liestman. *Fault-tolerant broadcast graphs.* Networks 15, pp. 159-171, 1985.

[13] M. Matsumoto, T. Nishimura. *Mersenne Twister A 623-dimensionally equidistributed uniform pseudorandom number generator.* ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, pp.3-30, 1998.

[14] M. Paquette, A. Pelc. *Optimal Decision Strategies in Byzantine Environments.* Proc. 11th International Colloquium on Structural Information and Communication Complexity (SIROCCO), pp. 245-254, 2004.

[15] M. Paquette, A. Pelc. *Fast Broadcasting with Byzantine Faults.* Proc. 7th IFAC Symposium on Cost Oriented Automation, pp. 311-316, 2004.

[16] M. Pease, L. Lamport, R. Shostak. *The byzantine generals problem.* ACM Transactions on Programming Languages and Systems, 4(3):pp. 382-401, 1982.

[17] A. Pelc. *Fault-Tolerant Broadcasting and Gossiping in Communication Networks.*, Networks 28, pp. 143-156, 1996.

[18] M. Spasojevic, P. Berman. *Voting as the Optimal Static Pessimistic Scheme for Managing Replicated Data.* IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 1, pp. 64-73, 1994.

[19] J. Tang, N. Natarajan. *A static pessimistic scheme for handling replicated databases.* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989.

[20] Z. Tong, R. Y. Kain. *Vote assignment in weighted voting mechanisms.* IEEE Transactions on computers, Vol. 40, No. 5, pp. 664-667, 1991.