Sauvegarde des données dans les réseaux P2P

Rabih Tout

N° d'ordre : **96 - 2010** Année 2010

THESE DE L'UNIVERSITE DE LYON

Délivrée par

L'UNIVERSITE CLAUDE BERNARD LYON1

ECOLE DOCTORALE INFORMATIQUE ET MATHEMATIQUES

Pour l'obtention du diplôme de

« DOCTORAT EN INFORMATIQUE »

(arrêté du 7 août 2006)

présentée et soutenue publiquement le **25 juin 2010**

par

M. TOUT Rabih

SAUVEGARDE DES DONNÉES DANS LES RÉSEAUX P2P

Directeur de thèse : Pr. GHODOUS Parisa

Composition du Jury :

Rapporteurs : M. HAMEURLAIN Abdelkader, Pr. à l'Université Paul Sabatier.
M. YETONGNON Kokou, Pr. à l'Université de Bourgogne.

Examinateurs : M. AIT AMEUR Yamine, Pr. à l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique.
M. LBATH Ahmed, Pr. à l'Université Joseph Fourier.
M. BENSLIMANE Djamal, Pr. à l'Université Claude Bernard Lyon1.
Mme GHODOUS Parisa, Directrice de thèse, Pr. à l'Université Claude Bernard Lyon1.
M. Tanasoiu Mihai, Responsable industriel, Directeur général Alter Systems.

À mes parents, Naïm et Wafa

À mes sœurs, Manal et Maya

Remerciements

Je tiens tout d'abord à remercier chaleureusement ma directrice de thèse, Madame Parisa GHODOUS pour l'aide compétente qu'elle m'a apportée et pour ses conseils qui m'ont été très précieux pour structurer et améliorer la qualité de mon travail. J'ai été particulièrement touché par son encouragement et sa disponibilité tout au long de mon travail de thèse. Elle a su me laisser la liberté nécessaire pour l'accomplissement de mes travaux, tout en y gardant un œil critique et avisé.

Cette thèse s'est déroulée dans le cadre d'une convention CIFRE. Je tiens à remercier l'Association Nationale de la Recherche et de la Technologie (ANRT) et la société Alter Systems, le partenaire industriel, pour le financement de cette thèse.

Je remercie très chaleureusement mon père Naïm qui m'a accompagné tout au long de la thèse et avec qui j'ai partagé des moments agréables. Il a toujours été présent pour m'encourager et partager avec moi les idées scientifiques. Je le remercie également pour la relecture de ma thèse et pour les corrections et les conseils qu'il m'a apportés.

J'adresse également ma reconnaissance au directeur et directeur adjoint du Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS), de m'avoir accueilli dans ce laboratoire ainsi que pour la disposition des moyens nécessaires au bon déroulement du travail au sein du laboratoire.

Je remercie Monsieur Abdelkader Hameurlain, Professeur à l'université Paul Sabatier et Monsieur Kokou Yetongnon, Professeur à l'université Paul Cézanne, d'avoir accepté de rapporter mon travail et je suis très reconnaissant pour leur temps consacré à ce travail ainsi que pour leurs remarques et leurs suggestions. J'adresse mes remerciements à Mesdames et Messieurs les membres du jury d'avoir accepté d'évaluer et de juger mon travail.

Je remercie aussi les membres de l'équipe Service Oriented Computing avec lesquels j'ai pu partager des moments agréables de recherche mais aussi de détente.

Ce mémoire est la trace de quatre années de travail mais aussi quatre années de vie. Je tiens à remercier tous ceux qui m'ont accompagné et encouragé pendant ce temps, en particulier mes amis : Sarah, Ouaiss, Youssef, Yahya, Nader, Sahar, Mohamad, Mathilde,

Laetitia, Isabelle, Osman, Samer, Hicham, Achraf, Nihad et tous ceux qui méritent d'être remerciés, ils se reconnaitront…

Je remercie évidemment ma famille pour son irremplaçable et inconditionnel soutien. Un merci plein d'amour et d'affection d'être restée si près de moi, malgré les distances. Cette thèse est un peu la leur aussi. Elle représente l'aboutissement du soutien et des encouragements que mes parents m'ont accordés tout au long de ma vie.

Aujourd'hui la sauvegarde des données est une solution indispensable pour éviter de les perdre. Plusieurs méthodes et stratégies de sauvegarde existent et utilisent différents types de support. Les méthodes de sauvegarde les plus efficaces exigent souvent des frais d'abonnement au service liés aux couts du matériel et de l'administration investis par les fournisseurs. Le grand succès des réseaux P2P et des applications de partage de fichiers a rendu ces réseaux exploitables par un grand nombre d'applications surtout avec la possibilité de partager les ressources des utilisateurs entre eux. Les limites des solutions de sauvegarde classiques qui ne permettent pas le passage à l'échelle ont rendu les réseaux P2P intéressants pour les applications de sauvegarde.

L'instabilité dans les réseaux P2P due au taux élevé de mouvement des pairs, rend la communication entre les pairs très difficile. Dans le contexte de la sauvegarde, la communication entre les nœuds est indispensable ce qui exige une grande organisation dans le réseau. D'autre part, la persistance des données sauvegardées dans le réseau reste un grand défi car la sauvegarde n'a aucun intérêt si les données sauvegardées sont perdues et que la restauration devient impossible.

L'objectif de notre thèse est d'améliorer l'organisation des sauvegardes dans les réseaux P2P et de garantir la persistance des données sauvegardées. Nous avons donc élaboré une approche de planification qui permet aux nœuds de s'organiser dans le but de mieux communiquer entre eux. D'autre part, pour garantir la persistance des données sauvegardées, nous avons proposé un approche de calcul probabiliste qui permet de déterminer, selon les variations dans le système, le nombre de répliques nécessaire pour qu'au moins une copies persiste dans le système après un certain temps défini. Nos approches ont été implémenté dans une application de sauvegarde P2P.

TITRE en anglais

Data Backup in P2P Networks

RESUME en anglais

Nowadays, data backup is an essential solution to avoid losing data. Several backup methods and strategies exist. They use different media types. Most efficient backup methods are not free due to the cost of hardware and administration invested by suppliers. The great success of P2P networks and file sharing applications has encouraged the use of these networks in multiple applications especially with the possibility of sharing resources between network users. The limitations of traditional backup solutions in large scale networks have made P2P networks an interesting solution for backup applications.

Instability in P2P networks due to peers' high movement rate makes communication between these peers very difficult. To achieve data backup, communications between peers is essential and requires a network organization. On the other hand, the persistence of backed up data in the network remains a major challenge. Data backup is useless if backed up copies are lost.

The objective of this thesis is to improve the backup organization and ensure backed up data persistence in P2P networks. We have therefore developed a planning approach that allows nodes to organize themselves in order to better communicate with each other. On the other hand, to ensure data persistency, we proposed a probabilistic approach to compute the minimum replicas number needed for a given data so that at least one copy remains in the system after a given time. Our two approaches have been implemented in a P2P backup application.

MOTS-CLES

Peer-to-Peer, Backup, Network clustering, Scheduling, Redundancy, Persistence, Erasure coding, Probabilistic guarantee.

Laboratoire LIRIS – Laboratoire d'InfoRmatique en Image et Systèmes d'information
Université Claude Bernard Lyon1 - Bâtiment Nautibus Campus de la Doua - 8, Bd Niels Bohr
69622 Villeurbanne Cedex

Table des matières

Liste des Figures

Introduction

Aujourd'hui, l'informatique est devenue indispensable dans beaucoup de domaines. La saisie des données sur support informatique facilite le stockage, l'archivage et le transport des données. Pour éviter la perte de données, la sauvegarde est une solution indispensable. Selon les caractéristiques et l'usage, nous avons le choix entre différentes solutions de sauvegarde. Certaines solutions sont gratuites, d'autres payantes. Les techniques de sauvegarde les plus efficaces sont celles qui stockent les données loin de la source originale. Dans ce genre de sauvegarde, une connexion est établie avec un serveur distant sur lequel les données sont copiées. Les données sauvegardées sont souvent répliquées ce qui permet d'augmenter le niveau de sécurité. Ce genre de solution exige des frais pour les fournisseurs tels que les prix des serveurs dédiés à la sauvegarde et le prix de l'administration centrale (comptes clients, sécurité, sauvegarde, surveillance, répliques, redondance, …). Ces frais imposent que ce genre de technique soit facturé aux clients en quantité de données sauvegardées. Aujourd'hui, le volume de données stockées devient de plus en plus important et une solution de sauvegarde payante ne semble pas pratique pour la plupart des utilisateurs.

La solution de sauvegarde gratuite qui se présente aujourd'hui est celle basée sur les réseaux P2P qui ont déjà connu un grand succès avec la sortie des applications de partage de fichiers qui a contribué à augmenter considérablement le nombre de personnes se connectant à ces réseaux. Aujourd'hui, on observe une admiration pour les systèmes de fichiers P2P qui permettent de partager des ressources à large échelle. Le succès des réseaux P2P a entrainé le développement de plusieurs applications basées sur ce type de réseaux. Parmi ces applications, on observe les applications de sauvegarde de données dont le principe est de remplacer les serveurs dédiés à la sauvegarde par l'espace disque non utilisé par les utilisateurs du réseau de sauvegarde. Chaque utilisateur va pouvoir stocker ses propres données sur des serveurs distants. Ces serveurs sont les ordinateurs des autres utilisateurs.

Le principe de la sauvegarde P2P consiste à crypter d'abord les données à sauvegarder pour les découper ensuite en blocs qui seront synchronisés à travers le réseau. Pour assurer la redondance des données, il faudra créer plusieurs copies de ces données.

Problématique

Les problèmes de sauvegarde de données dans les réseaux P2P sont nombreux et vastes à résoudre, mais nous restreindrons notre étude à deux petites parties de cette problématique qui sont l'organisation de la sauvegarde et la persistance des données. Ces deux problèmes sont très importants et jouent un rôle fondamental dans le cycle de sauvegarde ainsi que sur les coûts et la qualité des sauvegardes effectuées.

Les réseaux P2P sont instables à cause des connexions et déconnexions que l'on peut avoir dans ce genre de réseaux. Chaque nœud du réseau peut rejoindre ou quitter le système à tout moment sans prévenir les autres nœuds. Si ce départ a lieu au moment où un autre nœud est entrain de copier ou de télécharger des données, ce dernier va sûrement perdre la connexion et être obligé de tout reprendre. On est donc face à un grand problème d'organisation car on est incapable de contrôler la présence des nœuds dans le réseau. Ce problème d'organisation peut être acceptable dans le cas de téléchargement de fichiers multimédias car ce type de téléchargement n'est pas urgent. Par contre si un utilisateur se trouve incapable de finaliser sa sauvegarde ou sa restauration à cause de l'instabilité du réseau, ceci devient inacceptable. On ne peut pas attendre plusieurs jours pour effectuer une sauvegarde. À chaque instant, on peut être confronté à des risques de perte de données.

La plupart des systèmes de partage de fichiers multimédias utilisent une stratégie presque identique qui consiste à supprimer du réseau les fichiers les moins téléchargés. Cette technique permet d'éliminer les fichiers qui ne sont plus demandés par les utilisateurs du système. Par exemple, un ancien film qui n'est plus demandé ou téléchargé par les utilisateurs du système peut être supprimé sans aucun problème. Ceci est fait dans le but de libérer de l'espace mémoire afin de pouvoir stocker de nouveaux fichiers. Il suffit de traquer les taux de téléchargements pour prendre de telles décisions. Cela montre bien qu'on n'a pas un espace de stockage infini. Cet espace est finalement les parties des disques non utilisées par les utilisateurs. Dans le cas des applications de partage de fichiers multimédias, ça ne pose pas de problème si on supprime un ancien film car les utilisateurs réclament toujours les fichiers les plus récents. Dans le contexte de sauvegarde, cette technique ne peut pas être appliquée car les données sauvegardées ne sont téléchargées que par leur propre utilisateur. Le téléchargement ne se produit pas souvent mais uniquement en cas d'une restauration due à une perte des données. Un autre point important est la persistance des données dans le système. Pour les applications de partage de fichiers, la perte d'un ou plusieurs fichiers ne pose aucun problème alors que dans un contexte de sauvegarde, la perte d'une sauvegarde peut coûter très cher. Les sauvegardes doivent persister dans le système pour être utilisées en

cas de besoin. Garantir cette persistance est un grand défi dans ce genre de réseaux. Comment peut-on garantir que les nœuds sur lesquels nous avons répliqué nos données, vont rester accessibles ? Une réplication massive sur un grand nombre de nœuds peut être une solution fiable mais pas très pratique à cause des limites de l'espace mémoire qui doit être utilisé pour assurer les sauvegardes de tous les nœuds du système. Même si la plupart des disques disposent aujourd'hui d'un grand espace de stockage, l'utilisateur n'est souvent pas prêt à proposer un grand pourcentage de son disque pour stocker les données des autres utilisateurs. Il faut ajouter à tous ces problèmes l'aspect de sécurité [**1**] [**2**], le routage [**3**] [**4**] [**5**] , la recherche des données [**6**] et d'autres problèmes qu'on peut rencontrer dans ce genre de réseaux.

Objectifs

L'organisation de la sauvegarde des données dans les réseaux P2P est une tâche difficile à accomplir vu le taux de mouvements des nœuds dans le réseau. Beaucoup de requêtes échangées peuvent ne pas aboutir à cause d'un manque d'organisation ou de communication entre les nœuds. Avec la structuration des réseaux P2P [**7**] [**8**], on est capable d'optimiser les requêtes et les temps de réponses. Par contre la structuration classique ne permet pas d'avoir des informations sur le comportement des nœuds dans le réseau. On ne peut pas savoir si un nœud x présent actuellement dans le réseau va l'être après un certain moment.

Notre travail de thèse se situe dans le domaine de l'organisation des sauvegardes et de la persistance des données sauvegardées dans le réseau P2P. L'objectif principal de l'organisation est de permettre aux nœuds de mieux collaborer entre eux ce qui permet d'optimiser la sauvegarde et d'obtenir une meilleure qualité de service. La persistance des données est indispensable pour avoir un système de sauvegarde fiable. Il faut être capable de maintenir les données dans le système même si ce dernier n'est pas stable en prenant en compte que l'espace mémoire peut être limité. Notre objectif est donc de pouvoir garantir la persistance des données dans un réseau P2P.

Méthodologies de notre approche

Pour organiser la sauvegarde des données et aider les nœuds à mieux collaborer entre eux, nous visons à élaborer, entre les nœuds du réseau, une méthodologie d'échange d'informations liées aux comportements des nœuds. Cette méthodologie consiste à échanger

des informations importantes qui vont permettre de déterminer si un nœud est libre ou occupé et de prévoir sa période de présence dans le réseau.

Afin d'atteindre notre objectif, nous avons proposé un système de planification [**9**] permettant à chaque nœud du système de présenter aux autres nœuds, ses propres plannings et de consulter les plannings des autres. Chaque nœud participant au réseau de sauvegarde doit obligatoirement remplir un planning de sauvegarde et un planning de service dans lesquels il indique les moments durant lesquels il souhaite effectuer une sauvegarde de ses propres données et les moments durant lesquels il est libre pour présenter des services aux autres nœuds du système. Ces plannings sont indexés sous forme de clés permettant de faciliter la recherche.

Pour garantir la persistance des données sauvegardées, nous avons proposé un système de calcul probabiliste [**10**] qui consiste à déterminer, en prenant en compte les variations dans le système, le nombre de répliques à effectuer de chaque bloc de manière à ce que ce nombre soit largement suffisant pour qu'au moins une copie persiste dans le système après un certain temps.

Nous avons validé nos approches par le développement d'une application de sauvegarde. Notre application gère la planification et la persistance des données dans le réseau.

Ce travail de thèse a été mené dans le cadre d'une Convention Industrielle de Formation par la Recherche (CIFRE[1]) au sein de l'équipe SOC[2], *Service Oriented Computing*, du laboratoire LIRIS[3] (Laboratoire d'Informatique en Image et Systèmes d'information) de Lyon. Le partenaire industriel de la convention CIFRE est la société Alter Systems[4], société de services informatiques et éditeur de logiciels

Organisation de la thèse

[1] http://www.anrt.asso.fr/fr/espace_cifre/accueil.jsp

[2] http://liris.cnrs.fr/soc/

[3] http://liris.cnrs.fr/

[4] http://www.altersystems.fr

Dans le premier chapitre, nous allons présenter une introduction générale sur la sauvegarde des données. Dans ce chapitre, nous abordons les différents types de sauvegarde, les supports et les stratégies utilisées. Nous montrons les différences entre l'architecture classique et celle du P2P.

Dans le deuxième chapitre, nous présentons un état de l'art sur les systèmes de sauvegarde P2P. Nous commençons par une présentation de l'historique des applications de partages de fichiers et nous suivons l'évolution et le passage vers des applications de sauvegarde de données basées sur les réseaux P2P.

Dans le chapitre 3, nous présentons un état de l'art sur la persistance des données. Nous abordons les techniques de redondance des données et présentons en détail quelques systèmes de redondance en expliquant leur fonctionnement. Nous terminons ce chapitre par une présentation de l'efficacité des techniques de redondance.

Dans le quatrième chapitre, nous proposons un système de planification des sauvegardes dans un environnement P2P. Nous présentons en détail différents types de plannings proposés et nous expliquons le fonctionnement de notre approche et du protocole de sauvegarde basé sur cette planification. Les résultats de nos simulations sont présentés à la fin de ce chapitre.

Dans le chapitre 5, nous proposons une approche probabiliste qui utilise les techniques de redondances pour garantir la persistance des données dans un système de sauvegarde à large échelle. Notre approche est basée sur la chaîne de Markov et simule les états réels des nœuds dans un réseau P2P. Nous montrons ensuite comment cette approche peut évoluer en intégrant la réplication classique des données dans le système de redondance.

Dans le chapitre 6, nous présentons en détail l'architecture de notre application de sauvegarde qui a été implémentée au sein de l'entreprise Alter Systems. Nous concluons ce mémoire par un bilan des travaux réalisés suivi des perspectives de travail à réaliser.

Chapitre 1 Sauvegarde des données

1.1 Sauvegarde des données

Aujourd'hui l'usage du PC et d'internet devient indispensable pour toute personne. En utilisant les PC, on peut sûrement stocker un volume important de données. Ce sont désormais des pans de notre existence que nous conservons sur nos disques durs (photos de vacances, films, musiques, courriers électroniques, carnets d'adresses, etc.). Dans le cadre professionnel, les entreprises stockent des données très importantes liées à leurs productions. Il s'agit des applications développées, des codes source, de sites web et même parfois des données liées aux clients grâce aux services fournis. Un simple accident de disque dur, un virus, une surtension ou une mauvaise manipulation peuvent effacer ces données sans grand espoir de les retrouver. Il ne faut pas oublier non plus qu'à ces données s'ajoute la configuration de notre environnement et de nos logiciels. Il suffit d'un simple crash pour se retrouver obligé de tout réinstaller, de mettre les pilotes à jour, de reconfigurer tous ses logiciels, etc. C'est à ce moment là qu'on se rend compte de la masse de travail et du temps investi au quotidien dans l'utilisation de son ordinateur. Et si on arrive après un certain temps à reprendre la même configuration du PC, il n'y a alors aucun recours par rapport aux données perdues.

Il n'y a qu'une seule solution pour éviter que cela se produise. Cette solution miracle est de faire une sauvegarde régulière. L'opération consiste à mettre des données en sécurité de façon à pouvoir les récupérer ou les restaurer en cas de perte. Il vaut mieux prévenir que guérir. La petite perte de temps investie à sécuriser ses fichiers peut compenser une autre, bien plus conséquente, en cas de panne. Pour cela il est important de bien définir les besoins de sauvegarde, les outils à utiliser, le support de stockage et la stratégie de sauvegarde. La technique la plus fréquemment utilisée est la recopie des données sur un support indépendant du système initial. Le choix d'une technique de sauvegarde se fait en prenant en compte la capacité de stockage, la vitesse de sauvegarde, la fiabilité du support, la facilité à restaurer les données et bien sûr le coût de l'ensemble.

Dans la suite nous allons présenter différents supports utilisés dans la sauvegarde, leurs avantages et leurs inconvénients, ainsi que des stratégies de sauvegarde les plus courantes.

1.2 Les supports utilisés dans la sauvegarde

Nous distinguons plusieurs types de support pour stocker les données. Quelle que soit la nature du support, nous avons toujours affaire au même type d'informations qui sont les données informatiques et notre objectif est toujours le même, c'est la protection de ces données pour pouvoir les retrouver en cas de perte. Les différents types de support sont :

1.2.1 La sauvegarde sur disque dur

Cette méthode semble être la plus simple pour sauvegarder les données. Il suffit de recopier ou de dupliquer les données du disque dur principal ou d'une carte mémoire sur un autre disque dur. La sauvegarde sur disque dur peut quelquefois être assistée par un logiciel dédié, ce qui permet de faciliter la tâche et de la faire régulièrement sans risque d'oubli.

- **Avantages :** La facilité de stockage (il suffit de copier les données sur le disque dur sur lequel on désire effectuer la sauvegarde) et la facilité de mise en place (utilisation de disque dur externe avec une simple connexion USB). Cette méthode a un coût faible pour une grande capacité de stockage. Elle est souvent assistée par un logiciel de sauvegarde dédié.
- **Inconvénients :** Un disque dur reste quand même un support mécanique qui peut tomber en panne. Il ne faut pas oublier que les disques durs ne sont pas résistants aux chocs.

1.2.2 Sauvegarde sur CD/DVD

Aujourd'hui, la majorité des ordinateurs sont livrés avec un graveur de CD ou de DVD. On a là une solution simple et économique pour sauvegarder les données. Un CD permet de sauvegarder entre 650 et 700 Mo de données alors qu'un DVD simple couche permet de sauvegarder 4,7 Go et un DVD double couche 8,5 Go.

- **Avantages :** La facilité de stockage pour un faible coût et un faible encombrement.
- **Inconvénients :** Avec le temps, le support CD ou DVD peut se dégrader. Il faudra donc penser à le tester régulièrement et à le protéger des rayures. Un CD ou un DVD gravé sur ordinateur n'a pas la même qualité qu'un CD ou DVD pressé en usine. La taille n'est pas suffisante quand il s'agit de fichiers volumineux ; on a alors besoin de plusieurs DVD.

1.2.3 La sauvegarde sur mémoire flash

Les mémoires flash (clé USB, carte mémoire, …) sont utilisées aujourd'hui dans un très grand nombre d'appareils et sous de nombreux formats. Une mémoire flash peut être utilisée comme un disque dur. Elle ne nécessite pas d'alimentation permanente et sa capacité de stockage est de plus en plus importante. Certaines clés USB font aujourd'hui 64 Go.

- **Avantages :** Les mémoires flash permettent une facilité de stockage, faible coût, faible encombrement, faible consommation d'énergie et rapidité de lecture et d'écriture. Une carte mémoire est assimilée à un disque dur quand elle est connectée à l'ordinateur.
- **Inconvénients :** Une carte mémoire est petite et fragile. Il faut un lecteur dédié au format de la carte-mémoire. Par exemple une carte Memory Stick ne rentre pas dans un lecteur Micro SD. Les nouveaux disques durs à base de mémoire flash (SSD) sont encore très chers.

1.2.4 La sauvegarde des données en ligne

Il existe deux types de sauvegarde en ligne, gratuit et payant. La sauvegarde en ligne est une bonne solution qui nous permet de profiter à faible coût de l'évolution de la technologie de stockage. Il suffit d'envoyer les données à sauvegarder vers un serveur distant et c'est l'hébergeur qui se charge de la maintenance et de la sécurité, il fait migrer les données vers d'autres serveurs pour avoir plusieurs répliques sur des serveurs situés dans des endroits séparés. L'envoi des données peut être réalisé par une connexion FTP ou par internet. Cette méthode reste beaucoup plus efficace que les supports précédents car les données sont stockées sur plusieurs serveurs distants. Dans les cas précédents (disque dur, CD/DVD, clé USB, …), il y a toujours le risque de perdre le support ou que ce dernier tombe en panne. L'idéal est que le support sur lequel se trouvent les sauvegardes soit dans un endroit différent de celui des données originales. La sauvegarde en ligne reste donc la meilleure puisque le serveur est situé loin dans un endroit sécurisé et que plusieurs répliques sont faites sur d'autres serveurs distants.

- **Avantages :** La facilité de stockage depuis n'importe quel ordinateur avec un accès internet. La maintenance est effectuée par l'hébergeur et les infrastructures sont souvent fiables et sécurisées.

- **Inconvénients :** Dans la plupart des cas, cette solution est payante pour l'utilisateur car elle exige des frais pour les fournisseurs (administration et maintenance) qui ne peuvent pas la rendre gratuite pour les clients. Il faut souvent une connexion internet en haut débit pour sauvegarder de gros fichiers.

Dans cette partie, nous avons examiné les différents types de support utilisés dans la sauvegarde des données. Chaque support possède des avantages et des inconvénients. Pour cela, il faut choisir le support le mieux adapté à la situation et au contexte de sauvegarde désiré.

La sauvegarde ne se limite pas uniquement au choix du support de stockage mais aussi à la stratégie qu'il faut employer. Dans la partie qui suit nous allons présenter différentes stratégies de sauvegarde.

1.3 Les stratégies de sauvegarde

Il existe plusieurs méthodes de sauvegarde. La majorité des outils de sauvegarde proposent au moins trois méthodes de sauvegarde auxquelles nous nous intéressons dans la partie suivante. Ces outils peuvent également proposer d'autres méthodes de sauvegarde, mais nous ne présentons que les trois méthodes les plus utilisées.

1.3.1 La sauvegarde complète ou totale

La sauvegarde complète ou totale consiste à sauvegarder toutes les données (répertoires, sous répertoires et fichiers sélectionnés) que celles-ci soient anciennes, modifiées ou non. C'est le type de sauvegarde le plus simple, le plus rapide et le plus précis pour restaurer les données. Cette méthode est la plus lente et consomme beaucoup en terme d'espace disque ce qui empêche son usage pour toutes les sauvegardes à effectuer. D'autres méthodes, plus rapides, permettent de sauvegarder uniquement les données modifiées.

1.3.2 La sauvegarde différentielle

La sauvegarde différentielle permet de sauvegarder les données qui ont été modifiées ou ajoutées depuis la dernière sauvegarde complète. Elle permet de sauvegarder les données plus rapidement qu'avec une sauvegarde complète. Pour effectuer la restauration, il suffit de copier sur disque la dernière sauvegarde complète et la sauvegarde différentielle la plus récente. L'inconvénient de cette méthode est qu'elle ne permet de restaurer que le dernier état d'un

fichier et pas un état se trouvant entre la dernière sauvegarde complète et la sauvegarde différentielle.

1.3.3 La sauvegarde incrémentale

La sauvegarde incrémentale consiste à sauvegarder les données qui ont été modifiées ou ajoutées depuis la dernière sauvegarde complète, différentielle ou incrémentale. Elle permet de sauvegarder les dernières modifications sur les fichiers plus rapidement qu'avec une sauvegarde complète. Cette méthode est plus rapide que la sauvegarde différentielle.

Exemple : Si le jour J, une sauvegarde complète est réalisée, le jour J+2 la sauvegarde incrémentale est réalisée par référence au jour J. Le jour J+3, la sauvegarde est réalisée par rapport au jour J+2, etc.…

Pour effectuer la restauration, nous n'avons besoin que de la sauvegarde incrémentale effectuée le dernier jour. Dans l'exemple précédant, nous aurons besoin uniquement de la sauvegarde incrémentale du jour J+3 (sans les J et J+2) pour restaurer et obtenir la dernière version de la totalité des données.

Dans les entreprises on emploie souvent une stratégie de sauvegarde qui utilise les méthodes précédentes. Voici un exemple des stratégies de sauvegardes les plus répandues :

- **Stratégie 1**

Jour	Type de sauvegarde
Lundi	Sauvegarde incrémentale
Mardi	Sauvegarde incrémentale
Mercredi	Sauvegarde incrémentale
Jeudi	Sauvegarde incrémentale
Vendredi	Sauvegarde complète

La stratégie 1 consiste à effectuer une sauvegarde incrémentale pendant les quatre premiers jours de la semaine (lundi au jeudi) et de faire une sauvegarde complète à la fin de la semaine (vendredi). Cela permet de réduire la quantité de données sauvegardées durant la semaine. Une sauvegarde complète (celle du vendredi dans notre cas), reste indispensable dans le cas du moindre souci avec les versions incrémentales précédentes.

- **Stratégie 2**

Jour	Type de sauvegarde
Lundi	Sauvegarde différentielle
Mardi	Sauvegarde différentielle
Mercredi	Sauvegarde différentielle
Jeudi	Sauvegarde différentielle
Vendredi	Sauvegarde complète

Cette stratégie consiste à effectuer des sauvegardes différentielles du lundi au jeudi et une sauvegarde complète le vendredi.

- **Stratégie 3**

Jour	Type de sauvegarde
Lundi	Sauvegarde complète
Mardi	Sauvegarde complète
Mercredi	Sauvegarde complète
Jeudi	Sauvegarde complète
Vendredi	Sauvegarde complète

La 3ème stratégie consiste à effectuer une sauvegarde complète tous les jours de la semaine. Cette stratégie est la plus coûteuse en termes de temps et de ressources mais reste la plus flexible et la plus efficace. Pour restaurer les données, il suffit de prendre la sauvegarde de la veille sans avoir besoin des autres sauvegardes.

1.4 Exemples de logiciels de sauvegarde

Les utilisateurs peuvent toujours utiliser des solutions de sauvegarde et gérer seuls leurs propres sauvegardes sans avoir besoin d'un abonnement chez un fournisseur. Voici quelques exemples de logiciels de sauvegarde de données :

- **Symantec Norton Ghost :** Norton Ghost est un logiciel conçu par Symantec pour cloner un disque dur. Son intérêt est qu'il permet d'éviter de réinstaller chaque logiciel à part. Ghoster (cloner) un PC est une tâche rapide qui ne nécessite pas beaucoup de compétences. L'image obtenue peut être utilisée pour dupliquer une machine sur d'autres machines et permet de restaurer l'état du disque au moment de la création de l'image.

- **Cobian Backup :** Cobian Backup est un logiciel de sauvegarde avec une interface simple. Il effectue des sauvegardes dans un répertoire local ou à distance par l'intermédiaire d'un client FTP inclus. Les options relatives à chaque sauvegarde sont nombreuses et claires (compression, sauvegarde incrémentale, récursivité dans les répertoires, exclusion par masque, lancement de programmes avant et après le backup…). Les mises à jour du logiciel sont faites automatiquement par internet.

- **Handy Backup :** C'est une solution de sauvegarde facile à utiliser. Il fonctionne en mode service Windows avec une variété de supports de mémoire ainsi qu'une sauvegarde FTP à distance. Ce logiciel permet d'effectuer des sauvegardes planifiées. Il existe dans plusieurs versions (Standard, Professional, Full et Server).

Dans la partie suivante nous allons présenter des architectures réseau client/serveur et P2P en expliquant chacune et en montrant les différences entre ces deux architectures.

1.5 Les architectures Client/serveur et P2P

1.5.1 Introduction

Dans un réseau, nous pouvons distinguer deux architectures qui sont l'architecture client/serveur (c/s) et l'architecture Pair-à-Pair ou Peer-to-Peer (P2P). Dans cette partie nous allons présenter ces deux architectures en montrant leurs usages et leurs caractéristiques.

1.5.2 L'architecture client/serveur

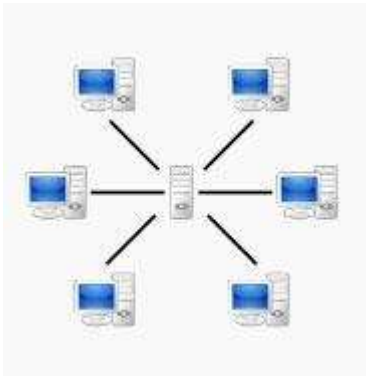


Figure 1 - Architecture Client/Serveur

L'architecture client/serveur (c/s) décrit la relation entre deux programmes d'ordinateurs. Dans cette architecture, le programme client fait une demande de service à l'autre programme qui est celui du serveur. Les fonctions standard du réseau telles que l'échange de mail, l'accès au web et aux bases de données, sont basées sur le modèle client-serveur. Par exemple, un navigateur web est un programme client qui se trouve sur le PC de l'utilisateur et qui permet l'accès à n'importe quel serveur de web dans le monde. Le navigateur transmet votre demande à un serveur web qui, à son tour, transmet la réponse à votre navigateur web qui affiche les informations pour vous. La Figure 1 montre un exemple de l'architecture c/s dans laquelle tous les ordinateurs sont liés à un serveur central. Ces ordinateurs n'ont aucune connaissance du réseau et chacun d'eux peut communiquer uniquement avec le serveur central sans même savoir que les autres ordinateurs existent.

Le modèle c/s est devenu l'une des idées centrales de l'informatique du réseau. La plupart des applications sont aujourd'hui développées en utilisant ce modèle. Il est même utilisé dans les principaux protocoles d'applications internet. Ces jours-ci, les clients sont le plus souvent des navigateurs, même si cela n'a pas toujours été le cas. Les serveurs sont plutôt des serveurs web, serveurs de base de données, serveurs de messagerie, … Les applications de type client sont par exemple les navigateurs web, les clients de messagerie, de chat en ligne, … Et les applications de type serveur comprennent les serveurs web, ftp, serveurs d'applications, serveurs de base de données, serveurs de noms, serveurs de messagerie, serveurs de fichiers, d'impression, …. La plupart des services web sont aussi du type serveurs.

Dans cette architecture nous avons souvent plusieurs clients connectés au même serveur, appelé serveur central. Dans les applications de backup, on peut avoir plusieurs clients qui sauvegardent en même temps. Dans un tel cas, une grande quantité de données est envoyée vers le serveur qui doit être capable de les gérer convenablement. Un serveur est souvent beaucoup plus puissant qu'un Pc normal (client). Il peut avoir plusieurs processeurs, beaucoup de mémoire et un grand espace de stockage. Cependant, cette architecture possède des avantages aussi bien que des inconvénients.

1.5.2.1 Avantages de l'architecture c/s

- L'architecture client serveur permet une maintenance facile et rapide vu que tous les clients sont connectés à un serveur central. La maintenance peut être effectuée tout simplement sur le serveur lui-même. Il est même possible dans certains cas de remplacer, mettre à jour ou réparer un serveur sans que les clients connectés s'en aperçoivent ou qu'ils soient affectés par ce changement.
- Toutes les données sont stockées sur le serveur, ce qui permet d'avoir un niveau de sécurité plus élevé car les contrôles d'accès et de sécurité sont configurés sur une seule machine et puisque seuls les clients qui ont le droit d'accès aux données pourront le faire.
- L'administration et la mise à jour des données centralisées est beaucoup plus facile à gérer que les données réparties.
- Les technologies supportant l'architecture c/s sont plus matures que les autres.

1.5.2.2 Inconvénients de l'architecture c/s

- Si beaucoup de clients se connectent en même temps au serveur, ce dernier risque de ne pas supporter la charge. Cela peut causer une congestion du trafic sur le réseau.
- Si jamais le serveur tombe en panne ou n'est pas disponible pour une certaine raison, aucun client ne peut accéder aux données.
- Si jamais une catastrophe (attaque terroriste, tremblement de terre, tempête, …) met le serveur hors service, aucun client ne peut accéder à ses données.
- Cette architecture exige des frais supplémentaires à cause de sa configuration avancée et du matériel dédié qui coûte plus cher que les machines standard. À cela, il faut

ajouter le coût de l'administration et de la maintenance qui sont indispensables au bon fonctionnement de ce genre d'architecture.

1.5.3 L'architecture Pair-à-Pair

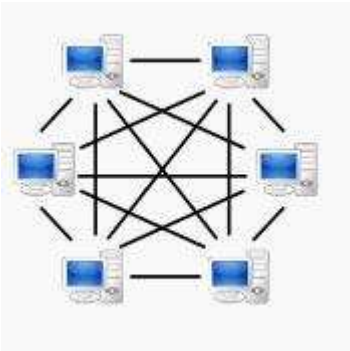


Figure 2 - Architecture Pair-à-Pair

Un autre type d'architecture de réseau est connu sous le nom de Pair-à-Pair (P2P). Un réseau P2P est un réseau décentralisé, distribué, sans organisation hiérarchique. Chaque nœud participant, joue à la fois le rôle de serveur et de client et a des responsabilités et un statut équivalent. L'organisation dans un tel réseau repose sur l'ensemble des pairs, il n'y a pas d'entité chargée d'administrer ce réseau. Dans la Figure 2, nous avons un ensemble d'ordinateurs ou nœuds qui sont connectés entre eux. Nous pouvons remarquer qu'il n'existe pas de serveur central unique. Chaque nœud ou pair peut jouer le rôle de serveur central puisque tous les autres nœuds du réseau peuvent se connecter à lui. Chaque nœud a une connaissance du réseau et peut établir une connexion avec n'importe quel autre nœud du réseau. L'utilisation d'un système P2P nécessite l'installation d'un logiciel particulier pour chaque nœud souhaitant rejoindre le réseau. Le P2P est devenu populaire à partir du logiciel Napster [**11**] créé par Shown Fanning. Napster est une application P2P de partage et d'échange de fichiers musicaux mp3. Le P2P offre un accès simple, rapide et quasiment gratuit pour télécharger et diffuser des données, puisqu'il ne requiert aucun moyen technique lourd et coûteux. La bande passante de votre connexion ainsi que celle des autres utilisateurs influeront fortement sur le réseau P2P. En plus du partage de fichiers, le P2P permet de communiquer (téléphonie VoIP, messagerie instantanée, etc.) avec les utilisateurs du réseau, il permet aussi de diffuser la télévision par Internet.

Les systèmes P2P se répartissent en deux grandes catégories qui sont l'architecture centralisée et l'architecture décentralisée.

1.5.3.1 Architecture centralisée

Dans l'architecture centralisée, un serveur de noms centralise l'entrée en communication et la coordination des pairs. Le reste des échanges est effectué ensuite par connexion directe entre les nœuds du réseau. Globalement, la majorité des échanges se fait par connexion directe. Dans les applications de partage de fichiers multimédia basées sur l'architecture P2P centralisée, le serveur de noms se connecte à une base qui contient l'index des fichiers multimédia. A la demande d'un nœud, le serveur lui fournit l'index ou l'adresse du fichier à télécharger. Le transfert du fichier va ensuite se faire par connexion directe avec une machine possédant le fichier. Les informations peuvent aussi transiter d'un utilisateur à l'autre, mais le serveur reste indispensable et si jamais ce dernier tombe en panne, le réseau s'effondre.

1.5.3.2 Architecture décentralisée

Une architecture P2P décentralisée ne comporte pas de serveur de noms. Les nœuds doivent donc assurer la connexion de nouveaux nœuds. Cela peut se faire moyennant différents mécanismes :

- Connaître au moins un nœud toujours présent.
- Broadcasting local ou multicast sur une adresse déterminée.
- Recevoir l'invitation d'un nœud déjà connecté.
- Autres…

Un nœud ne se connecte pas à un serveur unique mais à plusieurs. Le système dans ce cas est plus robuste et permet de résister à des attaques car, si jamais un ou plusieurs serveurs tombent en panne, cela n'empêche pas le fonctionnement du système. En contrepartie, le trafic est plus important et la recherche de fichier est plus difficile et prend beaucoup plus de temps, chaque requête étant adressée à chaque pair connecté, et chacun d'entre eux faisant de même. Pour pouvoir effectuer une recherche, il faut que tous les serveurs aient des informations sur l'emplacement des données, ce qui exige la réplication de ces informations sur tous les serveurs dans le système. Dans les systèmes décentralisés structurés, une organisation de connexion est maintenue entre les nœuds. Elle est basée essentiellement sur les tables de hachage distribuées (DHT). Le nombre des messages nécessaires pour effectuer une recherche est beaucoup plus petit que le logarithmique du nombre N des utilisateurs du réseau. Cette partie est présentée avec un peu plus de détails dans le paragraphe suivant.

1.5.3.3 Tables de hachage

Pour améliorer la complexité de la recherche, les concepteurs de systèmes P2P se sont tournés vers des structures de données connues. En effet, améliorer les algorithmes de recherche nécessite d'organiser le réseau, donc de le structurer. Une table de hachage distribuée est une technologie d'indexation qui permet l'identification et l'obtention d'une information dans un système réparti. Elle permet théoriquement la multiplication de fichiers. Cette technique permet d'associer un identifiant unique (hash) à chaque pair, fichier ou mot clef. Cet identifiant est calculé en fonction du contenu et non à partir du nom. Cela permet d'éviter le téléchargement de plusieurs fichiers offrant un contenu identique alors qu'ils sont différents en apparence puisqu'ils portent des noms différents. La table de hachage fournit un service de recherche aux utilisateurs. Dans cette table, on stocke des couples (clé, valeur), ce qui permet à chaque utilisateur de récupérer la valeur associée à une clé donnée. Les couples (clé, valeur) sont répartis sur tous les nœuds du système de telle manière que la déconnexion d'un pair ne cause pas une perte importante pour l'ensemble du réseau. Une DHT permet de gérer un important nombre d'utilisateurs, ainsi que le départ d'anciens nœuds et l'arrivée de nouveaux nœuds. Les opérations de base d'une DHT sont : *store (clé, valeur)* pour stocker une clé et sa valeur associée et *lookup (clé)* permettant de récupérer la valeur associée à la clé.

La technologie de la DHT est apparue avec la décentralisation des réseaux P2P car cette structure ne nécessite pas de serveur central. Chaque fois qu'un utilisateur rejoint le système, il n'a qu'à se connecter à n'importe quel nœud du réseau pour en faire partie. Les nœuds d'un réseau pair-à-pair basé sur une DHT n'ont pas une connaissance globale du réseau. Chaque nœud peut se rapprocher de la donnée cherchée jusqu'à la trouver si elle existe.

Il existe d'autres architectures décentralisées qui n'utilisent pas des mécanismes d'indexation (clé, valeur). Dans ces architectures, une requête est envoyée à toutes les machines du réseau, ce qui génère un nombre important de messages entre les utilisateurs ainsi que certains problèmes de réponses à certaines requêtes liés à la qualité et au temps de réponse. L'utilisation de la DHT limite l'envoi de messages concernant la recherche des fichiers car il n'est pas nécessaire d'envoyer un message à tous les nœuds mais de transmettre simplement la requête de pair en pair jusqu'à atteindre le pair qui a l'information cherchée. Certains algorithmes de recherche dans les DHT peuvent atteindre au pire une complexité en $O(log(n))$ pour un réseau de n nœuds « $O(log(n)) << log(n)$ ». Cette complexité est assez faible ce qui assure théoriquement un temps de réponse rapide à une requête [**12**].

Une notion de distance virtuelle est introduite dans la plupart des DHT. Elle a la forme d'une fonction calculant la distance entre deux nœuds. Chaque nœud possède une clé qui correspond à son propre identifiant. Un nœud *x* possède toutes les clés pour lesquelles *x* est la clé la plus proche (en distance virtuelle calculée par l'algorithme). Le but est de mettre en œuvre une topologie répondant à deux besoins essentiels au bon fonctionnement :

- Un degré assez faible pour une mise à jour rapide de la table de routage suite à l'arrivée ou au départ de nœuds dans le réseau.
- Un diamètre permettant de faire des recherches rapides et efficaces.

L'intérêt principal de cette technique est que les modifications (rajout ou suppression de nœuds) changent uniquement les clés des nœuds adjacents sans rien changer pour les autres nœuds.

Le principe de fonctionnement de la couche réseau est que, pour chaque nœud *n* on connaît, soit un nœud qui le connaît directement, soit un nœud plus proche de *n* qu'on ne l'est. Pour contacter le nœud *n*, il suffit d'envoyer le message au nœud que l'on connaît être le plus proche de *n,* qui va à son tour le transmettre au nœud le plus proche de *n*, et ainsi de suite.

Le résultat d'une fonction de hachage est codé sur *n* bits (n varie selon les protocoles). Le hachage de l'adresse IP représente l'identifiant de l'ordinateur sur le réseau. Le hachage du nom d'un fichier donne la clé. Les couples (clé, identifiant) sont ensuite stockés sur les nœuds du réseau de façon à avoir, pour chaque ressource du réseau, l'adresse de l'utilisateur qui la possède. Une redondance est introduite dans le système de façon à ce que le départ d'un nœud du système n'engendre pas la perte des métadonnées qu'il stocke.

Il existe plusieurs implémentations d'algorithmes de recherche. Dans certaines spécifications de protocole, chaque nœud conserve les informations sur d'autres nœuds afin d'accélérer les recherches. Certains protocoles de recherche permettent de diviser la distance par deux à chaque fois qu'une requête est envoyée à un autre utilisateur. Cela permet d'atteindre dans les meilleures implémentations une complexité logarithmique en $O(\log(n))$.

1.5.3.3.1 Exemple de DHT : Chord

Chord [**13**] est un projet P2P subventionné par le gouvernement des Etats-Unis d'Amérique. Ce projet est développé sous la licence MIT (Massachusetts Institute of Technology) qui ne possède pas de copyright et peut donc être modifiée pour s'adapter à des besoins particuliers dans le cadre d'exploitations à des fins non commerciales. La topologie

de la DHT Chord est une topologie en anneaux et utilise donc une notion de successeurs et de prédécesseurs dont elle garde trace dans sa table de routage à *m* entrées. Chaque utilisateur du réseau peut contacter directement *m* nœuds du réseau pour transmettre des requêtes. Chord est utilisé dans plusieurs applications comme CFS (Collaborative File System), qui est un système de fichiers distribué à l'échelle de l'Internet. Chord repose sur une topologie en anneaux ; un nœud Chord a la connaissance de son prédécesseur et de son suivant. Une fonction de hachage régulière génère une clé pour chaque nœud à partir de son adresse IP. Une fois la clé générée, chaque nœud est placé dans l'anneau de manière à ordonner les clés par ordre croissant. Chaque nœud Chord est responsable de l'intervalle de clés [clé(nœud actuel), clé(suivant)[.

Chord supporte bien le facteur d'échelle. La longueur moyenne d'un chemin pour acheminer une requête évolue en O(log(N)). La garantie de performances de Chord repose sur trois aspects qui sont : (1) l'utilisation d'une fonction de hachage au niveau d'un ou plusieurs nœuds, (2) l'ajout de nœuds virtuels dans chacun des nœuds réels qui permet d'améliorer l'équilibre du trafic et (3) le choix, pour un nœud donné, de ses fingers en fonction du temps de latence mesuré.

1.5.3.3.2 Exemple de DHT : Pastry

Pastry [**14**] reprend la structure circulaire de Chord à quelques détails près, le temps de recherche reste logarithmique. Les identifiants des pairs peuvent être générés aléatoirement (répartition uniforme). Le routage est basé sur les préfixes, mais il existe une plus grande liberté dans la table de routage, qui donne pour un certain nombre de préfixes une IP quelconque de pair ayant ce préfixe.

1.6 Conclusion

Comme nous avons vu dans les sections précédentes, la sauvegarde des données est très importante pour toute personne qui a des données à protéger. Nous avons envisagé plusieurs supports de stockage de données et nous signalons que, même si le meilleur support est le serveur distant (la sauvegarde en ligne), cette méthode reste souvent payante pour l'utilisateur. Dans la plupart des cas la sauvegarde en ligne exige des frais d'abonnement au service ce qui n'est pas envisageable pour tous ceux qui veulent protéger leurs données. Les fournisseurs de ce genre de services ne peuvent pas les rendre gratuits car pour assurer les sauvegardes des clients, les fournisseurs doivent installer plusieurs serveurs ce qui nécessite du matériel dédié (souvent cher) et de l'administration. Pour être capable de récupérer les frais

investis et faire des bénéfices, les fournisseurs sont dans l'obligation de rendre ces solutions payantes pour les utilisateurs. Nous avons parlé de deux architectures réseau qui sont l'architecture client serveur et l'architecture P2P. L'architecture client serveur qui a connue beaucoup de succès possède plusieurs limites (section 1.5.2.2) d'où le besoin de passer vers une architecture décentralisée comme le P2P qui permet de passer à l'échelle.

Dans le chapitre suivant, nous allons voir comment le P2P peut s'introduire dans la sauvegarde après le grand succès survenu dans les applications de partage de fichiers multimédias. Nous allons présenter aussi différents systèmes de sauvegarde P2P existants en expliquant le fonctionnement de chacun d'eux.

Chapitre 2 Les systèmes de sauvegarde P2P

2.1 Introduction

On observe aujourd'hui une admiration pour les systèmes de fichiers pair-à-pair. Leur but était à l'origine de pouvoir partager, à grande échelle, des fichiers multimédia. Dans ces systèmes, on passe à l'échelle et chaque nœud a le rôle d'un client et d'un serveur à la fois. Avec le temps, d'autres systèmes non dédiés exclusivement à la diffusion de fichiers multimédia ont été développés. Ils se présentent comme des systèmes de partage d'espaces de stockage et intègrent des mécanismes assurant la confidentialité et la pérennité des données. Récemment, certaines approches commencent à utiliser le P2P dans la sauvegarde. Le principe est d'exploiter les ressources non utilisées du pair pour stocker les données d'autres pairs dans le réseau.

Cette section propose une présentation rapide de quelques systèmes de partage de fichiers. Nous présentons ensuite l'un des systèmes de stockage les plus connus puis nous donnons une description de quelques systèmes pair-à-pair dédiés à la sauvegarde des données.

2.2 Les systèmes de partage de fichiers

2.2.1 Napster

Le premier système de partage de fichiers pair-à-pair est Napster. Au départ, Napster a été créé par un jeune étudiant américain de 18 ans qui souhaitait pouvoir échanger facilement de la musique avec ses amis. L'application Napster envoie au serveur l'adresse IP assignée à la machine de l'utilisateur ainsi que la liste des titres mp3 qu'il désire partager avec les autres utilisateurs. Ces informations sont alors stockées dans un répertoire sur un serveur centralisé. Lorsqu'un utilisateur cherche un titre, Napster lui fournit la liste des adresses IP des utilisateurs qui le possèdent et qui sont en ligne. Cela permet au demandeur de télécharger le titre cherché. Ce système n'offre aucun mécanisme garantissant une pérennité des données.

2.2.2 eMule

eMule [**15**] est basé sur le même principe que Napster. L'index des fichiers est centralisé et partagé par plusieurs serveurs. Pour accéder à la recherche et obtenir des résultats pertinents, il est nécessaire d'avoir à jour la liste des serveurs. Ce système n'offre aucun mécanisme garantissant une pérennité des données.

2.2.3 Gnutella

Contrairement à Napster et eMule, Gnutella [**16**] est le premier système pair-à-pair totalement décentralisé. Il permet la recherche et la récupération d'objets sans nécessiter de serveurs. Ce système a commencé comme un petit protocole P2P décentralisé. Sa décentralisation a donné aux utilisateurs « paix et confiance », à la différence de son prédécesseur Napster. Gnutella était un logiciel « open source » constamment développé par un groupe de développeurs de programmes, ce qui a permis une évolution constante du réseau Gnutella. Beaucoup de programmes P2P fonctionnent sur ce réseau. Comme dans les systèmes précédents, ce système ne possède pas un mécanisme garantissant la pérennité des données.

2.2.4 KaZaa

KaZaa [**17**] est basé sur une architecture pair-à-pair décentralisée. Les utilisateurs sont reliés directement entre eux. KaZaa obtient un succès et une popularité importante. La possibilité de reprendre un téléchargement interrompu et le fait de pouvoir télécharger un même fichier de plusieurs sources afin d'augmenter la vitesse permettent de pallier les inconvénients de la disponibilité temporelle. KaZaa a connu une expansion importante avec la sortie des forfaits internet illimités. Cela a permis aux utilisateurs de laisser leurs ordinateurs personnels connectés de façon à augmenter la disponibilité temporelle et quantitative des médias. Parallèlement, le débit des connexions internet par le câble a augmenté grâce à l'ADSL.

2.2.5 Freenet

Freenet [**18**] est un projet qui vise à permettre une liberté totale d'expression et d'information basée sur la sécurité de l'anonymat. Son originalité réside dans le fait que les fichiers sont stockés sur les pairs du système et non à la source. Ainsi, chaque ordinateur du réseau stocke une partie des informations disponibles sur le réseau. L'espace alloué par

chaque nœud sert donc à stocker des fragments de données chiffrées du réseau, données dont l'utilisateur ne connaît pas le contenu. Un ordinateur qui fait transiter une information peut en garder une copie dans son cache. Cette duplication permet d'améliorer la durée de vie et de réduire les temps d'accès. Dans Freenet, les fichiers peu référencés disparaîtront en premier. Le système assure une certaine pérennité aux données les plus demandées.

2.2.6 BitTorent

BitTorent [**19**] est un protocole récent d'échange de fichiers. Il permet de distribuer largement de grandes quantités de données en répartissant la charge. Les blocs téléchargés peuvent arriver dans un ordre quelconque depuis des sources multiples. Ce système a l'avantage de créer une sorte de cercle vertueux lors du partage des fichiers car dès que des parties du fichier souhaité sont téléchargés, elles sont disponibles pour les autres utilisateurs. Un système de récompense est mis en place. Il incite les utilisateurs à coopérer. Le principe consiste à redonner des morceaux du fichier pour pouvoir en acquérir de nouveaux plus rapidement.

Avant de rentrer dans les détails des systèmes de sauvegarde à large échelle, nous allons présenter brièvement un des systèmes de stockage pair-à-pair.

2.3 Les systèmes de stockage P2P

Suite au grand succès des systèmes de partage P2P, la nécessité de créer des infrastructures sécurisées passant correctement à l'échelle et le besoin de stocker de plus en plus de données ont motivé la création de systèmes de stockage pair-à-pair. Nous allons présenter brièvement l'un de ces systèmes qui est OceanStore.

OceanStore [**20**] est à la fois un système de sauvegarde et de stockage. Il permet de stocker les données et d'y accéder rapidement. Dans OceanStore, les données sont stockées sur des serveurs dédiés possédant une forte connectivité et une large bande passante. Pour assurer la disponibilité des données, même en cas de défaillance, un mécanisme de redondance et d'auto-surveillance des serveurs est mis en place. Un mécanisme de réplication existe aussi. Son rôle est d'assurer l'accès rapide aux données en plaçant les répliques sur des serveurs proches de l'utilisateur afin de lui garantir les meilleurs temps d'accès à ces données. Ce système repose sur une table de hachage distribuée (DHT) et utilise les réplications simples pour assurer de bonnes performances en lecture. L'évolution de ce système repose sur sa particularité de mises à jour des données et ses performances en tant que système de

fichiers. Ce système reste donc loin de l'optique que nous avons adoptée pour les systèmes de sauvegarde.

2.4 Les systèmes de sauvegarde P2P

Après l'élaboration de systèmes de stockage on commence à ressentir le besoin de systèmes de sauvegarde. Le but de ce genre de systèmes est de passer à l'échelle en remplaçant, pour stocker et sauvegarder les données des utilisateurs, l'usage de serveurs centraux par les espaces libres de disques des machines. Le principe n'est pas loin de celui des systèmes de partage de fichiers sauf que, dans un système de sauvegarde, les données sauvegardées ne sont pas accessibles à tout le monde. Les données sont cryptées et seul le propriétaire a le droit de les télécharger et de les décrypter.

De nombreux systèmes de sauvegarde P2P ont été élaborés. Ces systèmes se ressemblent entre eux sur certains points et sont plus ou moins différents sur certains aspects de leur architecture. Cependant, aucun de ces systèmes n'est exploité en conditions réelles. Quelques systèmes ont juste atteint le stage du prototype. Dans cette partie, nous allons aborder quelques systèmes de sauvegarde P2P.

2.4.1 Past

Past [**21**] est un système simple de sauvegarde P2P qui utilise, pour le routage et la recherche, une DHT et le système Pastry. Dans ce système, les répliques sont parfaites pour assurer la redondance et la fiabilité de la sauvegarde. Les fichiers stockés dans PAST possèdent un identifiant unique généré lors de l'insertion, ce qui empêche une insertion multiple d'un même fichier. Les fichiers peuvent être partagés, à la discrétion du propriétaire, en distribuant l'identifiant (éventuellement anonyme) ainsi qu'une clé de déchiffrement s'il le faut. PAST ne supporte pas l'effacement des fichiers. Dans PAST, les réplications sont classiques et ne gèrent pas le découpage des fichiers en blocs ; ainsi le coût de réplication n'est pas très important.

2.4.2 pStore

pStore [**22**] est un système de sauvegarde pair-à-pair qui offre à l'utilisateur la possibilité de sauvegarder et de restaurer ses données d'une manière sécurisée dans un réseau distribué composé de nœuds inconnus. Chaque utilisateur du système installe et utilise un client qui l'aide à générer des clés et à choisir les fichiers à sauvegarder. Pour insérer un

fichier, pStore calcule un identifiant spécifique pour éviter les conflits avec d'autres fichiers portant le même nom et appartenant à d'autres utilisateurs. pStore utilise l'algorithme de synchronisation Rsync [**23**] qui est un algorithme en open source. C'est l'un des algorithmes les plus utilisés pour la synchronisation des fichiers. Rsync consiste à découper un fichier en blocks et utilise une fonction de hachage pour calculer des empreintes de ces blocks. Ces empreintes sont envoyées vers une autre machine et sont utilisées par le récepteur pour chercher des blocks similaires. Rsync utilise deux fonctions de hachage, une rapide mais peu fiable et une très fiable (MD4) mais coûteuse en terme de calcul. Dans pStore, chaque fichier est découpé en blocs chiffrés de taille fixe. Les métadonnées (MD) sont assemblées indiquant ainsi la façon avec laquelle les blocs sont rassemblés pour reconstituer à nouveau la version désirée du fichier. Les MD et les données sont insérées dans le réseau P2P.

pStore se base sur Chord à cause du temps de recherche attractif de ce dernier. Si jamais le contenu d'un fichier change et qu'il est choisi pour être sauvegardé, seulement les modifications de ce fichier seront stockées. Pour faire une restauration, l'utilisateur spécifie le nom du fichier désiré et sa version, ainsi les MD sont récupérées pour indiquer l'endroit où les blocs sont stockés. Une fois que les blocs sont récupérés, leurs signatures électroniques sont examinées pour vérifier leur intégrité. pStore fait plusieurs réplications du fichier pour augmenter sa disponibilité et assurer sa sauvegarde. Si jamais l'un des pairs tombe en panne, les blocs de données peuvent être récupérés à partir d'autres pairs.

À la différence des systèmes de partage de fichiers P2P, un utilisateur pStore ne peut pas utiliser les fichiers des autres utilisateurs stockés sur son disque dur. Ces informations sont cryptées afin de préserver la confidentialité des données. Pour établir une équité entre les utilisateurs du système, chaque nœud désirant sauvegarder ses données sur d'autres nœuds doit accepter ou allouer la même taille de stockage aux autres nœuds. Les utilisateurs de pStore n'ont pas le droit d'effacer les données stockées sur leur disque dur. Sauf en cas d'accident, seuls les propriétaires des fichiers ont le droit de les effacer.

2.4.3 PeerStore

Contrairement à pStore qui store indifféremment les données et les MD, PeerStore [**24**] utilise pour les MD une table de Hachage distribuée (DHT). Cette méthode facilite la détection des répliques et évite toute migration de données en cas de départ ou d'arrivée d'un pair. Avant de commencer la sauvegarde, un fichier est découpé en blocs de tailles fixes disposant chacun d'un identifiant unique. Avant de chercher des partenaires pour stocker les

données, PeerStore commence par éliminer tous les blocs qui ont assez de répliques dans le réseau. Le nombre de répliques est déterminé en consultant la DHT. Cette étape permet de laisser de l'espace pour les nouvelles répliques. Chaque fois que des nouvelles répliques sont faites, les DHT sont mises à jour. La restauration des données est similaire à celle de pStore. Elle est réalisée en récupérant d'abord la liste des blocs, ce qui permet de connaître l'emplacement des blocs qu'il faut télécharger ainsi que la méthode nécessaire pour reconstituer le fichier demandé. PeerStore emploie une méthode pour réaliser l'équité entre les nœuds. Chaque nœud a le droit de contacter les autres nœuds pour vérifier l'existence de ses données. Les nœuds qui trichent sont punis par l'élimination de leurs blocs.

2.4.4 Pastiche

Pour diminuer la répétition des données dans le réseau, Pastiche [**25**] prend des images complètes des disques de stockage en se basant sur le fait que deux nœuds utilisant par exemple le même système d'exploitation ou les mêmes applications auront un grand nombre de fichiers en commun. Ces données communes peuvent être stockées sans cryptage. Pastiche utilise « Content-based indexing » comme algorithme d'indexation et de synchronisation. Les données sont découpées en blocs de tailles variables selon le contenu. Lorsqu'un nouveau fichier est créé, il est programmé pour le découpage. Pastiche sélectionne les voisins les plus proches avec au moins un voisin éloigné pour assurer une diversité géographique des données.

Pastiche utilise la DHT Pastry pour faciliter la découverte des nœuds. Chaque nœud est déterminé par un identifiant qui est un hachage de son nom de domaine (FQDN). Pendant sa recherche, un nœud de Pastiche essaie de trouver des voisins avec des données similaires. Cela réduit la taille des données à répliquer. Avant d'ajouter au réseau un nouveau morceau de données, Pastiche vérifie si ce morceau existe déjà. Si jamais ce morceau existe, son adresse est ajoutée à la liste et le compteur est incrémenté. Sinon, le fichier est crypté et envoyé dans le réseau.

Pour effectuer une restauration, un nœud Pastiche identifie les morceaux utiles à la restauration et les télécharge du voisin le plus proche. Pour établir une récupération complète de la machine, un nœud de pastiche garde une copie de ses MD de racine sur chaque membre de son ensemble. Lors du processus de restauration, un pair rejoint le réseau avec le même identifiant, qui était calculé à partir de son nom d'hôte. Ce pair obtient alors son nœud de

racine de l'un de ses voisins et le déchiffre, ce qui va lui permettre de récupérer ses propres données.

2.4.5 Samsara

Le système de sauvegarde P2P Samsara [**26**] est une extension de Pastiche. Le but de Samsara est de s'assurer que les nœuds ne consomment pas plus que ce qu'ils apportent. Ce but est plus facile à réaliser avec une relation symétrique entre les nœuds. Le principe dans Samsara consiste à punir le nœud qui participe mais ne répond pas aux demandes des autres et ne fournit pas les données des autres nœuds qu'il stocke chez lui.

Avant qu'un nœud ne rejoigne un groupe, il faut qu'il initialise son stockage en remplissant une partie en crédit. Chaque nœud a le droit de vérifier l'existence de ses données chez les nœuds qui les stockent. Pour répondre à une requête, le nœud n'est pas obligé d'envoyer toutes les informations sur les objets stockés mais juste la dernière clé hachée cumulée qui montre que toutes les données sont bien stockées. Lorsqu'un nœud ne répond pas aux requêtes, les autres nœuds se débarrassent d'un pourcentage de ses répliques avec une certaine probabilité. S'il répond plus tard, il est toujours capable de récupérer ses données à partir d'autres répliques et de refaire de nouvelles réplications. La distinction entre les nœuds qui trichent et ceux qui ont des problèmes de connexion est difficile mais très importante. Les nœuds ayant des problèmes de connexion ne doivent pas avoir toutes leurs données jetées comme c'est le cas dans Pastiche. Une solution proposée dans ce papier est de donner aux nœuds un certain temps pour répondre aux requêtes, ce qui leur permet de récupérer leurs données en cas de problèmes techniques ou de connexion, objectif qui représente le grand intérêt de la sauvegarde.

Dans Samsara, les nœuds offrent un stockage gratuit aux nouveaux membres comme prix de la création d'une nouvelle collaboration. La durée de ce stockage s'appelle « Grace period ». Un nœud désirant faire une attaque, pourra par exemple faire des répliques durant cette période sans stocker des données puis choisir de nouveaux nœuds une fois que cette période est terminée. C'est très facile à réaliser dans un système comme Pastiche où les nœuds choisissent leurs sites de répliques. Un nœud qui triche et ne répond jamais aux requêtes, va perdre toutes ses données au bout d'un certain temps. Cette méthode permet de distinguer les nœuds qui ont eu un problème technique de ceux qui trichent. Un nœud qui triche, ne répond jamais aux requêtes alors qu'un nœud qui a des problèmes techniques peut récupérer ses données puis répondre de nouveau aux requêtes des autres utilisateurs.

2.5 Conclusion

On remarque que la plupart des systèmes de sauvegarde P2P qui existent aujourd'hui ont des caractéristiques en commun. Dans la plupart de ces systèmes, un fichier est crypté, découpé en blocs de tailles fixes qui sont synchronisés à travers le réseau sous forme de plusieurs répliques pour assurer la redondance des données. Le découpage en blocs est intéressant car il évite le transfert de fichiers volumineux à travers le réseau et permet un téléchargement simultané à partir de plusieurs sources. Les blocs de données sont envoyés à travers le réseau P2P vers plusieurs nœuds du réseau. Des répliques sont faites pour certains blocs. Ces répliques sont aussi envoyées vers des nœuds du réseau.

Les ordinateurs d'aujourd'hui sont puissants, performants et connectés presque tout le temps au réseau internet. À l'opposé de leurs antécédents, ces ordinateurs sont équipés d'un grand espace de stockage qui dépasse souvent les besoins de l'utilisateur. Cela ne veut pas dire qu'on peut envoyer des blocs de données à n'importe quel nœud une fois que ce dernier est connecté au réseau. Dans les systèmes cités ci-dessus, on parle de l'envoi de répliques vers des nœuds du réseau. Le problème est que si on envoie des requêtes vers des nœuds incapables de répondre à notre demande (stockage ou récupération de données), ces requêtes seront refusées. Un refus de service de la part d'un nœud peut être argumenté par le besoin de ce dernier de sa bande passante complète ou par les ressources locales de la machine. Le fait d'envoyer des requêtes au hasard peut souvent générer un taux plus élevé de refus que d'acceptation. Ceci peut arriver tout simplement à cause d'un manque d'organisation entre les pairs du réseau.

Les systèmes de sauvegarde P2P dépendent de la consommation individuelle de chaque nœud par rapport à sa cotisation. Des études [**27**] sur Gnutella et Napster confirment que beaucoup d'utilisateurs consomment sans contribuer. Plusieurs mécanismes ont été proposés pour assurer l'équité de stockage. Ces mécanismes utilisent un outil tiers et exigent une certaine notion de frais centralisés et administratifs que les systèmes P2P sont censés éviter. La solution pourra alors être dans un système symétrique. Samsara est un système de sauvegarde P2P dont le but est de s'assurer que les nœuds ne consomment pas plus que ce qu'ils apportent. Le principe dans Samsara consiste à punir le nœud qui participe mais ne répond pas aux demandes des autres et ne fournit pas les données des autres nœuds qu'il stocke chez lui. Cette idée semble intéressante au premier abord mais reste difficile à appliquer. Si jamais un nœud a des problèmes avec sa machine (un disque dur qui tombe en

panne, une perte de données, …), ce nœud sera puni sans qu'il soit malveillant. La méthode de punition n'est pas logique dans certains cas. L'idéal sera de calculer la perte totale du système en groupant tous les problèmes liés à la perte des données. Par exemple le taux de mouvement (connexion, déconnexion) pour un réseau P2P peut être évalué et calculé. Si on oublie la méthode de punition et qu'on se concentre sur le but de garantir à chaque moment la persistance des données dans le système sans prendre en compte les causes de perte, cela offrira une meilleure qualité de service.

Chapitre 3 Persistance des données

3.1 Introduction

La persistance des données est un facteur important dans les systèmes P2P. Vu que les réseaux P2P sont dynamiques et que chaque nœud du réseau a une durée de vie inconnue, il est nécessaire d'être fortement tolérant aux pannes dans ce genre de systèmes. Le fait de stocker une donnée sur un seul pair du réseau est risqué puisque ce dernier peut disparaître à tout moment. Cependant la pérennité des données est un enjeu important. Pour assurer cette pérennité, il faudra créer plusieurs répliques des données critiques et les stocker sur plusieurs pairs du réseau de façon à pouvoir récupérer une copie de l'un des pairs si jamais certains pairs tombent en panne. Dans notre contexte, nous traitons la persistance des données et non pas la disponibilité des données à tout moment. Le but est de garder les données dans le système et d'être sûr de ne pas les perdre.

Dans [**28**], les auteurs s'intéressent à fournir une garantie probabiliste de maintenir dans le système un ensemble de nœuds (appelé cœur) en tenant compte du taux de mouvement du réseau. Cela consiste à définir un sous ensemble de nœuds sur lesquels les données critiques seront répliquées. Selon le taux de mouvement du réseau, il faut calculer le nombre de répliques à faire (en moyenne d'une réplique par nœud) de manière à pouvoir récupérer après un certain temps au moins une copie ou une réplique. Cela se traduit en une relation entre la taille du réseau, la taille du cœur et le taux de mouvement du réseau (le pourcentage des nœuds qui rejoignent ou quittent le réseau en une unité de temps). Les auteurs ne parlent pas de découpage en blocs. Les fichiers sont répliqués à travers le réseau avec leur taille entière, ce qui peut causer des problèmes dans le cas de fichiers volumineux. D'autre part, le découpage en blocs permet à chaque bloc d'être identifié et téléchargé séparément de telle sorte que le téléchargement simultané de plusieurs blocs appartenant au même objet soit réalisable.

Les auteurs supposent que le nombre des nœuds qui quittent le système en une unité de temps est égal au nombre des nœuds qui rejoignent le système. Cela veut dire que le nombre de nœuds dans le système reste toujours constant et indépendant des évolutions du système. Les auteurs supposent aussi que les nœuds qui quittent le système sont remplacés par des nouveaux n'ayant aucune connaissance du système. Chaque nœud qui joint le système est

donc toujours un nouveau nœud qui n'a aucune mémoire du système et ne possède aucune donnée.

Le cas simulé est très loin de la réalité. Si on suppose que chaque nœud, en quittant le réseau (perte de connexion, problème technique, crash, …) perd toute connaissance du réseau et perd aussi toutes les données, alors dans le cas ou le nœud sortant possède une donnée critique, on doit créer une nouvelle réplique pour remplacer cette dernière. Dans les systèmes réels, un nœud peut se déconnecter dans plusieurs conditions mais cela ne veut pas dire qu'il va perdre ses données. Il pourra se connecter plus tard et rejoindre le système. Si le nœud rejoint le système après un certain temps, cela évitera de faire des répliques supplémentaires qui pourront être coûteuses. Finalement, le modèle proposé n'utilise pas un système de redondance comme ceux que nous allons voir dans la section suivante.

3.2 Redondance des données

La méthode classique pour assurer une certaine pérennité est de faire appel à la redondance des données. La redondance des données est une propriété de certains tableaux de disque qui fournit la tolérance aux pannes, de sorte que tout ou une partie des données peut être récupérée en cas de panne.

La technique classique et la plus simple de redondance pour fournir une haute disponibilité des données est la simple réplication [**29**] sur un certain nombre de pairs dans le système. Son régime est facile à comprendre. Par exemple, pour faire face à n pannes dans le système, il sera nécessaire de dupliquer n fois les données (n étant le facteur de réplication). Cela veut dire que même si n pairs tombent en panne, la récupération des données reste possible. Ceci n'est pas le cas si n +1 pairs tombent en panne. Cependant cette technique consomme beaucoup en espace de stockage.

D'autres techniques de redondance basées sur les codes correcteurs [**30**] ont été développées. Ces techniques nommées « mécanisme de redondance avec fragmentation » minimisent l'espace de stockage. Leur principe est de fragmenter les blocs de données en un certains nombre n de fragments puis de créer r fragments de redondance de manière à pouvoir récupérer les données à partir de n'importe quel n fragments parmi les $(n + r)$.

Nous présentons ici plusieurs techniques de redondance de données dont celles utilisées dans les systèmes RAID ainsi que différents types de codes correcteurs dont Reed Solomon et Tornado.

3.2.1 Redondance dans les systèmes RAID

La technologie RAID [**31**] (acronyme de Redundant Array of Inexpensive Disks à l'époque), ce qui signifie « matrice redondante de disques bon marché » et qui est devenu aujourd'hui l'acronyme de « Redundant Array of Independent Disks » ce qui signifie « redondance de disques indépendants », permet de constituer une unité de stockage en assemblant plusieurs disques durs. L'unité créée a une grande tolérance aux pannes et donc une haute disponibilité. Ce système réduit les frais de stockage, mais n'assure pas la robustesse nécessaire pour résister au taux élevé d'échecs attendus dans plusieurs environnements. Les disques assemblés selon la technologie RAID peuvent être utilisés de différentes façons, appelées Niveaux RAID. Les niveaux RAID les plus utilisés sont :

3.2.1.1 RAID niveau 0

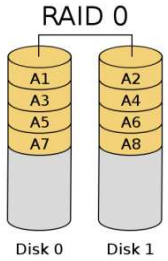


Figure 3 - Système RAID 0

Un système RAID 0 divise les données de manière uniforme sur deux disques ou plus sans intégrer la tolérance aux pannes ni les informations de parité pour la redondance. RAID 0 est habituellement utilisé pour augmenter les performances mais peut également être utilisé comme un moyen de créer un petit nombre de grands disques virtuels à partir d'un grand nombre de petits disques physiques. Dans la Figure 3 on remarque que les données sont réparties sur deux disques sans réplication. Aucune donnée de l'un des disques n'existe sur l'autre. Ce type de RAID est parfait pour des applications requérant un traitement rapide d'une grande quantité de données. Cette architecture n'assure en rien la sécurité des données. Si l'un des disques tombe en panne, la totalité des données du RAID est perdue.

3.2.1.2 RAID niveau 1

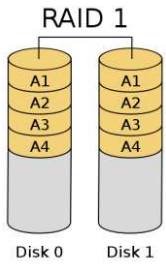


Figure 4 - Système RAID 1

Un RAID 1 crée une copie exacte (ou miroir) d'un ensemble de données sur deux disques ou plus. Ceci est utile lorsque la performance de lecture ou la fiabilité sont plus importantes que la capacité de stockage de données. Dans la Figure 4 on remarque que les données du disque 0 sont dupliquées sur le disque 1 ce qui permet une grande tolérance aux pannes. Cette tolérance aux pannes est la plus grande possible, il suffit qu'un seul disque soit en fonctionnement pour accéder à l'intégralité des informations stockées.

3.2.1.3 RAID niveau 3

Le niveau 3 propose de stocker les données sous forme d'octets sur chaque disque et de dédier un des disques au stockage d'un bit de parité. De cette manière, si l'un des disques tombe en panne, il serait possible de reconstituer l'information à partir des autres. Si deux disques tombent en pannes simultanément, il serait impossible de remédier à la perte de données.

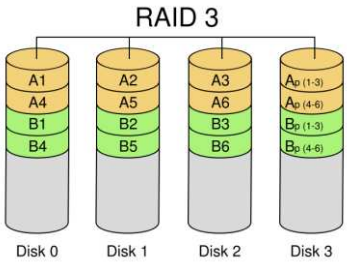


Figure 5 - Système RAID 3

3.2.1.4 RAID niveau 4

Le RAID4 est sensiblement semblable au RAID3 sauf que le niveau 4 travaille par blocs et non pas par octets. Le RAID4 offre des performances nettement supérieures au niveau 3 et ne nécessite pas autant de synchronisme entre les disques.

3.2.1.5 RAID niveau 5

Le niveau 5 est similaire au niveau 4 par le calcul de parité. Dans RAID 5, la parité est calculée secteur complet et non pas par bloc. La différence avec le niveau 4 est que cette parité est répartie sur l'ensemble des disques.

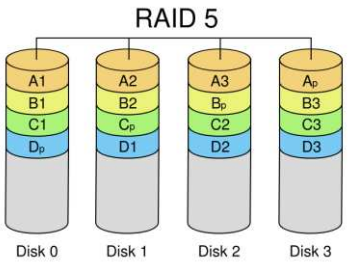


Figure 6 – Système RAID 5

Cela permet à RAID 5 d'améliorer énormément l'accès aux données car l'accès aux bits de parité est réparti sur les différents disques. Les performances obtenues dans RAID 5 sont très proches de celles obtenues dans RAID 0 tout en assurant une tolérance élevée aux pannes, raison pour laquelle c'est l'un des modes RAID les plus intéressants en termes de performance et de fiabilité.

3.2.2 Les codes correcteurs

En plus des systèmes de réplications cités ci-dessus, les DHTs du P2P ont proposés un autre système de redondance basé sur les codes correcteurs. Les codes correcteurs (erasure coding) permettent de fournir une redondance sans le coût d'une réplication stricte. Les codes correcteurs divisent un fichier en m fragments et recodent ces fragments en n fragments avec $n > m$. ($r = \frac{n}{m}$ est le facteur de redondance du codage). Un taux de code r multiplie les coûts de stockage par le facteur r. La propriété la plus intéressante des codes correcteurs est que le fichier d'origine peut être reconstruit à partir de n'importe quels m fragments. Lorsque le nombre de fragments disponibles dans le système est inférieur à m, la récupération du fichier devient impossible. Deux catégories de codes correcteurs ont été particulièrement étudiées et présentent des propriétés intéressantes Reed Solomon et Tornado.

3.2.2.1 Reed Solomon

Un code de type Reed-Solomon [**32**] découpe un bloc de données en n fragments de taille $\frac{T}{n}$ (T étant la taille du bloc). À partir de ces fragments, sont calculés r fragments de redondance. Chaque fragment de redondance a aussi la taille $\frac{T}{n}$. Les (n+r) fragments sont répartis sur (n+r) nœuds différents du réseau. Pour récupérer le bloc initial de données, il suffit de récupérer n fragments parmi les (n+r). Donc tant qu'il reste n fragments dans le système, le bloc initial de données peut être récupéré. Cela veut dire qu'un bloc de données peut supporter jusqu'à r défaillances.

Le codage s'effectue de la manière suivante : Le bloc de données est considéré comme un vecteur colonne $D=(d_i)_{i\in[1..n]}$ de dimension n où les d_i sont les fragments. On crée à partir de D un vecteur colonne $R=(r_j)_{j\in[1..(n+r)]}$ de dimension (n+r) dont les éléments seront les fragments incluant les fragments du bloc initial et les fragments de redondance.

Le vecteur R est déterminé par une matrice A de dimension $(n+r)\times n$ telle que n vecteurs lignes quelconques de A soient linéairement indépendants.

Soit $A=(a_{(j,i)})_{j\in[1..(n+r)],i\in[1..n]}$ une matrice $(n+r)\times n$ telle que n'importe quels n vecteurs lignes de A soient linéairement indépendants. Le produit $AD=R$ donne un vecteur colonne de dimension $(n+r)$. Le vecteur D peut être reconstruit à partir de n fragments quelconques de R. Soit R' le vecteur formé par ces fragments, $R'=(r_k)_{k\in[1..n]}$.

Soit A' la matrice carré $n\times n$ construite à partir des lignes de A correspondantes aux fragments de R'. $A', D \text{ et } R'$ sont liés par la relation : $A'D=R'$

A' est inversible, donc $A'^{-1}R'=D$. Avec la matrice inverse, le codage est juste une question de multiplication matrice vecteur comme le décodage.

Les codes Reed-Solomon sont intéressants malgré leur complexité de calcul. En général, les codes Reed-Solomon sont utilisés pour des tailles de bloc assez petites (taille ≤ 256) autrement les temps de codage et de décodage sont trop important [**33**]. Cette technique de redondance s'avère très pratique pour économiser de l'espace utile. Ces codes sont largement utilisés et éprouvés.

3.2.2.2 Tornado

Contrairement aux codes de type Reed-Solomon, il est préférable d'utiliser les codes Tornado [**34**] pour les blocs de grande taille car la complexité de décodage est bien moins importante. Ces codes ont été conçus pour la diffusion satellitaire. Les codes Tornado protègent un groupe de n fragments par r fragments de redondance. Le nombre de fragments, $n+x$ avec $x=\varepsilon\times n$ et $1>\varepsilon>0$, suffisant pour reconstruire le bloc initial de données dépend du codage Tornado utilisé. En général, la taille des blocs est prise de telle sorte que le nombre de fragments de redondance soit égal à la moitié du nombre initial des fragments mais ce n'est pas toujours le cas. La Figure 7 suivante illustre la génération des $h=4$ paquets de redondance à partir de $k=8$ paquets d'information. On note par le symbole $\oplus$ l'opérateur représentant le « ou exclusif ».

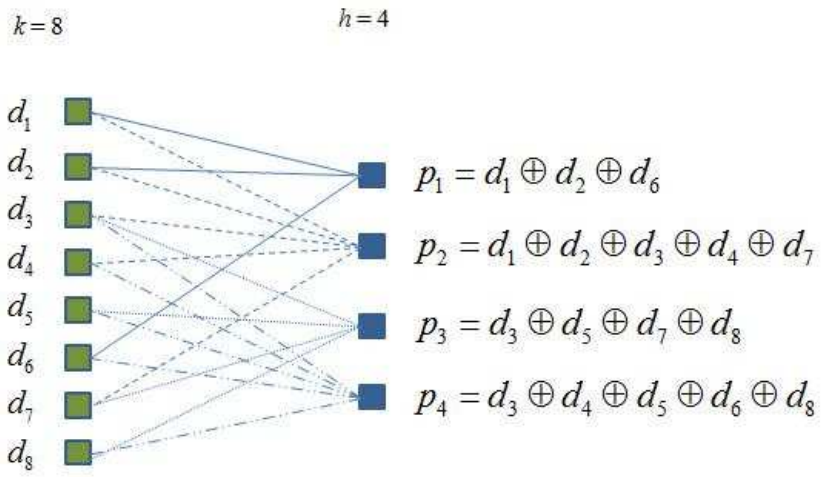


Figure 7 – Génération des paquets de redondance

Si jamais les paquets d_4 et p_2 sont perdus. Le paquet d_4 peut être réparé à partir des paquets d_3, d_5, d_6, d_8 et p_4 car $d_4 = d_3 \oplus d_5 \oplus d_6 \oplus d_8 \oplus p_4$. Une fois d_4 reconstruit, on en déduit immédiatement p_2. Le schéma précédant peut être interprété comme un graphe biparti dans lequel le premier ensemble de sommets est constitué par les paquets d'information alors que le deuxième ensemble de sommets contient les paquets de redondance. La construction de ce graphe est décrite dans [**35**].

On peut considérer que Reed Solomon est un cas particulier de Tornado. Le code correcteur d'erreurs Tornado encode et décode beaucoup plus vite les informations.

Temps Encodage (en s), blocs de 1Ko		
Taille	Reed Solomon	Tornado
250 Ko	4.6	0.06
500 Ko	19	0.12
1 Mo	93	0.26
2 Mo	442	0.53
4 Mo	1717	1.06
8 Mo	6994	2.13
16 Mo	30802	4.33

Temps Décodage (en s), blocs de 1 Ko		
Taille	Reed Solomon	Tornado
250 Ko	2.06	0.06
500 Ko	8.4	0.09
1 Mo	40.5	0.14
2 Mo	199	0.19
4 Mo	800	0.40
8 Mo	3166	0.87
16 Mo	13829	1.75

Figure 8 – Comparaison entre Reed Solomon et Tornado

Dans la Figure 8, on voit une comparaison issue de l'article [**35**]. Tornado est environ 100 fois plus rapide sur les petits blocs de données et environ 10 000 fois plus rapide sur les gros blocs. Finalement, Tornado est un code correcteur très efficace en termes de performances et de protection des données. Les codes de type Reed Solomon et Tornado sont appelés les codes correcteurs d'erreurs (erasure coding).

3.3 Efficacité des techniques de redondance

Plusieurs études théoriques ont été faites sur la redondance des données. Dans [**28**] les auteurs définissent la notion de persistance d'un groupe de nœuds (cœur) pour garantir l'existence dans le système de données critiques après un certain temps. Plusieurs facteurs ont été définis tels que la taille du système, la taille du cœur, le taux de mouvement du réseau (taux des entrés et des sorties des nœuds). Le modèle ne prend pas en compte le découpage des fichiers en blocs. Les fichiers sont répliqués avec leur taille complète ce qui peut causer des problèmes dans le cas de fichiers volumineux. Le découpage de fichier en blocs permet de les identifier indépendamment et de télécharger le même fichier simultanément à partir de plusieurs sources ce qui permet d'accélérer le téléchargement, d'autant plus que la réplication de chaque bloc reste possible et simple même en cas de faible connexion. Dans le même

article, les auteurs supposent que les nœuds qui quittent le système ne reviennent jamais. Ceci est loin de la réalité car avec cette supposition, à chaque fois qu'un nœud quitte le système il perd toute connaissance de ce dernier et il est remplacé par un nouveau nœud qui n'a aucune connaissance du système. Cela veut dire que si le nœud qui sort possède des données critiques (répliques), ces données seront perdues. Dans la réalité, un nœud possédant des données peut se déconnecter pour une certaine période et rejoindre le réseau plus tard. Un nœud peut quitter le système à cause d'un problème technique comme la déconnexion temporaire mais il peut quand même garder ses données. Si on considère qu'à chaque fois que le nœud est déconnecté, il perd ses données, cela veut dire qu'à chaque fois qu'un nœud possédant des donnés critiques disparaît du réseau, on va faire de nouvelles répliques pour remplacer les répliques perdues et maintenir la persistance du cœur. Le fait de créer de nouvelles répliques a un certain coût. Dans ce papier, les codes de redondance tels que Reed Solomon ou Tornado ne sont pas utilisés. Les auteurs définissent un modèle probabiliste en liant les paramètres définis. Le modèle permet de calculer un paramètre par rapport aux autres. Par exemple, il permet de calculer la taille du cœur à prendre selon la taille du réseau et le taux de mouvement pour qu'on ait au moins une réplique après un certain temps ce qui assure la probabilité de récupérer le fichier après un certain temps.

Des comparaisons dans [**36**] [**30**] ont montré que les codes correcteurs sont plus efficaces que la réplication classique. Des études quantitatives sont effectuées sur les deux systèmes. Les auteurs étudient le temps moyen pour que le système tombe en panne et qu'on ne puisse plus récupérer les données (Mean Time to Failure : MTTF). Selon les auteurs de l'article, l'usage des codes correcteurs augmente la persistance des données d'un ordre important par rapport à la réplication simple.

Alors que beaucoup de comparaisons ont affirmé que le codage est le vainqueur par rapport à la réplication classique en raison d'énormes économies de stockage pour les mêmes niveaux de disponibilité (ou à l'inverse, d'énormes gains de disponibilité pour les mêmes niveaux de stockage), les études dans [**37**] ont conclus que bien que les gains de codage existent, elles dépendent fortement des caractéristiques des nœuds. Les avantages de codage sont ainsi limités dans certains cas, ils peuvent facilement être compensés par certains inconvénients comme la complexité supplémentaire des codes.

Un nouveau système de redondance appelé hybride est proposé dans [**38**]. Il combine les codes correcteurs avec la réplication classique. Le modèle fonctionne avec le principe de codage sauf qu'une fois les blocs de données sont téléchargés et assemblés sur un certain

nœud pour construire le fichier original, ce dernier est gardé dans le système et présente une nouvelle source de téléchargement. Donc le fait d'avoir un fichier complet dans le système est le même principe que la réplication classique qui consiste à faire des copies du fichier sur plusieurs nœuds du réseau. Ce modèle augmente l'existence des fichiers dans le système. De plus on n'a pas besoin d'un coup supplémentaire pour la réplication. Le principe est juste de garder le fichier au lieu de l'effacer après l'avoir téléchargé. Ce modèle peut être efficace mais ne convient pas dans notre cas. Nous travaillons dans un contexte où chaque utilisateur veut sauvegarder ses données personnelles cryptées. Par mesure de sécurité chaque utilisateur doit être autorisé à télécharger ou voir uniquement ses propres données. Les données téléchargées par un utilisateur quelconque ne doivent pas être ajoutées au système.

Dans l'article [**39**], les auteurs utilisent les chaînes de Markov pour analyser le problème de persistance. Des formules sont dérivées pour trois systèmes différents qui sont la réplication simple, les codes correcteurs et le système hybride qui combine les deux précédents. Le modèle propose trois états pour chaque nœud du système : connecté, déconnecté et hors du système. Le taux de mouvement dans le système (taux des vas et vient) est utilisé pour définir la probabilité qu'un nœud quitte définitivement le système. Cela n'est pas très réaliste car la sortie définitive d'un nœud ne peut pas être définie que d'une manière spécifique. Quand un nœud se déconnecte on ne peut pas savoir s'il va se reconnecter ou s'il est tombé en panne. Le seul moyen de le savoir est que le nœud lui-même envoie une information pour signaler qu'il quitte définitivement le système ou s'il existe une règle prédéfinie qui considère qu'au bout d'un certain temps d'absence (déconnexion) un nœud est considéré hors du système. L'article n'aborde pas la question de déterminer le nombre minimum de répliques nécessaires pour que le système puisse survivre après un certain temps. Les résultats de ce papier montrent que le modèle hybride est le meilleur sauf dans le cas où le taux de mouvement devient très élevé et que la disponibilité des nœuds dans le système devient faible. Dans ces cas là, la réplication classique peut être parfois plus performante.

3.4 Conclusion

Dans ce chapitre, nous avons étudié la persistance des données. Nous avons présenté différentes techniques de redondance des données qui sont les systèmes RAID et les codes correcteurs Reed Solomon et Tornado en expliquant le fonctionnement de chaque système et les différences entre les différentes techniques de redondance. Nous avons conclu en montrant l'efficacité des techniques de redondance et en citant un certain nombre de travaux sur ce

domaine. Dans le chapitre qui suit, nous présentons notre approche de planification pour un système de sauvegarde P2P.

Chapitre 4 Planification pour un système de sauvegarde P2P

4.1 Introduction

Les ordinateurs d'aujourd'hui sont puissants, performants et branchés presque tout le temps au réseau internet. Contrairement à leurs prédécesseurs, ces ordinateurs sont équipés d'un grand espace de stockage qui dépasse souvent les besoins de l'utilisateur ; d'où l'idée d'exploiter ces espaces pour sauvegarder les données des autres utilisateurs. À la place d'un serveur central dédié à la sauvegarde des données, nous passons à l'échelle et nous utilisons les ressources des nœuds pour effectuer cette sauvegarde. Pour effectuer une sauvegarde, les blocs de données à sauvegarder sont envoyés vers d'autres nœuds du réseau. La question qui se pose est : Quels sont les nœuds vers lesquels on peut envoyer les blocs de données ? Parmi les nœuds connectés, plusieurs nœuds utilisent les ressources de leur machine (bande passante, processeur, mémoire, …). Si nous envoyons les blocs de données vers un nœud incapable de répondre à notre requête (sauvegarde ou restauration de données), cette dernière sera sûrement refusée. Un refus de coopération de la part d'un nœud peut être argumenté par le besoin de ce dernier de sa bande passante complète ou des ressources locales de la machine. Le fait d'envoyer des requêtes au hasard peut générer un taux plus élevé de refus que d'acceptation. Ceci peut arriver tout simplement à cause d'un manque d'organisation entre les nœuds du réseau.

Dans cette partie, nous proposons pour les systèmes de sauvegarde P2P, un système de planification basé sur la topologie du réseau P2P. Notre système organise la communication et l'envoi de requêtes entre les pairs. Dans la partie suivante, nous présentons l'architecture du système de planification ainsi que son fonctionnement.

4.2 Architecture globale du système de planification

Nous présentons dans cette partie l'architecture de notre système de planification qui est à la base de notre système de sauvegarde. Nous définissons un réseau dans lequel les pairs sont organisés en groupes (cluster). Chaque groupe constitue un ensemble de pairs qui collaborent entre eux pour fournir les services de sauvegarde. La collaboration des pairs d'un groupe permet à ce dernier d'avoir un fonctionnement similaire à celui d'un serveur puissant.

La figure 9 présente l'architecture de notre système de sauvegarde. Chaque groupe est constitué de plusieurs pairs qui peuvent communiquer et collaborer entre eux. Pour chaque cluster, un maître est choisi qui est souvent un pair ayant plus de ressources que les autres et une connectivité élevée. Chaque maître possède un identifiant unique avec lequel il est indexé dans la DHT. Cette indexation caractérise le réseau des maîtres. L'intérêt d'avoir un maître pour chaque groupe dans la DHT est de pouvoir indexer les informations des autres nœuds du groupe dans la DHT par l'intermédiaire de ce nœud maître. Donc le maître indexe les informations de son groupe dans la DHT et peut consulter les informations des autres groupes dans la DHT ce qui lui permet de répondre aux requêtes des nœuds de son groupe et de leur passer des informations concernant les autres groupes. Nous définissons deux types de connexions : une connexion (intra-groupe) qui est la connexion entre les pairs d'un même groupe et qui permet à ces pairs de collaborer entre eux. Le deuxième type de connexion est la connexion de service qui permet de fournir le service de sauvegarde.

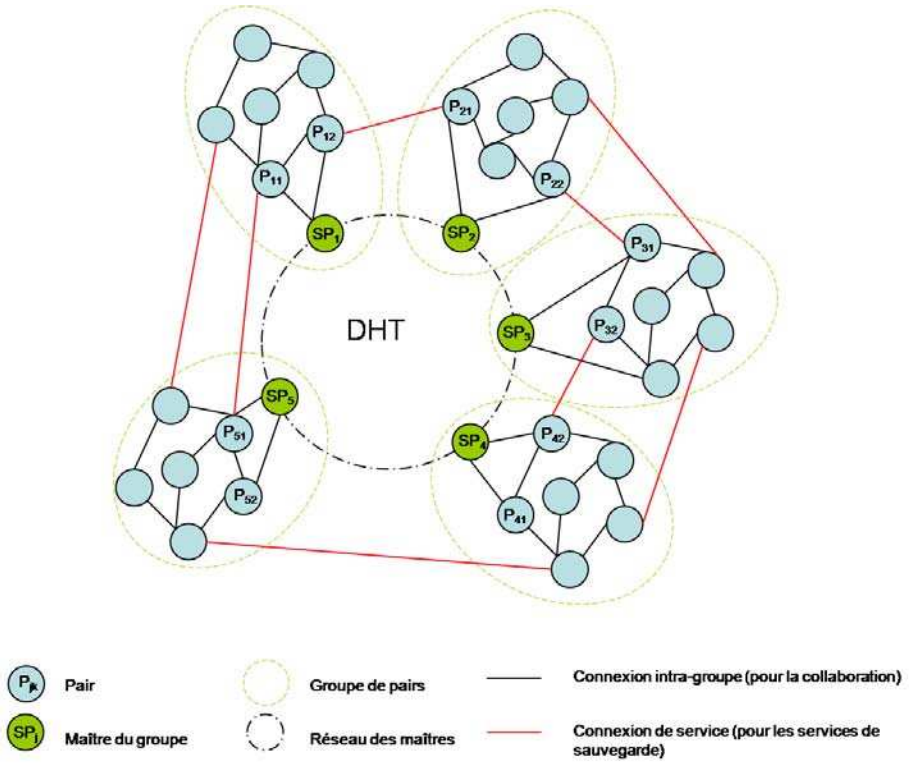


Figure 9 - Architecture de notre système de sauvegarde

4.2.1 Le modèle réseau

Nous considérons notre réseau P2P comme un graphe connecté $G=(N,C)$ où N est l'ensemble des nœuds représentant l'ensemble des pairs et C est l'ensemble des bords représentant les connexions entre les pairs. Nous considérons la partition $N=\{N_i\}_{1<i<k}$ de N, où N_i est l'ensemble des pairs d'un cluster. $Card(N_i)$ est la taille du cluster N_i. Elle est égale au nombre de pairs appartenant à ce cluster.

Pour organiser la collaboration entre les pairs de chaque groupe, l'un d'entre eux au moins doit être choisi comme maître du groupe ou cluster. Chaque groupe possède uniquement un seul maître. Les maîtres des groupes sont indexés dans une DHT par leur identifiant. Nous considérons la partition $L=\{CC_k,CB_l\}$ de C, où CC_k représente la connexion logique des pairs pour la collaboration (connexion intra-cluster) et CB_l représente la connexion pour le service de sauvegarde. Si le maître du groupe se déconnecte ou tombe en panne, un autre nœud est élu pour le remplacer.

4.2.2 Les Plannings

Nous avons défini deux classes de plannings qui sont les plannings de pairs et les plannings de groupes. Le planning de chaque pair est utilisé pour présenter les disponibilités ou les besoins de ce pair dans le système. Le planning d'un groupe est l'union des plannings des pairs appartenant à ce dernier.

Pour chaque pair, nous définissons deux plannings : un planning de service (Peer Service Schedule : PSS) et un planning pour les sauvegardes (Peer Backup Schedule : PBS). Les plannings peuvent être configurés pour une tache quotidienne, hebdomadaire ou mensuelle. Ces plannings sont remplis, pour chaque pair, au cours de la configuration initiale du logiciel de sauvegarde. Il est également possible de modifier plus tard un planning. Les plannings sont stockés sous forme d'une matrice (7×24) dans laquelle les lignes représentent les jours de la semaine et les colonnes représentent les créneaux horaires. Les 24 heures du jour sont divisées en 24 tranches de 1h chacune.

4.2.2.1 *Le planning de service d'un pair*

Le planning de service d'un pair (Peer Service Schedule : PSS) est un planning dans lequel un pair indique les heures durant lesquelles il sera connecté et libre et peut par conséquent présenter des services aux autres nœuds du réseau.

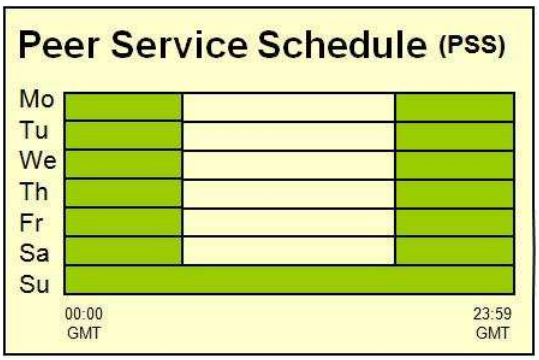


Figure 10 - Planning de service d'un pair

Par exemple si l'utilisateur de la machine sait à l'avance que le dimanche, il n'utilise pas sa machine ou que sa machine ne sera pas chargée, il peut laisser sa machine connectée pour que les autres utilisateurs du réseau puissent bénéficier des ressource de cette dernière. La Figure 10, montre un exemple de PSS pour un pair. Les zones en vert indiquent les périodes durant lesquelles le pair ou la machine est capable de présenter des services.

4.2.2.2 *Le planning de sauvegarde d'un pair*

Le planning de sauvegarde d'un pair (Peer Backup Schedule : PBS) est un planning dans lequel un pair indique les heures durant lesquelles il souhaite sauvegarder ses données sur les autres pairs du réseau.

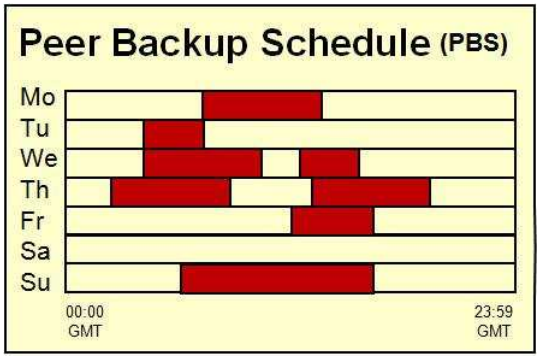


Figure 11 - Planning de sauvegarde d'un pair

Il suffit de remplir le planning de sauvegarde avec les horaires durant lesquels l'utilisateur désire sauvegarder ses données. La Figure 11 donne un exemple de PBS pour un pair. Les zones en rouge sont les périodes durant lesquelles le pair ou l'utilisateur souhaite sauvegarder ses données.

Pour chaque groupe dans le réseau, nous définissons deux types de plannings qui sont : Le planning de service d'un groupe et le planning de sauvegarde d'un groupe. Le planning d'un groupe est stocké par le maître du groupe et manipulé par ce dernier.

4.2.2.3 Le planning de service d'un groupe

Le planning de service d'un groupe (Group Service Schedule : GSS) est l'union des plannings de service de tous les pairs de ce groupe. Comme le réseau P2P est dynamique, chaque fois qu'un pair quitte ou rejoint un groupe, le planning de ce groupe change pour rester à jour.

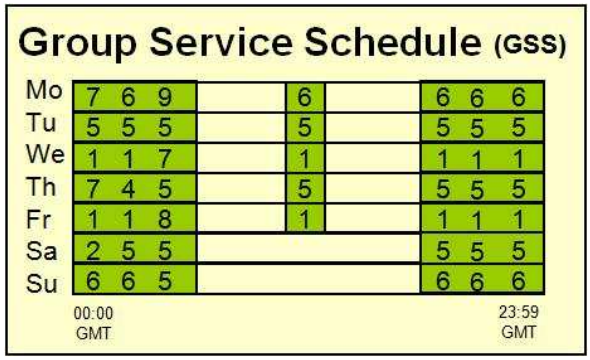


Figure 12 - Planning de service d'un groupe

Un seul planning de service est calculé par groupe. Ce planning est calculé à partir des plannings de tous les pairs du groupe. Pour chaque créneau horaire, la valeur dans le planning de service du groupe est égale au nombre de pairs du groupe qui présentent leurs services pendant ces horaires. La Figure 12 présente le planning de service d'un groupe. Le créneau du Lundi (minuit - 1h du matin) a la valeur 7. Cela veut dire que pour ce groupe, il y a sept pairs qui présentent des services pendant cette période.

4.2.2.4 *Le planning de sauvegarde d'un groupe*

Le planning de sauvegarde d'un groupe (Group Backup Schedule : GBS) est l'union des plannings de sauvegarde de tous les pairs de ce groupe. Pareillement au planning de service, l'entrée ou le départ d'un pair du group entraine le changement du planning de sauvegarde pour que ce dernier reste à jour.

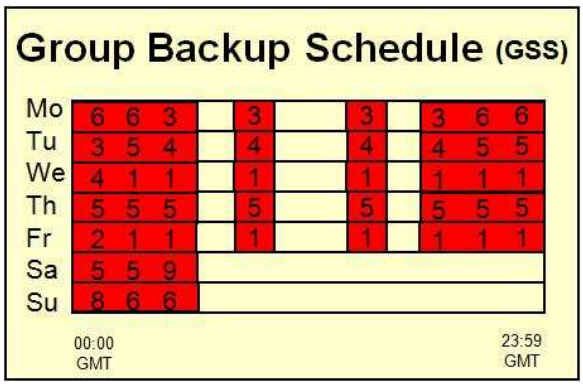


Figure 13 - Planning de sauvegarde d'un groupe

Un seul planning de sauvegarde est calculé par groupe. Ce planning est calculé à partir des plannings de tous les pairs du groupe. Pour chaque créneau horaire, la valeur dans le planning de sauvegarde du groupe est égale au nombre de pairs du groupe qui veulent sauvegarder leurs données pendant ces horaires. La Figure 13 présente le planning de sauvegarde d'un groupe. Le créneau du Lundi (minuit - 1h du matin) a la valeur 6. Cela veut dire que pour ce groupe, il y a six pairs qui souhaitent sauvegarder leurs données durant cette période.

4.2.3 Transformation des données

Chaque pair identifie les données qu'il veut sauvegarder. Chaque fichier est divisé en plusieurs blocs de taille fixe. Les blocs sont cryptés en utilisant l'algorithme SHA-512[5] qui est une fonction de hachage cryptographique qui dérive de SHA-1. L'intérêt de cet algorithme de hachage est de protéger les données sauvegardées. Un identifiant unique est attribué à chaque bloc de données, cependant chaque sauvegarde est représentée sous la forme d'une liste d'identifiants uniques. Ces identifiants sont utilisés dans le processus de restauration des données.

[5] http://fr.wikipedia.org/wiki/SHA-512

4.2.4 Indexation des groupes

Notre architecture de réseau est basée sur une DHT dans laquelle sont indexés les maîtres de tous les groupes. La DHT est utilisée pour faciliter et accélérer la recherche du groupe le plus convenable pour intégrer un nouveau pair ou pour effectuer une sauvegarde. À cet effet nous proposons des clefs qui permettent de reconnaître la disponibilité d'un pair pour présenter des services de sauvegarde ou pour sauvegarder ses propres données. Ces clés sont calculées à partir des plannings des groupes examinés dans la partie 4.4. Pour un planning de groupe, chaque période (tranche horaire) de chaque jour est convertie en un nombre selon les règles définies dans les tableaux de la Figure 14. La combinaison du jour et de la période va constituer la clé.

Day	Mo	Tu	We	Th	Fr	Sa	Su
Digits	000	001	010	011	100	101	111

Time	00:00 – 01:00	01:00 – 02:00	02:00 – 03:00	03:00 – 04:00	…
Digits	00000	00001	00010	00011	…

Figure 14 - Tableau de conversion des jours et des heures

Par exemple pour mardi entre 2h et 3h la clé est la combinaison de 001 (mardi) et de 00010 (2h-3h) ; ce qui donne la clé : 00100010.

Comme nous l'avons vu dans la partie précédente, chaque maître de groupe possède un planning global qui indique, pour chaque tranche horaire de la semaine, le nombre de pairs présentant un service. Dans chaque groupe le maître va transformer ce planning global de service en un ensemble de clés avec un numéro qui représente le nombre de pairs pour chaque créneau horaire. Par exemple si pour le mardi (3h-4h), nous avons 5 pairs du même groupe qui présentent des services, nous aurons la clé 00100011 cinq fois. Après leur création, les clés sont indexées par le maître du groupe dans la DHT. Le but de cette indexation est de rendre la recherche d'un groupe plus rapide et plus efficace.

4.3 Le protocole de sauvegarde

Le protocole de sauvegarde comporte deux étapes. La première étape consiste à trouver le groupe le plus convenable pour effectuer la sauvegarde et la deuxième étape est l'exécution de la sauvegarde par les pairs de ce groupe.

4.3.1 Recherche de groupe

Pour effectuer la sauvegarde, un pair P_{ik} doit trouver un ou plusieurs groupes dans lesquels il existe un ou plusieurs pairs capables d'effectuer cette sauvegarde. Parcourir tous les pairs du réseau et les interroger un à un n'est pas une solution efficace. Si P_{ik} ne possède pas de connexion de sauvegarde vers aucun groupe, il va contacter le maître SP_i de son groupe et va lui communiquer un ensemble de clés recueillies sur son planning de sauvegarde PBS. En utilisant ces clés, SP_i va pouvoir consulter la DHT et trouver dans chaque groupe les pairs qui sont prêts à sauvegarder les données de P_{ik}. La Figure 15(a) présente l'exemple d'un pair P_{jk} qui veut sauvegarder ses données. P_{jk} envoie une requête vers le maître SP_j de son groupe. SP_j cherche dans la DHT et trouve le groupe G_p dans lequel il existe un ou plusieurs pairs disponibles pour effectuer la sauvegarde de P_{jk}.

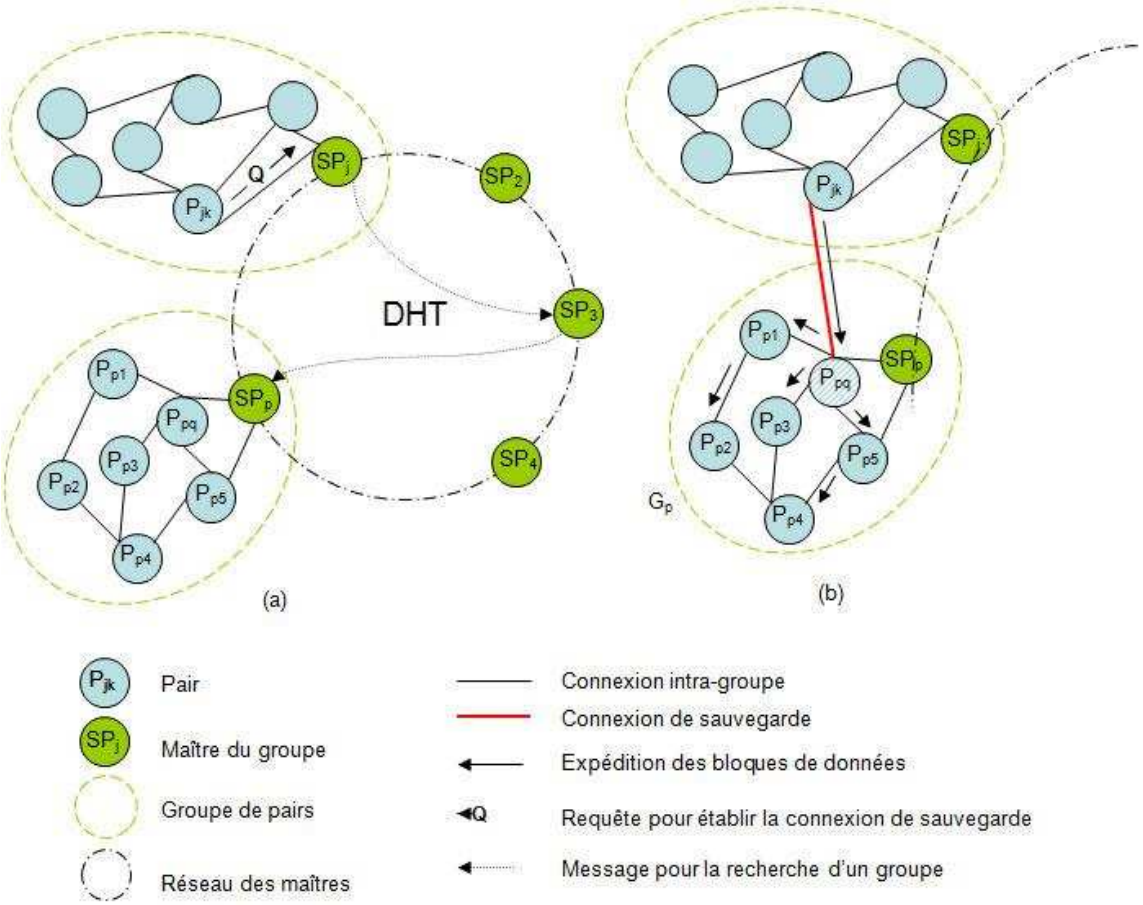


Figure 15 - Exécution de la sauvegarde

4.3.2 Exécution de la sauvegarde

Nous supposons que le groupe G_p est le groupe qui va effectuer la sauvegarde pour le pair P_{jk}. Un coordinateur est choisi parmi les pairs les moins chargés du groupe. Dans la Figure 15(b), P_{pq} est choisi comme coordinateur. Le coordinateur a pour rôle d'effectuer le transfert des données d'un nœud du groupe vers d'autres nœuds dans un autre groupe. Le fait de choisir l'un des nœuds les moins chargés comme coordinateur permet à ce dernier de répondre aux requêtes de son groupe plus facilement qu'un nœud plus chargé. Une connexion de sauvegarde est donc établie entre P_{jk} et P_{pq}. À travers cette connexion, P_{jk} envoie ses blocs de données pour que P_{pq} exécute le processus de sauvegarde en collaboration avec d'autres pairs de son groupe. Cette notion de groupe aide chaque pair à avoir des connaissances sur les autres pairs participants au même groupe. Ceci permet de définir une stratégie de sauvegarde collaborative qui aide à envoyer les blocs de données vers les pairs les plus disponibles et les moins chargés.

4.4 Résultats de simulation

Pour évaluer notre approche, nous avons utilisé le simulateur de réseau P2P PeerSim. Utilisant le langage Java, nous avons implémenté les plannings pour chaque pair du système. Chaque planning est représenté sous forme d'une matrice. Pour nos tests, les matrices sont remplies d'une manière aléatoire. Pour respecter l'équité, nous imposons à chaque nœud du système de donner autant d'espace disque que celui dont il a besoin pour sauvegarder ses propres données. Nous avons lancé plusieurs simulations en faisant varier le nombre de nœuds dans le système. Le but de ces simulations était de comparer les protocoles de sauvegarde avec et sans planification.

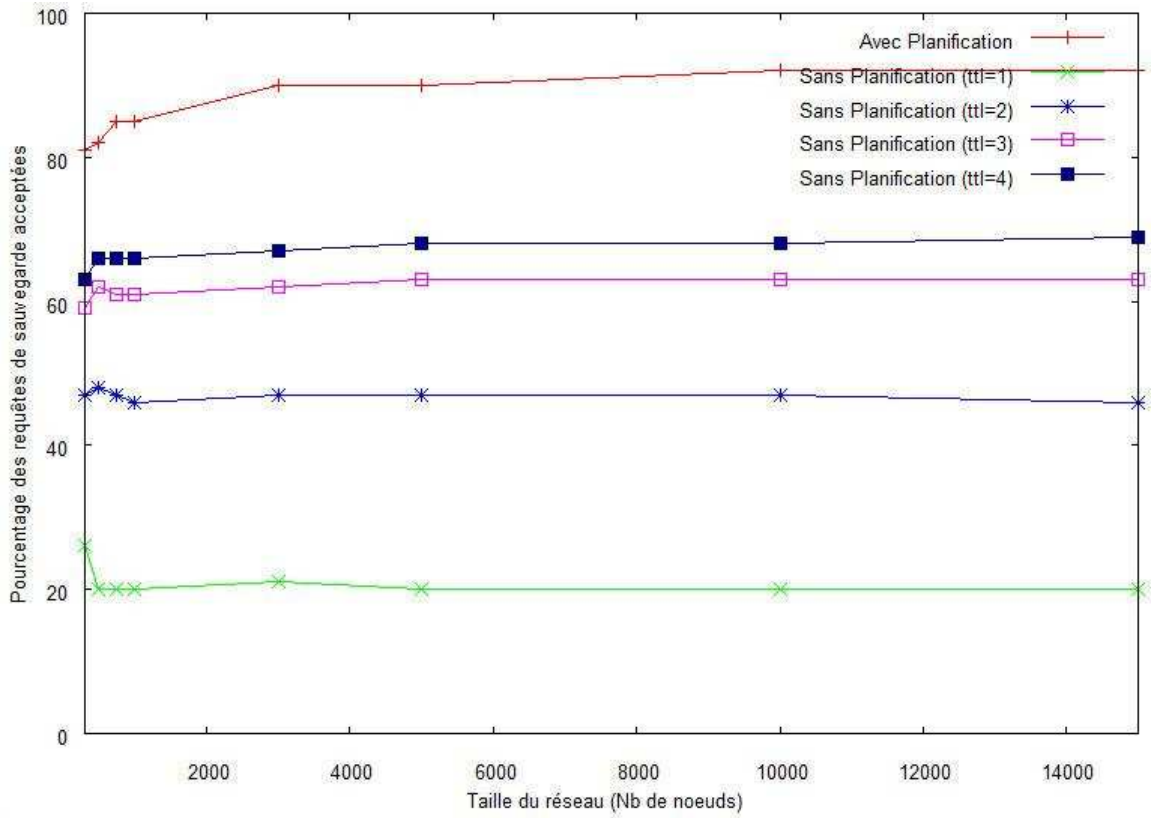


Figure 16 - Requêtes planifiées vs. Requêtes aléatoires

Dans la Figure 16, nous comparons la sauvegarde planifiée avec celle non planifiée. Dans la sauvegarde non planifiée, les requêtes sont envoyées d'une manière aléatoire. C'est-à-dire, si un nœud veut sauvegarder des données, il va contacter un nœud du réseau d'une façon aléatoire en lui demandant l'autorisation de lui envoyer des blocs de données à sauvegarder. La taille de notre réseau varie entre 300 et 15000 nœuds. Comme on peut constater dans cette figure, pour une sauvegarde planifiée, le pourcentage des requêtes acceptées varie entre 80% et 92%. Dans cette sauvegarde, chaque pair utilise la DHT pour effectuer sa recherche et trouver directement les pairs sur lesquels il peut effectuer sa sauvegarde. Dans la sauvegarde non planifiée, un pair doit chercher d'une manière aléatoire des pairs disponibles à la sauvegarde de ses données. Nous avons effectué plusieurs simulations en variant le ttl (time to live) des requêtes. Une valeur de ttl = 1 veut dire que chaque requête partant du pair qui effectue la recherche, atteint uniquement un seul pair. Pour un ttl = 2, si la réponse du pair est négative, la requête va continuer vers un autre pair du réseau (toujours d'une manière aléatoire) et ainsi de suite. Dans la figure 16, on remarque que même avec un ttl = 4 le pourcentage des requêtes acceptées ne dépasse pas 70%. D'après les simulations effectuées, nous avons constaté que, même en utilisant la sauvegarde planifiée, le pourcentage des

requêtes acceptées n'arrivait jamais à 100%. Après avoir analysé les résultats, nous avons découvert que les requêtes refusées l'étaient à cause d'un manque d'espace mémoire. Pour un certain créneau horaire, même lorsque le nombre des pairs qui souhaitent sauvegarder leurs données est inférieur au nombre des pairs prêts à effectuer un service, on peut, à un certain moment, avoir un pair disponible en terme de créneau horaire mais incapable de stocker des données à cause du manque d'espace mémoire. Si jamais nous changeons les paramètres de la simulation et que l'espace mémoire dédié aux services est plus élevé que celui dédié à la sauvegarde, ceci peut augmenter le pourcentage des requêtes acceptées.

Chapitre 5 Persistance des données dans les systèmes P2P

5.1 Introduction

Pour étudier la persistance des données, nous présentons dans cette partie un modèle statistique qui utilise les codes correcteurs d'erreurs. Comme nous l'avons vu dans le chapitre 3, les auteurs de [**28**] définissent la notion de persistance d'un groupe de nœuds (appelé cœur) formé par les nœuds ayant des données critiques, en vue de garantir que ces données critiques restent dans le système après un certain temps. Plusieurs facteurs ont été définis tel que la taille du système, la taille du cœur, le taux de mouvements dans le réseau (taux des entrées et sorties des nœuds). Dans l'approche proposée, les auteurs n'envisagent pas un découpage des fichiers en blocs. Les fichiers sont répliqués avec leur taille complète ce qui peut causer des problèmes dans le cas de fichiers volumineux. L'avantage du découpage de fichiers en blocs, c'est qu'il permet de les identifier indépendamment et de télécharger le même fichier simultanément à partir de plusieurs sources, ce qui permet d'accélérer le téléchargement. D'autre part la réplication de chaque bloc reste plus simple même dans le cas d'une connexion faible.

5.2 1ère approche – Persistance d'un noyau dans un système P2P

Dans un premier temps nous avons repris le travail de [**28**] et nous lui avons intégré la notion de découpage d'un fichier en blocs en utilisant les codes correcteurs d'erreurs. L'intérêt du découpage d'un fichier en blocs est de faciliter l'envoi des fichiers à travers le réseau. Aujourd'hui, beaucoup de fichiers ont une grande taille surtout les fichiers multimédia. Il n'est pas évident de transférer ce genre de fichiers en un seul bloc à travers le réseau. Dans le cas de découpage en blocs, l'usage des codes correcteurs permet de créer des blocs de redondances qui augmentent la chance de récupérer le fichier original. Nous supposons que notre système est composé de N nœuds. Un nœud peut quitter le système volontairement ou à cause d'un problème technique ou d'un crash. Soit c le pourcentage des nœuds qui quittent le système par unité de temps. Chaque unité de temps, cN nœuds quittent le système. Ces nœuds sont remplacés par cN autres nœuds. Comme dans [**28**], nous

supposons que les nœuds qui rentrent dans le système sont nouveaux et n'ont aucune connaissance de l'état du système.

Utilisant les codes correcteurs d'erreurs, un fichier est divisé en m blocs qui sont recodés en n fragments. Au départ, le système S est composé d'un cœur Q qui contient les n nœuds possédants les fragments (à raison d'un fragment par nœud). Le nombre des nœuds qui ne possèdent aucun fragment est $N - n$.

Soit $Q(\delta)$ le cœur après δ unités de temps. On s'intéresse à trouver la probabilité que $Q(\delta)$ contienne au moins m fragments qui peuvent être utilisés pour reconstruire le fichier original. c étant le pourcentage des nœuds qui quittent le système par unité de temps, on affirme que :

Lemme 1 Le nombre de nœuds qui auraient quitté le système après δ unités de temps est :

$$\left(1-(1-c)^{\delta}\right)N \tag{1}$$

La démonstration est faite par induction sur δ.Ceci est vrai pour $\delta = 0$ et pour $\delta = 1$. Supposons que l'hypothèse est vraie pour l'ordre p, le nombre de nœuds qui auraient quitté le système après p unités de temps est alors :

$$\left(1-(1-c)^{p}\right)N \tag{2}$$

Le nombre des nœuds qui restent dans le système est :

$$N-\left(1-(1-c)^{p}\right)N = N(1-c)^{p} \tag{3}$$

Le nombre de nœuds qui restent dans le système au temps $(p+1)$ sera :

$$N(1-c)^{p} - N(1-c)^{p}c = N(1-c)^{p}(1-c) = N(1-c)^{p+1} \tag{4}$$

Le nombre de ceux qui auraient quitté le système après $(p+1)$ unités de temps est :

$$N - N(1-c)^{p+1} = N\left(1-(1-c)^{p+1}\right) \tag{5}$$

Notre hypothèse est donc vraie pour l'ordre $(p+1)$.

En utilisant le résultat du Lemme 1, le nombre des nœuds qui auraient quitté le système durant δ unités de temps est :

$$\alpha = \left\lceil \left(1-(1-c)^{\delta}\right) N \right\rceil \tag{6}$$

Soit $x_1,..x_n$ un groupe de n nœuds dans le système $S(\delta)$. Quelle est la probabilité que l'intersection de ce groupe avec $Q(\delta)$ contienne au moins m nœuds ?

Notre cas peut être représenté de la manière suivante :

Le système S est une urne contenant N boules (nœuds), tel que, au départ, n boules sont vertes et représentent l'ensemble Q(cœur) des n nœuds qui possèdent les n fragments, alors que les $N - n$ boules restantes sont noires.

On tire aléatoirement α boules de l'urne et on les peint en rouge. Après avoir changé la couleur en rouge, on remet chacune de ces boules dans l'urne. Ces α boules représentent les nœuds qui auraient quitté le système au temps δ (au début elles étaient vertes ou noires). Le système $S(\delta)$ contient à nouveau N boules.

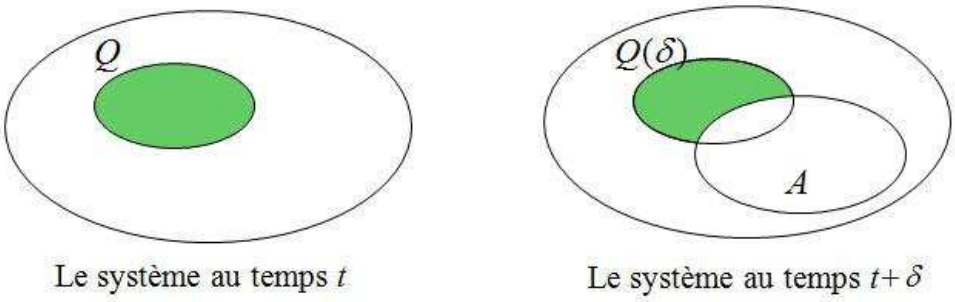


Figure 17 - Le système au temps t et au temps $t + \delta$

La Figure 17 représente le système au temps t et au temps $t' = t + \delta$. L'ensemble A représente les boules peintes en rouge. $Q(\delta)$ représente le cœur Q après avoir peint certaines de ses boules en rouge (ces boules représentent les nœuds du cœur qui ont quitté le système). Cela veut dire que l'ensemble $Q(\delta) \setminus A$ contient uniquement toutes les boules vertes. Soit β le nombre des boules de l'ensemble $A \cap Q(\delta)$, et λ le nombre des boules de l'ensemble $Q(\delta) \setminus A$.

A contient α boules. $Q(\delta)$ contient n boules. $A \cap Q(\delta) = E$ contient β boules. $A \cup Q(\delta)$ contient $(\alpha + n - \beta)$ boules.

$\alpha + n - \beta \leq N \Rightarrow \beta \geq \alpha + n - N$. Comme $\beta \geq 0$, on obtient $\beta \geq a = Max(0, \alpha + n - N)$.

D'autre part $\beta \leq n \; et \; \beta \leq \alpha \Rightarrow \beta \leq b = \min(\alpha, n)$. Donc $a \leq \beta \leq b$ et:

$$\Pr[\beta = k] = \frac{\binom{n}{k}\binom{N-n}{\alpha-k}}{\binom{N}{n}} = \Pr[\lambda = n-k] \tag{7}$$

On tire aléatoirement et successivement n boules $x_1,...,x_n$ de l'urne sans les remettre dans l'urne (Système $S(\delta)$ obtenu après δ unités de temps). Quelle est la probabilité que parmi ces n boules il y a au moins m de couleur verte.

La probabilité conditionnelle que parmi ces n boules il y a exectement p boules vertes, sachant que $\beta = k$ est :

$$\frac{\binom{n-k}{p}\binom{N-n+k}{n-p}}{\binom{N}{n}} \tag{8}$$

Et donc la probabilité conditionnelle que parmi ces n boules il y a un nombre de boules vertes $\geq m$, sachant que $\beta = k$ est :

$$\frac{\sum_{p=m}^{n}\binom{n-k}{p}\binom{N-n+k}{n-p}}{\binom{N}{n}} \tag{9}$$

Et la probabilité inconditionnelle de trouver un nombre de boules vertes $\geq m$ est :

$$\sum_{k=a}^{b}\left(\frac{\sum_{p=m}^{n}\binom{n-k}{p}\binom{N-n+k}{n-p}}{\binom{N}{n}}\right)\Pr[\beta = k] = \sum_{k=a}^{b}\sum_{p=m}^{n}\frac{\binom{n-k}{p}\binom{N-n+k}{n-p}\binom{n}{k}\binom{N-n}{\alpha-k}}{\binom{N}{n}\binom{N}{\alpha}} \tag{10}$$

5.3 Résultats de simulation

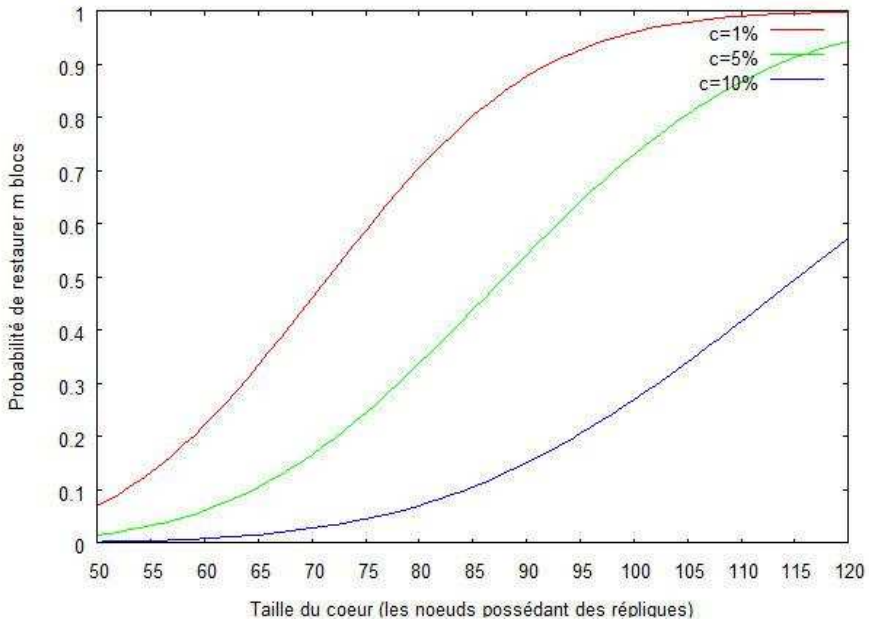


Figure 18 – Probabilité de récupérer *m* blocs

Dans la Figure 18, le réseau est composé de 1000 nœuds. Notre fichier à sauvegarder est divisé en 5 blocs. La taille du noyau (les nœuds possédant des répliques) varie entre 50 et 120 nœuds. Le taux de mobilité du réseau (*c*) est différent dans chacune des courbes. Nous avons calculé la probabilité de retrouver 5 nœuds (parmi ceux qui possèdent des répliques) dans le noyau après 10 unités de temps. Cette probabilité est calculée pour trois valeurs différentes de *c* qui sont 10%, 5% et 1%. Comme nous pouvons le constater, quand la valeur de *c* augmente, il faut avoir plus de nœuds dans le noyau pour obtenir une probabilité élevée alors que dans notre exemple, quand le taux de variation est de 1% et avec un noyau ayant 90 nœuds ou plus, nous arrivons à obtenir une probabilité élevée (entre 0.9 et 1) après 10 unités de temps alors qu'avec un taux de variation égal à 10% et avec un noyau de 120 nœuds, la probabilité reste inférieure à 0.6 après 10 unités de temps.

5.4 Synthèse

Comme on a vu dans la partie précédente, on a introduit le découpage de fichiers en blocs. Notre travail ne peut pas être comparé à celui de [24] car dans notre cas, le fichier est divisé en plusieurs blocs. La formule utilisée dans l'article [24] se présente comme un cas particulier de notre travail. En divisant le fichier en un nombre de blocs = 1, notre formule sera la même que celle présentée dans [24]. Cependant, le système présenté reste quand même loin de la réalité car les nœuds qui quittent le système ne reviennent jamais. À chaque fois

qu'un nœud quitte le système il perd toute connaissance de ce dernier et il est remplacé par un nouveau nœud qui n'a aucune connaissance du système. Cela veut dire que si jamais le nœud qui sort possède des données critiques (répliques), ces données seront perdues. Dans les systèmes réels, un nœud possédant des données critiques peut se déconnecter pour une certaine période et rejoindre plus tard le réseau. Un nœud peut quitter le système à cause d'un problème technique comme la déconnexion temporaire mais il peut quand même garder ses données critiques. Si on considère qu'à chaque déconnexion d'un nœud, celui-ci perd ses données, cela veut dire qu'à chaque fois qu'un nœud possédant des données critiques disparait du réseau, il faut faire de nouvelles répliques pour remplacer les répliques perdues et maintenir la persistance du cœur. Le fait de créer de nouvelles répliques a un certain coût. Dans ce papier, les codes de redondance tel que Reed Solomon ou Tornado ne sont pas utilisés. Dans la partie suivante, nous présentons un nouveau modèle de persistance qui utilise les codes correcteurs. Notre modèle est basé sur une chaîne de Markov pour présenter les différents états d'un nœud dans le système. Dans un premier temps, notre modèle utilise les codes correcteurs sans intégrer la réplication classique.

5.5 2ème approche : Modèle probabiliste basé sur une chaîne de Markov (Codage sans réplication)

Dans cette partie, nous proposons un modèle basé sur les codes correcteurs d'erreurs. Utilisant ces codes, un fichier est décomposé en m blocs $B_1, \cdots, B_m$, qui sont ensuite codés en n fragments, où $n > m$. Le rapport $\tau = \frac{m}{n} < 1$ est le taux de codage. On lance ces n fragments sur un réseau formé de N nœuds (N peut représenter le nombre des personnes qui déclarent être disposé à coopérer) à raison d'un fragment au plus par nœud. Pour récupérer le fichier, on a besoin de récupérer au moins m fragments distincts. Nous sommes au départ en présence d'un système S composé de n nœuds contenant les n fragments et d'une partie ne contenant aucun fragment dont le nombre des nœuds est $N - n$. Un nœud peut quitter le système ou le réintégrer à n'importe quel moment. Si après un intervalle de k unités de temps, on contacte aléatoirement n nœuds du réseau $S(k)$, quelle est la probabilité que ce groupe possède au moins m fragments distincts ? En d'autre terme quelle est la probabilité qu'on puisse récupérer le fichier en contactant ce groupe ?

Soit $S(t)$ le système au temps *t*. Le système $S(t+1)$ au temps *t* + 1 ne dépend que de $S(t)$ et non des états précédents. Il peut être représenté par une **chaîne de Markov**.

En mathématiques, une **chaîne de Markov** est un processus stochastique dans lequel la prédiction du futur à partir du présent ne nécessite pas la connaissance du passé. Une chaîne de Markov en temps discret est une séquence de variables aléatoires. L'ensemble de leurs valeurs possibles est appelé l'*espace d'états*, la valeur X_n étant l'état du processus au moment *n*.

Si la distribution de probabilité conditionnelle de X_{n+1} sur les états passés est une fonction de X_n seul, alors :

$$P\left(X_{n+1}=x\middle|X_0,X_1,\cdots,X_n\right)=P\left(X_{n+1}=x\middle|X_n\right) \tag{1}$$

où *x* est un état quelconque du processus. Comme l'espace des états est fini, alors la distribution de probabilité peut être représentée par une matrice stochastique :

$$P=\left(p_{i,j}\right) \tag{2}$$

appelée *matrice de transition d'un pas*, où

$$p_{i,j}=P\left(X_{n+1}=j\middle|X_n=i\right) \tag{3}$$

$P=\left(p_{i,j}\right)$ est la matrice de transition pour un pas. La somme des éléments d'une ligne vaut toujours 1. Ainsi P^k est la matrice de transition pour *k* pas.

Le système au temps *t* est constitué par les ensembles de nœuds $\left(A_1,A_2,A_3,A_4\right)$ où A_1 et A_2 sont les ensembles de nœuds connectés, alors que A_3 et A_4 sont les ensembles de nœuds non connectés. A_1 et A_3 sont constitués de nœuds qui ont des fragments, alors que A_2 et A_4 sont constitués de nœuds qui n'ont pas de fragments. A_i contient N_i nœuds. $N_1+N_2+N_3+N_4=N$ est constant.

Soit *c* le pourcentage des nœuds qui quittent (ou réintègrent) le système en une unité de temps. Alors $p=\frac{c}{100}$ est la probabilité qu'un nœud change sa position « quitte (ou réintègre)

le système » après une unité de temps ; $q = 1 - p$ est la probabilité qu'un nœud ne change pas sa position après une unité de temps.

La matrice de transition d'un pas est :

$$P = \begin{pmatrix} q & 0 & p & 0 \\ 0 & q & 0 & p \\ p & 0 & q & 0 \\ 0 & p & 0 & q \end{pmatrix} = qI + pJ \tag{4}$$

où

$$J = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{5}$$

On remarque que $J^2 = I$,

$$P^2 = \left(q^2 + p^2\right) I + 2pq\, J \tag{6}$$

$$u^0 = \left(1, \quad 0, \quad 0, \quad 0\right) \tag{7}$$

représente un nœud de A_1 à l'instant 0.

$$u^1 = u^0 P = \left(q, \quad 0, \quad p, \quad 0\right) \tag{8}$$

signifie qu'à l'instant 1, ce nœud a la probabilité q de rester dans A_1, et la probabilité p d'être dans A_3 ;

$$u^2 = u^1 P = u^0 P^2 \,;\, \ldots \,;\, u^k = u^0 P^k \tag{9}$$

En supposant que

$$X^0 = \left(N_1, \quad N_2, \quad N_3, \quad N_4\right) \tag{10}$$

ce qui signifie que A_i contient N_i nœuds à l'instant 0, alors

$$X^1 = X^0 P \text{ et } X^k = X^0 P^k \tag{11}$$

On ne change rien à la généralité du problème en remplaçant X^0 par X^t et X^k par X^{t+k}.

Dans notre cas

$$X^0 = (n, \quad N_2, \quad 0, \quad N_4) \tag{12}$$

avec $n + N_2 + N_4 = N$, ce qui permet de déterminer

$$X^k = X^0 P^k = (a_1, \quad a_2, \quad a_3, \quad a_4) \tag{13}$$

Après un intervalle de *k* unités de temps, notre système est composé de $\lceil a_1 \rceil = s_1$ nœuds qui contiennent des fragments, et $\lceil a_2 \rceil = s_2$ nœuds qui ne contiennent pas de fragments. On contacte alors aléatoirement *n* nœuds du réseau *S*(*k*). Quelle est la probabilité que ce groupe possède au moins *m* fragments ? En d'autre terme quelle est la probabilité qu'on puisse récupérer le fichier (ou probabilité de succès) en contactant ce groupe ?

Étant donné qu'on contacte aléatoirement n'importe quel groupe de *n* nœuds dans *S*(*k*), la probabilité que ce groupe possède au moins *m* fragments est :

$$\frac{\sum_{i=m}^{S_1} \binom{S_1}{i} \binom{S_2}{n-i}}{\binom{S_1 + S_2}{n}} \tag{14}$$

5.5.1 Résultats

Paramètre	Description
c	Pourcentage de nœuds qui rejoignent ou quittent le système par unité de temps
m	Nombre de blocs d'un fichier avant le codage
n	Nombre de fragments après le codage
k	Nombre d'unités de temps

Figure 19 - Tableau des paramètres utilisés dans les simulations

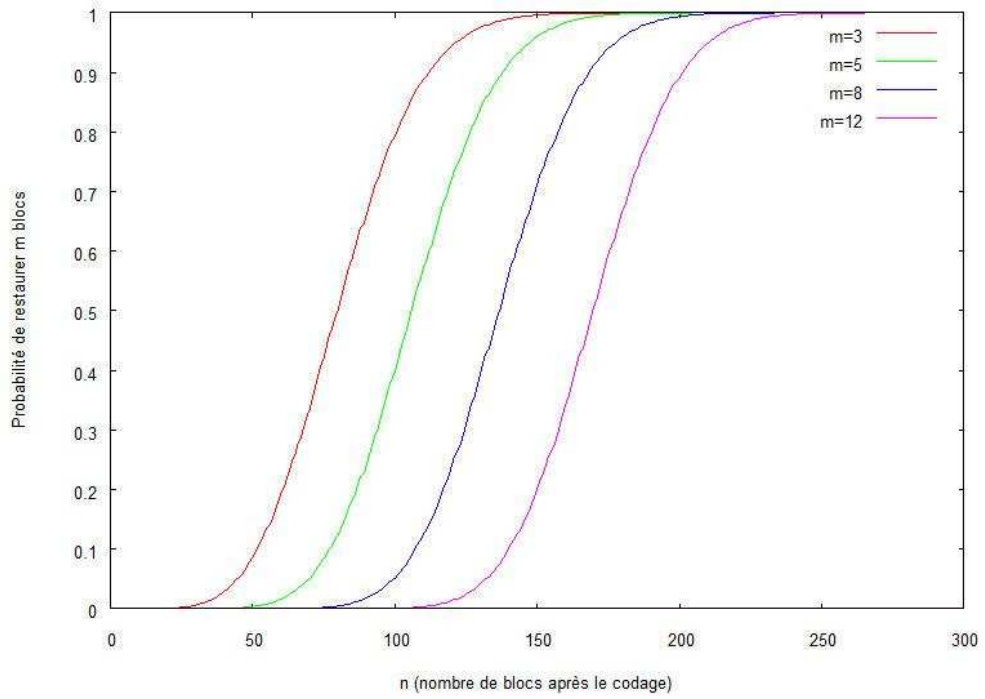


Figure 20 - La probabilité de récupérer au moins *m* fragments

Dans la Figure 20, le système est composé de 5000 nœuds (2000 connectés et 3000 déconnectés). Au départ les nœuds déconnectés ne possèdent aucune donnée. c = 0.1, k = 100. Pour plusieurs valeurs de *m,* on compare la probabilité de restaurer les *m* blocs (restaurer le fichier) par rapport à la valeur de *n*. Comme nous pouvons le constater, quand *m* a une valeur plus élevée, on a besoin de plus de blocs codés (fragments) pour restaurer les *m* blocs après *k* unités de temps.

Ce nombre de blocs de redondance est élevé. Dans la Figure 20, quand m = 5, pour une valeur de probabilité entre 0.9 et 0.99, on a besoin d'un nombre de fragments entre 135 et 175 qui est une valeur élevée. Cela est dû au fait que la réplication classique n'est pas utilisée. Dans la partie suivante nous allons utiliser le même modèle avec intégration de la réplication classique pour réduire le nombre de fragments de redondance à créer. Une fois les fragments de redondance créés, il suffira de faire quelques réplications classiques de ces fragments pour augmenter leur disponibilité et ainsi augmenter la chance de récupérer le fichier initial.

5.6 Modèle intégrant la réplication classique

Dans cette partie, on suppose qu'après le codage des blocs en un certains nombre *r* de fragments, on fait de chaque fragment un nombre de répliques égal à *l* et on désigne par *n* = *rl* le nombre total des répliques. Ces *n* répliques sont distribuées dans le réseau formé de *N* nœuds (N peut représenter le nombre de personnes qui déclarent être disposées à coopérer) à raison d'une réplique au plus par nœud. Pour récupérer le fichier, on a besoin de récupérer au moins *m* fragments distincts. Comme dans la partie précédente, notre système est représenté par une chaîne de Markov.

Une première approche du problème consiste, puisque au départ tous les fragments avaient le même nombre de répliques, à considérer qu'après *k* unités de temps, notre système est composé de $\left\lceil \frac{a_1}{r} \right\rceil = \lambda$ répliques pour chacun des fragments F_i, et de $\lceil a_2 \rceil = s_2$ nœuds qui ne contiennent pas de répliques. Le calcul de la probabilité est semblable au tirage par poignée dans une urne multicolore.

Tirages par poignées dans une urne multicolore : Soit *U* une urne contenant *N* boules de *r* couleurs différentes $c_1, c_2, \cdots, c_r$. On note N_j le nombre des boules de couleur c_j, $p_j = \frac{N_j}{N}$ étant dès lors la proportion de boules de couleur c_j dans l'urne *U*. On tire simultanément *n* boules de l'urne *U*. Quelle est la probabilité que se trouvent parmi elles k_j boules de couleur c_j, pour $1 \le j \le r$? Cette probabilité est nulle si les conditions suivantes ne sont pas satisfaites : $k_1 + k_2 + \cdots + k_r = n$ et $0 \le k_j \le N_j$ pout tout *j* tel que $1 \le j \le r$. Aussi nous nous plaçons dans le cas où ces conditions sont satisfaites.

Un résultat de l'épreuve est un échantillon de *n* boules prises parmi *N* : il y a $\binom{N}{n}$ échantillons possibles et ils sont équiprobables. Quel est le nombre des échantillons contenant k_j boules de couleur c_j , pour $1 \le j \le r$? Pour chaque couleur c_j, il y a $\binom{N_j}{k_j}$ choix possibles des k_j boules de couleur c_j ,donc en tout $\binom{N_1}{k_1} \times \binom{N_2}{k_2} \times \cdots \times \binom{N_r}{k_r}$ échantillons qui conviennent. La probabilité cherchée est alors

$$\frac{\binom{N_1}{k_1}\times\binom{N_2}{k_2}\times\cdots\times\binom{N_r}{k_r}}{\binom{N}{n}} \tag{1}$$

On obtient le même résultat si, au lieu de tirer les *n* boules simultanément, on les tire une à une sans remettre la boule tirée dans l'urne

Dans notre cas

$$N_1 = N_2 = \cdots = N_r = \left\lceil \frac{a_1}{r} \right\rceil = \lambda \tag{2}$$

$N_1 + N_2 + \cdots + N_r = s_1 = \lambda r$, $n = rl$. La probabilité d'obtenir k_j répliques du fragment F_j pour $1 \le j \le r$ et $n - (k_1 + k_2 + \cdots + k_r)$ nœuds ne contenant pas de répliques est :

$$\frac{\binom{N_1}{k_1}\times\binom{N_2}{k_2}\times\cdots\times\binom{N_r}{k_r}\times\binom{s_2}{n-(k_1+k_2+\cdots+k_r)}}{\binom{s_1+s_2}{n}} \tag{3}$$

La probabilité de récupérer le fichier est obtenue lorsque au moins *m* des valeurs $k_1, k_2, \cdots, k_r$ sont différentes de zéro.

$$\sum_{\text{pour au moins m des } k_i>0} \frac{\binom{N_1}{k_1}\times\binom{N_2}{k_2}\times\cdots\times\binom{N_r}{k_r}\times\binom{s_2}{n-(k_1+k_2+\cdots+k_r)}}{\binom{s_1+s_2}{n}} =$$

$$\sum_{\text{pour au moins m des } k_i>0} \frac{\binom{\lambda}{k_1}\times\binom{\lambda}{k_2}\times\cdots\times\binom{\lambda}{k_r}\times\binom{s_2}{n-(k_1+k_2+\cdots+k_r)}}{\binom{s_1+s_2}{n}} \tag{4}$$

Le fichier est perdu si le nombre des $k_i > 0$ est $\le m-1$. La probabilité π de perdre le fichier est :

$$\pi = \sum_{j=0}^{m-1} \sum_{h_1,h_2,\cdots,h_j>0} \frac{\binom{\lambda}{h_1}\times\binom{\lambda}{h_2}\times\cdots\times\binom{\lambda}{h_j}\times\binom{s_2}{n-\left(h_1+h_2+\cdots+h_j\right)}}{\binom{s_1+s_2}{n}} \qquad (5)$$

La probabilité de récupérer le fichier est $1-\pi$.

5.6.1 Réduction de la matrice de transition

La matrice de transition d'un pas est

$$P = \begin{pmatrix} q & 0 & p & 0 \\ 0 & q & 0 & p \\ p & 0 & q & 0 \\ 0 & p & 0 & q \end{pmatrix} = (1-p)I + pJ \qquad (1)$$

où

$$J = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \qquad (2)$$

est une matrice symétrique diagonalisable semblable à une matrice Q, où $P = BQB^{-1}$. Son polynôme caractéristique est

$$|P-\lambda I| = \begin{vmatrix} q-\lambda & 0 & p & 0 \\ 0 & q-\lambda & 0 & p \\ p & 0 & q-\lambda & 0 \\ 0 & p & 0 & q-\lambda \end{vmatrix} = \begin{vmatrix} q-\lambda-\dfrac{p^2}{q-\lambda} & 0 & 0 & 0 \\ 0 & q-\lambda-\dfrac{p^2}{q-\lambda} & 0 & 0 \\ p & 0 & q-\lambda & 0 \\ 0 & p & 0 & q-\lambda \end{vmatrix} \qquad (3)$$

(en remplaçant la ligne L_1 par $L_1 - \frac{pL_3}{q-\lambda}$ et L_2 par $L_2 - \frac{pL_4}{q-\lambda}$) $= \left[(q-\lambda)^2 - p^2\right]^2 =$ $(q-\lambda-p)^2(q-\lambda+p)^2 = (1-2p-\lambda)^2(1-\lambda)^2$.

Les valeurs propres sont donc $\lambda_1 = 1$ et $\lambda_2 = 1-2p$, qui sont des valeurs propres doubles. Les vecteurs propres associés à $\lambda_1 = 1$ sont donnés par l'équation :

$$(P-\lambda_1 I)V=0 \tag{4}$$

$$\begin{pmatrix} -p & 0 & p & 0 \\ 0 & -p & 0 & p \\ p & 0 & -p & 0 \\ 0 & p & 0 & -p \end{pmatrix}\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}=\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{5}$$

Ce qui donne : $a=c$ et $b=d$, donc

$$V=\begin{pmatrix} a \\ b \\ a \\ b \end{pmatrix}=a\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}+b\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}=aV_1+bV_2 \tag{6}$$

Les vecteurs propres associés à $\lambda_2=1-2p$ sont donnés par l'équation :

$$(P-\lambda_2 I)V=0 \tag{7}$$

$$\begin{pmatrix} p & 0 & p & 0 \\ 0 & p & 0 & p \\ p & 0 & p & 0 \\ 0 & p & 0 & p \end{pmatrix}\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}=\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{8}$$

Ce qui donne : $a=-c$ et $b=-d$, donc

$$V=\begin{pmatrix} a \\ b \\ -a \\ -b \end{pmatrix}=a\begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix}+b\begin{pmatrix} 0 \\ 1 \\ 0 \\ -1 \end{pmatrix}=aV_3+bV_4 \tag{9}$$

La matrice de passage *B* a ses colonnes formées par les vecteurs propres V_1, V_2, V_3, V_4.

$$B=\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \tag{10}$$

La matrice diagonale semblable à *P* a pour éléments diagonaux les valeurs propres

$$Q=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1-2p & 0\\ 0 & 0 & 0 & 1-2p\end{pmatrix} \tag{11}$$

et

$$B^{-1}=\begin{pmatrix}\frac{1}{2} & 0 & \frac{1}{2} & 0\\ 0 & \frac{1}{2} & 0 & \frac{1}{2}\\ \frac{1}{2} & 0 & -\frac{1}{2} & 0\\ 0 & \frac{1}{2} & 0 & -\frac{1}{2}\end{pmatrix} \tag{12}$$

est obtenue en résolvant le système d'équations : $\begin{cases}V_1=e_1+e_3 & V_2=e_2+e_4\\ V_3=e_1-e_3 & V_4=e_2-e_4\end{cases}$; ce qui donne :

$$\begin{cases}e_1=\frac{1}{2}(V_1+V_3) & e_2=\frac{1}{2}(V_2+V_4)\\ e_3=\frac{1}{2}(V_1-V_3) & e_4=\frac{1}{2}(V_2-V_4)\end{cases}$$

$$P^k=BQB^{-1}BQB^{-1}\cdots BQB^{-1}=BQ^kB^{-1}=\alpha I+\beta J \tag{13}$$

où $\alpha=\frac{1}{2}\left[1+(1-2p)^k\right]$ et $\beta=\frac{1}{2}\left[1-(1-2p)^k\right]$

$$P^k=\begin{pmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{pmatrix}\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & (1-2p)^k & 0\\ 0 & 0 & 0 & (1-2p)^k\end{pmatrix}\begin{pmatrix}\frac{1}{2} & 0 & \frac{1}{2} & 0\\ 0 & \frac{1}{2} & 0 & \frac{1}{2}\\ \frac{1}{2} & 0 & -\frac{1}{2} & 0\\ 0 & \frac{1}{2} & 0 & -\frac{1}{2}\end{pmatrix}=$$

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2}(1-2p)^k & 0 & -\frac{1}{2}(1-2p)^k & 0 \\ 0 & \frac{1}{2}(1-2p)^k & 0 & -\frac{1}{2}(1-2p)^k \end{pmatrix} = \alpha I + \beta J$$

Lorsque k tend vers l'infini, $(1-2p)^k$ tend vers 0 ; α et β tendent vers la même limite $\frac{1}{2}$.

D'où : $\lim_{k\to\infty} P^k = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}$.

Comme $\lim_{k\to\infty}\alpha = \lim_{k\to\infty}\beta = \frac{1}{2}$, si on a r fragments et qu'au départ on distribue $2l$ copies de chaque fragments, cela veut dire que si $X^0 = (2l, 2l, \cdots, 2l, N_1, 0, 0, \cdots, 0, N_2)$ alors quand k approche de l'infini, on atteint l'état stationnaire.

$$X = \left(l, l, \cdots, l, \frac{N_1 + N_2}{2}, l, l, \cdots, l, \frac{N_1 + N_2}{2}\right) \tag{14}$$

Il serait intéressant de choisir l de façon à ce que la valeur minimum de la probabilité (celle obtenue à l'état stationnaire) soit largement suffisante. Quand la valeur de c augmente, le système atteint plus rapidement l'état stationnaire. Comme au départ tous les fragments avaient le même nombre de répliques, il semble normal de considérer qu'ils auront le même nombre de répliques après k unités de temps.

5.6.2 Fragments avec un nombre différent de répliques

Supposons qu'à un certain temps t, tous les fragments ne possèdent pas le même nombre de répliques. Le système au temps t est constitué des ensembles de nœuds

$\left(A_1, A_2, \cdots, A_r, B, A'_1, A'_2, \cdots, A'_r, B'\right)$ ou $A_i\ ; 1 \le i \le r$ est l'ensemble des nœuds connectés qui possèdent des répliques du fragment F_i, alors que $A'_i\ ; 1 \le i \le r$ est l'ensemble des nœuds déconnectés qui possèdent des répliques du fragment F_i. B est l'ensemble des nœuds connectés qui n'ont pas de répliques alors que B' est l'ensemble des nœuds déconnectés qui n'ont pas de répliques.

Soit $X^t = \left(a_1, a_2, \cdots, a_r, b, a_1^{'}, a_2^{'}, \cdots, a_r^{'}, b^{'}\right)$, ou $a_i\ ; 1 \le i \le r$ est le nombre des nœuds connectés qui possèdent des répliques du fragment F_i, $a_i^{'}\ ; 1 \le i \le r$ est le nombre des nœuds déconnectés qui possèdent des répliques du fragment F_i, b est le nombre des nœuds connectés qui n'ont pas des répliques et b' est le nombre de nœuds déconnectés qui n'ont pas de répliques.

La matrice de transition d'un pas est :

$$P = \begin{pmatrix} q & 0 & \cdots & 0 & 0 & p & 0 & \cdots & 0 & 0 \\ 0 & q & \cdots & 0 & 0 & 0 & p & \cdots & 0 & 0 \\ \cdots & \cdots & \ddots & \cdots & \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & 0 & \cdots & q & 0 & 0 & 0 & \cdots & p & 0 \\ 0 & 0 & \cdots & 0 & q & 0 & 0 & \cdots & 0 & p \\ p & 0 & \cdots & 0 & 0 & q & 0 & \cdots & 0 & 0 \\ 0 & p & \cdots & 0 & 0 & 0 & q & \cdots & 0 & 0 \\ \cdots & \cdots & \ddots & \cdots & \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & 0 & \cdots & p & 0 & 0 & 0 & \cdots & q & 0 \\ 0 & 0 & \cdots & 0 & p & 0 & 0 & \cdots & 0 & q \end{pmatrix} \qquad (1)$$

La fonction

$$\begin{aligned} &\det\left(P - \lambda I\right) = \left[\left(q-\lambda\right)^2 - p^2\right]^{r+1} \\ &= \left(q-\lambda-p\right)^{r+1}\left(q-\lambda+p\right)^{r+1} = \left(1-2p-\lambda\right)^{r+1}\left(1-\lambda\right)^{r+1} \end{aligned} \qquad (2)$$

est appelée polynôme caractéristique de *P*. Les racines de ce polynôme sont les valeurs propres de la matrice $\lambda_1 = 1$ et $\lambda_2 = 1-2p$. La décomposition spectrale définit *P* comme un produit $P = BQB^{-1}$, où *Q* est une matrice diagonale et *B* est une matrice inversible convenable. Etant donné la décomposition spectrale, la puissance *k* de *P* peut être calculée par

$$P^k = BQB^{-1}BQB^{-1}\cdots BQB^{-1} = BQ^kB^{-1} = \alpha I + \beta J \tag{3}$$

et la puissance d'une matrice diagonale peut être calculée en prenant les puissances correspondantes des coefficients diagonaux.

La matrice de transition de *k* pas est :

$$P^k = \begin{pmatrix} \alpha & 0 & \cdots & 0 & 0 & \beta & 0 & \cdots & 0 & 0 \\ 0 & \alpha & \cdots & 0 & 0 & 0 & \beta & \cdots & 0 & 0 \\ \cdots & \cdots & \ddots & \cdots & \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & 0 & \cdots & \alpha & 0 & 0 & 0 & \cdots & \beta & 0 \\ 0 & 0 & \cdots & 0 & \alpha & 0 & 0 & \cdots & 0 & \beta \\ \beta & 0 & \cdots & 0 & 0 & \alpha & 0 & \cdots & 0 & 0 \\ 0 & \beta & \cdots & 0 & 0 & 0 & \alpha & \cdots & 0 & 0 \\ \cdots & \cdots & \ddots & \cdots & \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & 0 & \cdots & \beta & 0 & 0 & 0 & \cdots & \alpha & 0 \\ 0 & 0 & \cdots & 0 & \beta & 0 & 0 & \cdots & 0 & \alpha \end{pmatrix} = \alpha I + \beta J \tag{4}$$

où $\alpha = \frac{1}{2}\left[1+(1-2p)^k\right]$ et $\beta = \frac{1}{2}\left[1-(1-2p)^k\right]$

$$X^{t+k} = (\alpha a_1 + \beta a_1^{'}, \alpha a_2 + \beta a_2^{'}, \cdots, \alpha a_r + \beta a_r^{'}, \alpha b + \beta b^{'}, \alpha a_1^{'} + \beta a_1, \alpha a_2^{'} + \beta a_2, \cdots,$$

$$\alpha a_r^{'} + \beta a_r, \alpha b^{'} + \beta b) = \left(y_1, \quad y_2, \quad \cdots \quad y_r, \quad Y_2 \quad y_1^{'} \quad y_2^{'} \quad \cdots \quad y_r^{'} \quad Y_2^{'}\right) \tag{5}$$

$$N_i = \lceil y_i \rceil = \lceil \alpha a_i + \beta a_i^{'} \rceil \tag{6}$$

$$s_1 = \sum_{i=1}^{r} N_i \tag{7}$$

$$s_2 = \lceil Y_2 \rceil = \lceil \alpha b + \beta b^{'} \rceil \tag{8}$$

Etant donné qu'on contacte aléatoirement un ensemble de n de nœuds du le système $S(t+k)$, la probabilité de restaurer au moins *m* fragments distincts est obtenue lorsque au moins *m* des valeurs $k_1, k_2, \cdots, k_r$ sont différents de zéro. Cette probabilité est :

$$\sum_{\text{pour au moins } m \text{ des } k_i>0} \frac{\binom{N_1}{k_1}\times\binom{N_2}{k_2}\times\cdots\times\binom{N_r}{k_r}\times\binom{s_2}{n-(k_1+k_2+\cdots+k_r)}}{\binom{s_1+s_2}{n}} \tag{9}$$

La probabilité de perdre le fichier est:

$$\pi=\frac{\binom{s_2}{n}}{\binom{s_1+s_2}{n}}+\sum_{j=1}^{r}\sum_{a=1}^{k_j}\binom{N_j}{a}\times\frac{\binom{s_2}{n-a}}{\binom{s_1+s_2}{n}}+\sum_{1\le j_1<j_2\le r}\sum_{a=1}^{k_{j_1}}\sum_{b=1}^{k_{j_2}}\binom{N_{j_1}}{a}\times\binom{N_{j_2}}{b}\times\frac{\binom{s_2}{n-a-b}}{\binom{s_1+s_2}{n}}+\cdots$$
$$\cdots+\sum_{1\le j_1<\cdots<j_{m-1}\le r}\sum_{a_1=1}^{k_{j_1}}\cdots\sum_{a_{m-1}=1}^{k_{j_{m-1}}}\binom{N_{j_1}}{a_1}\times\cdots\times\binom{N_{j_{m-1}}}{a_{m-1}}\times\frac{\binom{s_2}{n-a_1-\cdots-a_{m-1}}}{\binom{s_1+s_2}{n}} \tag{10}$$

Le premier terme correspond à $j=0$ (Cela veut dire que aucun des n nœuds contactés ne possède pas de répliques). Dans le second terme $j=1\ jusqu'$à r veut dire que seulement un des n nœuds contactés possède des répliques. Dans le 3ème terme, seulement deux des n nœuds contactés possèdent des répliques. Dans le dernier seulement $(m-1)$ des n nœuds contactés possèdent des répliques.

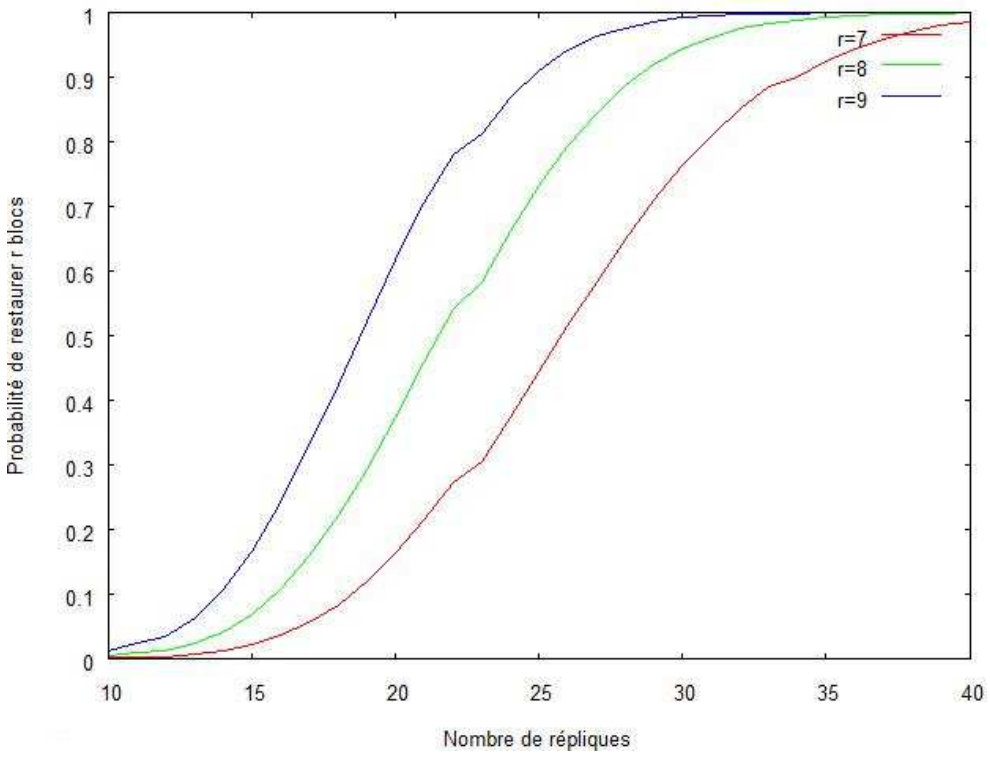
La probabilité de restaurer le fichier est donc $1-\pi$.

Comme nous pouvons le constater, les formules qui utilisent le codage erasure avec la réplication classique contiennent plusieurs paramètres qui peuvent être contrôlés de différentes façons dans les applications distribuées. Par exemple un nombre de blocs égal à un $(m=1)$ veut dire que les fichiers ne sont pas découpés en blocs et que le fichier est répliqué avec sa taille initiale. Si jamais on considère que le nombre de blocs après le codage est égal au nombre de blocs avant le codage $(m=n)$, cela veut dire que le découpage en blocs classique est utilisé sans le codage erasure. Si on ne veut pas intégrer la réplication classique, il suffit de mettre le paramètre l à un $(l=1)$ pour être dans le cas du codage erasure sans réplication classique. Les paramètres de notre formule peuvent être modifiés d'une manière flexible selon le cas qu'on désire étudier.

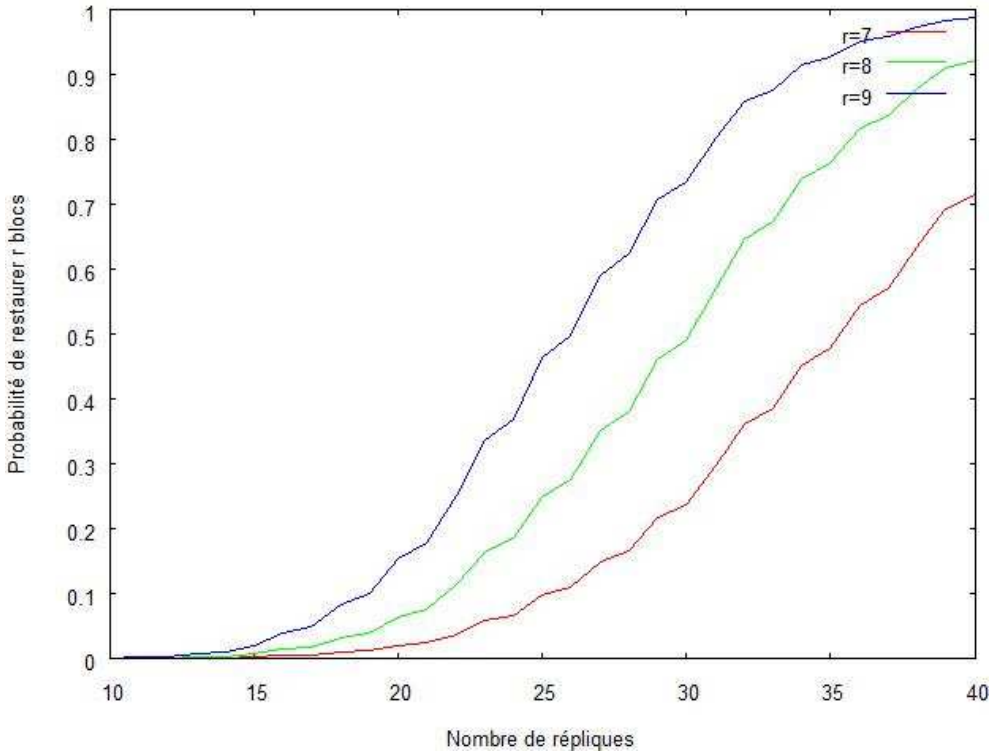
5.7 Résultats de simulation

Cette section exploite les formules précédentes pour relier plusieurs paramètres entre eux et fournir des relations entre ces paramètres. À garder à l'esprit que, dans un réseau P2P,

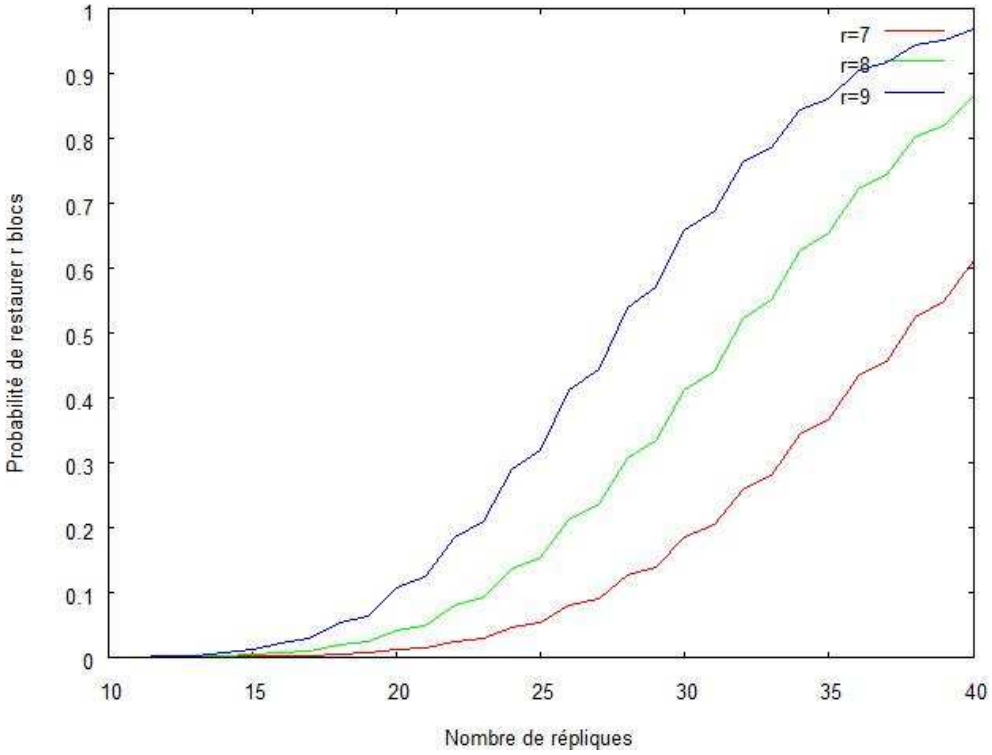
le taux de mouvement des nœuds peut changer d'un système à un autre. Ce paramètre est observé et non choisi manuellement au hasard. On assume que la capacité de stockage de chaque nœud du système est suffisante et que nous n'avons pas de soucis en ce qui concerne l'espace mémoire de chaque nœud.



(a): Le taux de mouvement est 0.1



(b): Le taux de mouvement est 1



(c): Le taux de mouvement est 10

Figure 21 - Probabilité de restaurer le fichier correspondant à plusieurs taux de codage erasure.

Dans la Figure 21, notre système est composé de 10 000 nœuds. Au départ 4000 nœuds sont connectés et 6000 nœuds déconnectés. Un fichier est divisé en 5 blocs qui sont codés en *r* blocs erasure (fragments). Le taux de mouvement *c* est différent dans chacun des graphes (a, b et c). Nous calculons la probabilité de restaurer *r* fragments après 100 unités de temps pour différentes valeurs de *r*. Comme nous pouvons le remarquer dans chaque graphe, lorsque le nombre *r* des blocs redondants (erasure) augmente, le nombre *l* des répliques à faire diminue et donne une bonne probabilité de restaurer le fichier. Le nombre *l* de répliques est lié au taux de codage erasure. Plus le nombre *r* de blocs erasure diminue, plus on ressent le besoin de faire un plus grand nombre de répliques pour garder une probabilité élevée de récupérer le fichier. Pareillement, si on augmente le nombre de blocs de redondance, le nombre de répliques diminue. En comparant les trois graphes (a, b et c) nous pouvons remarquer que lorsque le taux de mouvement *c* du système augmente, nous avons besoin de plus de répliques pour garder une valeur élevée de la probabilité. Cela est dû au fait que lorsque le taux de mouvement augmente, plus de nœuds peuvent quitter le système chaque unité de temps. Pour cela nous aurons besoins d'un nombre plus grand de répliques pour maintenir une valeur élevée de la probabilité.

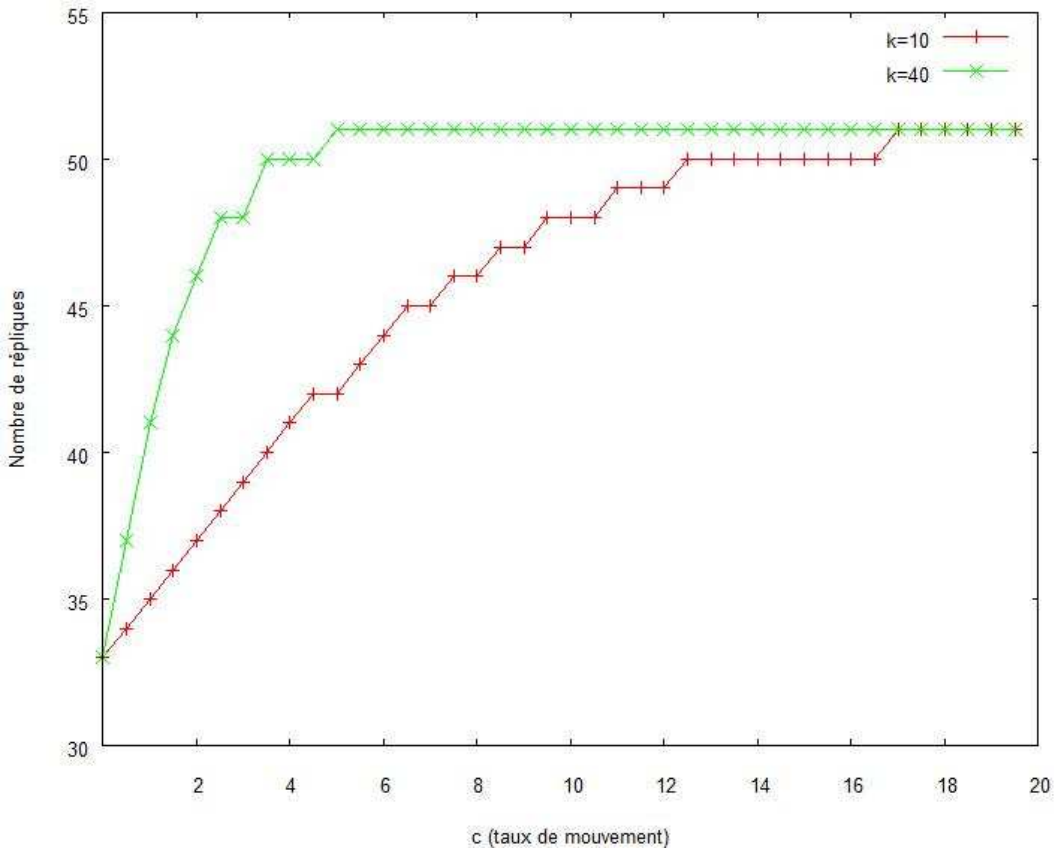


Figure 22 - Etat stationnaire

Dans la Figure 22, nous avons la même taille de réseau que celle du scénario précédant. Un fichier est divisé en 5 blocs qui sont codés en 8 blocs erasure (fragments). Nous calculons, lorsque le taux de mouvement du système croit, le nombre de répliques nécessaires pour obtenir (après k unités de temps), une probabilité de restauration du fichier égale à 0.99. Dans cette figure, nous avons deux courbes qui représentent deux valeurs de k. Comme nous pouvons le remarquer, pour les deux courbes, lorsque le taux de mouvement du système augmente, nous avons besoin de plus de répliques pour maintenir la probabilité à 0.99. Pour les deux courbes, nous atteignons l'état stationnaire lorsque le taux de mouvement augmente.

5.8 Temps nécessaire pour se rapprocher de l'état stationnaire

Plaçons nous dans le cas de m blocs $B_1,\cdots,B_m$, codés en r fragments $F_1,\cdots,F_r$. On fait de chaque fragment un nombre de répliques égal à $l = 2\ l'$ et on désigne par $n = rl$ le nombre

total des répliques. On lance ces n répliques sur un réseau formé de N nœuds à raison d'une réplique au plus par nœud.

Le calcul se fait de la façon suivante : On choisit d'abord c ; ceci permet de déterminer p puis q. On fixe un intervalle de temps k. On calcule ensuite

$$P^k = (qI + pJ)^k = \alpha I + \beta J \tag{1}$$

où

$$\alpha = \frac{1}{2}\left[1 + (1-2p)^k\right] \text{ et } \beta = \frac{1}{2}\left[1 - (1-2p)^k\right] \tag{2}$$

Soit

$$X^0 = (l, \quad l, \quad \cdots, \quad l, \quad N_1, \quad 0, \quad 0, \quad \cdots, \quad 0, \quad N_2) \tag{3}$$

avec

$$rl + N_1 + N_2 = N \tag{4}$$

On calcule le produit

$$X^k = X^0 P^k = (\alpha l, \quad \alpha l, \quad \cdots, \quad \alpha l, \quad \alpha N_1 + \beta N_2, \quad \beta l, \quad \beta l, \quad \cdots, \quad \beta l, \quad \alpha N_2 + \beta N_1) \tag{5}$$

Si k tend vers l'infini, X^k tend vers l'état stationnaire

$$X^\infty = \left(l', \quad l', \quad \cdots, \quad l', \quad \frac{N_1 + N_2}{2}, \quad l', \quad l', \quad \cdots, \quad l', \quad \frac{N_1 + N_2}{2}\right) \tag{6}$$

Il serait intéressant de savoir au bout de combien de temps on devient très proche de l'état stationnaire, auquel cas :

$$l' \le \alpha l = 2\alpha l' \le l' + 1 \Leftrightarrow l' \le l' + (1-2p)^k l' \le l' + 1 \Leftrightarrow 0 \le (1-2p)^k l' \le 1 .$$

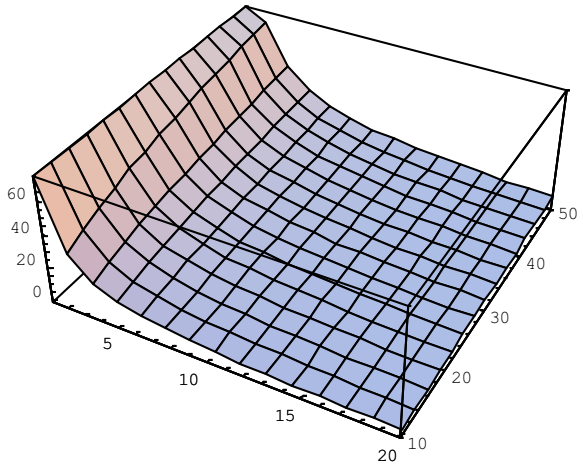
c.à.d.

$$(1-2p)^k \le \frac{1}{l'} \Leftrightarrow k \ln(1-2p) \le -\ln l' \Leftrightarrow k \ge \frac{-\ln l'}{\ln(1-2p)} .$$

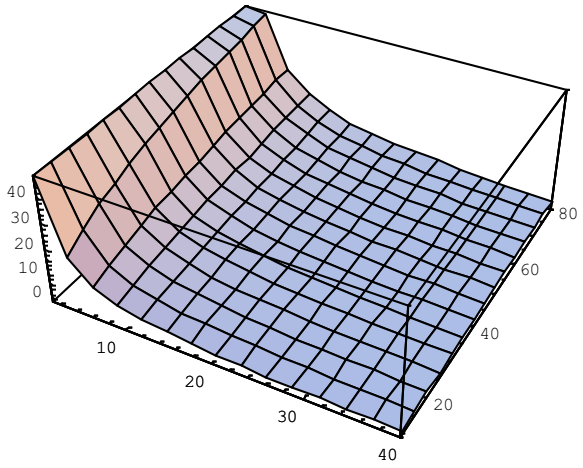
Considérons la fonction

$$k = \frac{-\ln l'}{\ln(1-2p)} = \frac{-\ln \frac{l}{2}}{\ln\left(1-\frac{c}{50}\right)} = \frac{\ln l - \ln 2}{\ln 50 - \ln(50-c)}$$

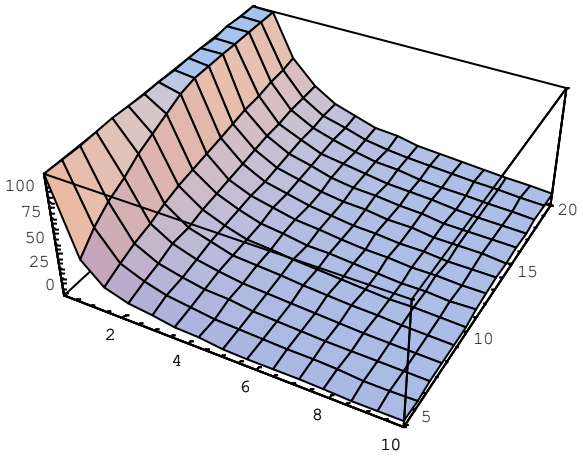
Le graphe de cette fonction permet de déterminer au bout de combien de temps on devient très proche de l'état stationnaire :



Temps nécessaire pour s'approcher de l'état stationnaire $1 \le c \le 20; \quad 10 \le l \le 50$



Temps nécessaire pour s'approcher de l'état stationnaire $1 \le c \le 40; \quad 10 \le l \le 80$



Temps nécessaire pour s'approcher de l'état stationnaire $0.1 \le c \le 10; \quad 4 \le l \le 20$

Si on suppose que $l > 2$ (on fait de chaque fragment au moins 3 répliques), et $c < 50$ (moins de la moitié des nœuds quittent ou réintègrent le système en une unité de temps), alors : $\frac{\delta k}{\delta l} > 0$ et $\frac{\delta k}{\delta c} < 0$; c.à.d. k croit lorsque l croit et décroit lorsque c croit.

Exemple : $m = 3$, $r = 5$, $l = 30$, $c = 0.5$, $N_1 = 1000$, $N_2 = 850$, $N = 2000$.

On devient très proche de l'état stationnaire après 270 unités de temps.

La probabilité de récupérer le fichier après 270 unités de temps est :

0.9967345319452382009000828046070 7. Ce résultat est très satisfaisant.

Par contre, si on prend $m = 3$, $r = 5$, $l = 30$, $c = 1.5$, $N_1 = 1000$, $N_2 = 850$, $N = 2000$. On devient très proche de l'état stationnaire après 89 unités de temps, et la probabilité de récupérer le fichier à l'état stationnaire est pratiquement la même dans les deux cas.

La seule différence est que, lorsque c croit, on s'approche de l'état stationnaire au bout d'une période de temps plus courte.

5.9 Performances dans les différentes approches

Dans la suite nous considérons un réseau de 2000 nœuds, avec un taux de mobilité c = 1%, (1% des nœuds quittent le système en une unité de temps). Ainsi $p = 0.01$. Nous considérons un fichier de 15 MB.

Nous lançons 40 répliques de ce fichier dans le réseau ; ce qui correspond à un volume de stockage égal à $15 \times 40 = 600$MB. Nous calculons ensuite la probabilité de récupérer le fichier après 100 unités de temps. Suivant la réplication classique utilisée par Gramoli :

$\alpha = \left\lceil \left(1-(1-0.01)^{100}\right)2000 \right\rceil = 1267.93$, à peu près 1268.

$n = 2000$, $q = 40$, $a = \max(0, \alpha - n + q) = 0$ et $b = \min(\alpha, q) = 40$.

La probabilité de récupérer le fichier après 100 unités de temps est :

$$\frac{\sum_{k=a}^{b}\left[\binom{n+k-q}{q}\binom{q}{k}\binom{n-q}{\alpha-k}\right]}{\binom{n}{q}\binom{n}{\alpha}} = 0.744563125363671808264001687613 67$$

Dans notre première approche, nous avons utilisé les codes correcteurs d'erreurs et nous avons intégré au travail de Gramoli la notion de découpage du fichier en blocs.

Nous découpons d'abord notre fichier en 5 blocs de 3 MB chacun, codés en 8 fragments de 3 MB chacun. Si nous lançons dans le réseau 25 répliques de chaque fragment, le volume de stockage est $8 \times 25 \times 3 = 600$ MB.

$\alpha = 1267.93$, à peu près 1268.

$N = 2000$, $n = 200$, $a = \max(0, \alpha + n - N) = 0$ et $b = \min(\alpha, n) = 200$.

La probabilité de récupérer le fichier après 100 unités de temps est :

$$\sum_{k=a}^{b}\sum_{p=m}^{n}\frac{\binom{n-k}{p}\binom{N-n+k}{n-p}\binom{n}{k}\binom{N-n}{\alpha-k}}{\binom{N}{n}\binom{N}{\alpha}} = \sum_{k=0}^{200}\sum_{p=5}^{200}\frac{\binom{200-k}{p}\binom{1800+k}{200-p}\binom{200}{k}\binom{1800}{1268-k}}{\binom{2000}{200}\binom{2000}{1268}}$$

=0.864482107656305361826938735639 45

Notre deuxième approche utilise les chaînes de Markov et le codage sans réplication. Nous découpons notre fichier en 5 blocs de 3 MB chacun, codés en 160 fragments de 3 MB chacun que nous lançons dans le réseau. Le volume de stockage est $160 \times 3 = 480$ MB.

$p = 0.01$, $X^0 = (n, \quad N_2, \quad 0, \quad N_4) = (160, 840, 0, 1000)$ avec $n + N_2 + N_4 = N = 2000$,

$P^k = \alpha I + \beta J$

où $\alpha = \frac{1}{2}\left[1+(1-2p)^{100}\right]$=0.5663 et $\beta = \frac{1}{2}\left[1-(1-2p)^{100}\right]$=0.4337,

$X^k = X^0 P^k = (a_1,\ a_2,\ a_3,\ a_4)$,

$X^{100} = (\alpha n, \alpha N_2 + \beta N_4, \beta n, \beta N_2 + \alpha N_4) = (90.6, 909.39, 69.39, 930.6)$

$s_1 = 90, s_2 = 909, n = 160$. La probabilité de récupérer le fichier après 100 unités de temps est :

$$\frac{\sum_{i=m}^{S_1}\binom{S_1}{i}\binom{S_2}{n-i}}{\binom{S_1+S_2}{n}} = \frac{\sum_{i=5}^{90}\binom{90}{i}\binom{909}{160-i}}{\binom{999}{160}} = 0.99957$$

Notre dernière approche utilise les chaînes de Markov et le codage avec réplication. Nous découpons notre fichier en 5 blocs de 3 MB chacun, codés en 8 fragments de 3 MB chacun. Nous faisons de chaque fragment 20 répliques que nous lançons dans le réseau. Le volume de stockage est $20 \times 8 \times 3 = 480$ MB.

$p = 0.01$, $X^0 = (n,\ N_2,\ 0,\ N_4) = (160, 840, 0, 1000)$ avec $n + N_2 + N_4 = N = 2000$,

Lorsque *k* tend vers l'infini, X^k tend vers $X = (80, 920, 80, 920)$ qui est l'état stationnaire.

La probabilité de perdre le fichier lorsque *k* tend vers l'infini est :

$$\pi = \frac{\binom{920}{160}}{\binom{1000}{160}} + \binom{8}{4}\sum_{a=1}^{10}\sum_{b=1}^{10}\sum_{c=1}^{10}\sum_{d=1}^{10}\frac{\binom{10}{a}\binom{10}{b}\binom{10}{c}\binom{10}{d}\binom{920}{160-a-b-c-d}}{\binom{1000}{160}} +$$

$$\binom{8}{3}\sum_{a=1}^{10}\sum_{b=1}^{10}\sum_{c=1}^{10}\frac{\binom{10}{a}\binom{10}{b}\binom{10}{c}\binom{920}{160-a-b-c}}{\binom{1000}{160}} + \binom{8}{2}\sum_{a=1}^{10}\sum_{b=1}^{10}\frac{\binom{10}{a}\binom{10}{b}\binom{920}{160-a-b}}{\binom{1000}{160}} +$$

$$\binom{8}{1}\sum_{a=1}^{10}\frac{\binom{10}{a}\binom{920}{160-a}}{\binom{1000}{160}}$$. La probabilité de récupérer le fichier lorsque *k* tend vers l'infini est :

1 - π = 0.967410049121619073677233451696 22

Il parait évident qu'en augmentant le nombre *l* des répliques de chaque fragment, on augmente la probabilité de récupérer le fichier et en même temps les frais de réplication. D'autre part, une petite valeur de *l*, tout en engendrant de petits frais pour la réplication, va diminuer considérablement la probabilité de récupérer le fichier au bout d'une période de temps assez courte. En conséquence, nous serons amenés au bout de cette période à récupérer le fichier pour le recoder et ensuite redistribuer de nouvelles répliques ; ce qui implique des frais bien plus importants. Il va de soi qu'un choix judicieux du nombre *l* des répliques de chaque fragment s'impose.

5.10 Conclusion

Dans ce chapitre, nous avons présenté une approche probabiliste qui utilise les techniques de redondance pour garantir la persistance des données dans un système de sauvegarde P2P. Notre approche est basée sur la chaîne de Markov avec laquelle nous avons présenté les différents états que peut avoir un nœud dans un réseau. Notre approche permet de faire différents type de calcul en modifiant les valeurs de paramètres tels que la taille du réseau, le taux de mouvement des nœuds, le temps d'exécution, le nombre des répliques, le facteur de redondance, …. A la fin de ce chapitre nous avons présenté un certains nombre de résultats obtenus par simulation.

Dans le chapitre suivant, nous allons présenter la mise en œuvre de nos travaux et de notre application de sauvegarde que nous avons implémentée au sein de l'entreprise.

Chapitre 6 Mise en œuvre

6.1 Introduction

Dans ce chapitre, nous allons décrire la mise en place de notre système de sauvegarde par l'implémentation d'une application de sauvegarde qui intègre des plugins. Nous décrivons l'architecture de notre application de sauvegarde ainsi que l'intégration, dans cette application, du système de planification. Dans un premier temps, nous allons présenter les outils nécessaires utilisés lors de la mise en œuvre du prototype. Ensuite, nous présentons des illustrations pour les différentes fonctionnalités.

6.2 Outils et langage d'implémentation de notre application

Pour l'implémentation de notre application de sauvegarde, nous avons utilisé le langage de programmation orienté objet C# (C Sharp) de Microsoft. Le C# qui est très proche de Java, fait partie de la plate-forme Microsoft.NET ; il a été crée pour que cette plate-forme soit dotée d'un langage permettant d'utiliser toutes ses capacités. La plate-forme Microsoft.NET permet le mixage des modules d'autres langages (C, C++, VB, CV.NET, …). L'entreprise a opté pour ce langage de programmation car la majorité des projets développés chez Alter Systems sont écrits avec du .NET (C#) et que la plupart des ingénieurs de l'entreprise ont une bonne expérience en C#.

Concernant l'environnement de développement, nous avons utilisé Microsoft Visual studio 2005 qui est une suite de développement pour Windows conçu par Microsoft. MS Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications bureautiques, des applications mobiles, des applications Web ainsi que des Services Web XML. Les applications développées en .NET ne sont pas compilées en langage machine. La compilation se fait dans un langage intermédiaire CIL (Common Intermediate Language). Une telle application est donc compilée, au moment de son exécution, dans le langage machine approprié à la plate-forme sur laquelle elle est exécutée. C'est ce qu'on appelle JIT (Just In Time). Cette séparation entre le programme binaire et la plateforme permet aux applications Microsoft de mieux supporter les différentes versions du système d'exploitation.

6.3 Architecture de notre application de sauvegarde

Notre application de sauvegarde est constituée d'une partie centrale appelée noyau et d'un certains nombre d'extensions qui s'ajoutent à ce noyau pour créer l'application. Dans cette partie nous allons présenter en détail la partie noyau ; nous allons ensuite voir les différents types d'extensions.

6.3.1 Le noyau

Le noyau de notre application de sauvegarde peut être représenté comme un petit programme qui prend en charge quelques tâches basiques et fait appel à d'autres composants (extensions) pour réaliser tout le reste. Les tâches du noyau sont les tâches essentielles de l'application telles que le déclenchement ou l'arrêt de la sauvegarde ou de la restauration, la sauvegarde d'un bloc de données, d'un fichier, d'un dossier …. Pour les autres tâches comme le découpage en blocs, la compression ou le cryptage des données, le noyau fait appel à d'autres extensions qui vont accomplir ces tâches. Cette technique permet de constituer plusieurs versions de l'application de sauvegarde d'une manière très flexible. Il suffit de grouper un certain nombre de composants ou d'extensions pour créer plusieurs versions du logiciel de sauvegarde différentes les unes des autres. Par exemple, si nous avons besoin d'un seul type de cryptage, il suffit de l'intégrer tout seul sans les autres extensions de cryptage.

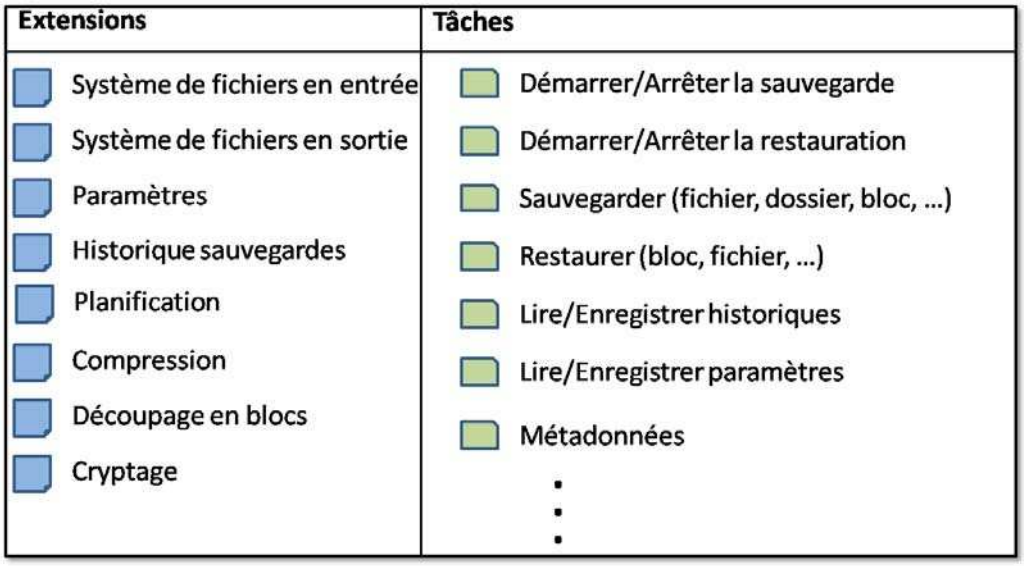


Figure 23 - Le noyau

La Figure 23 illustre une représentation graphique du noyau de notre application de sauvegarde. Dans cette figure, nous présentons les extensions et quelques tâches effectuées par ce programme.

6.3.2 Les extensions

Dans cette partie, nous allons présenter quelques extensions de notre application de sauvegarde ainsi que les composants de chaque extension.

6.3.2.1 Système de fichiers

Le système de fichiers permet de définir le type du système de sauvegarde. Plusieurs types existent :

- Local
- FTP
- HTTP
- P2P
- Outlook
- Thunderbird

Chacun de ces systèmes permet d'effectuer ou d'exécuter un certain nombre de tâches dans son propre environnement. Un système de fichiers local effectue les tâches comme la lecture et l'écriture sur un disque local alors qu'un système de fichiers FTP doit faire appel à un client FTP pour établir une connexion FTP à un serveur distant puis lire ou écrire sur ce serveur. Les méthodes de lecture, d'écriture ou de listage des fichiers et dossiers diffèrent d'un système à un autre. Le noyau décrit dans la partie précédente possède deux extensions de type système, un système en entrée et un autre en sortie. Ceci permet de faire plusieurs combinaisons de sauvegarde. Le système en entrée est le système à partir duquel la sauvegarde est effectuée et le système en sortie est le système sur lequel cette sauvegarde est envoyée. Prenons l'exemple d'un noyau qui a la configuration suivante :

- Système de fichiers en entrée : Local
- Système de fichiers en sortie : FTP

Cette configuration permet d'effectuer des sauvegardes du disque local vers un serveur FTP.

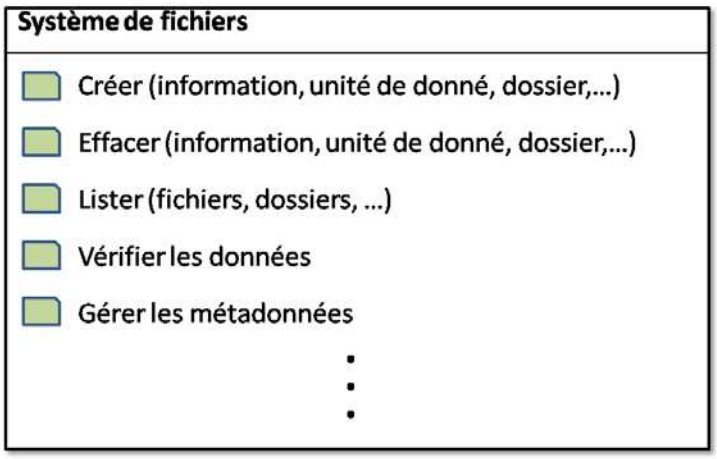


Figure 24 - Un système de fichiers

La figure 24 illustre un système de fichiers avec les différentes tâches ou méthodes qu'il prend en charge.

6.3.2.2 Paramètres

Ce composant ou extension permet de gérer toute la partie liée au paramétrage de l'application et des sauvegardes. Cette extension permet d'avoir plusieurs types de paramétrages allant d'un paramétrage local sous forme d'un fichier texte ou XML à une configuration distante stockée sur un serveur (FTP, Web Service, …). Ce composant permet d'effectuer plusieurs tâches telles que la création d'un nouveau paramétrage, la lecture et la mise à jour de paramètres existants. Il contient deux sous composants qui sont LocalParameters (pour gérer le paramétrage local) et DistantParameters (pour gérer le paramétrage distant). La Figure 25 montre une illustration de l'extension de paramètres.

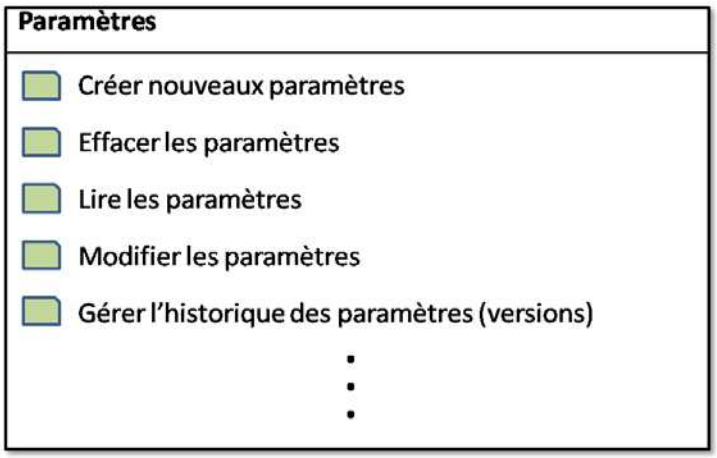


Figure 25 - Extension de paramétrage

6.3.2.3 Planification

Ce composant permet de gérer la partie planification. Comme nous l'avons déjà présenté dans la partie proposition, notre application de sauvegarde contient deux types de planning, un planning pour déclarer les créneaux de sauvegarde et un autre pour déclarer les créneaux des services. Le composant de planification prend en compte toutes les tâches liées à la partie planification (créer, mettre à jour ou supprimer un planning, gérer les clés d'un planning, …).

Le composant de planification est illustré dans la Figure 26 ci-dessous :

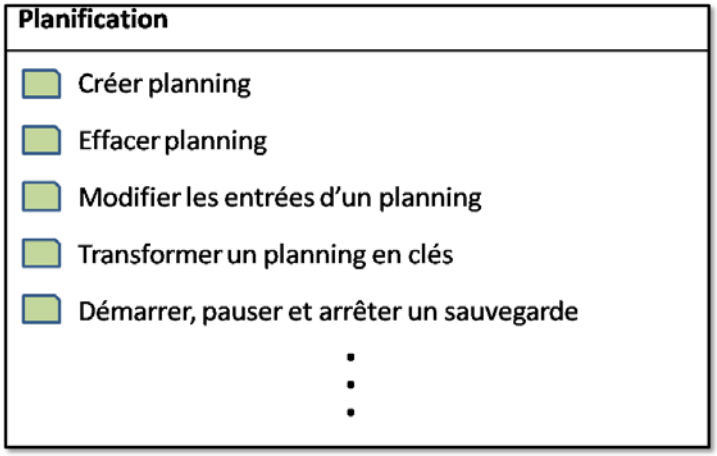


Figure 26 - Extension de paramétrage

Le composant de planification comporte plusieurs extensions pour gérer les différents types de plannings. On peut avoir des plannings partagés sur un réseau, des plannings sur des serveurs distants, ….

6.3.2.4 Découpage en blocs

Le composant de découpage en blocs contient plusieurs extensions comme l'illustre la figure 27 :

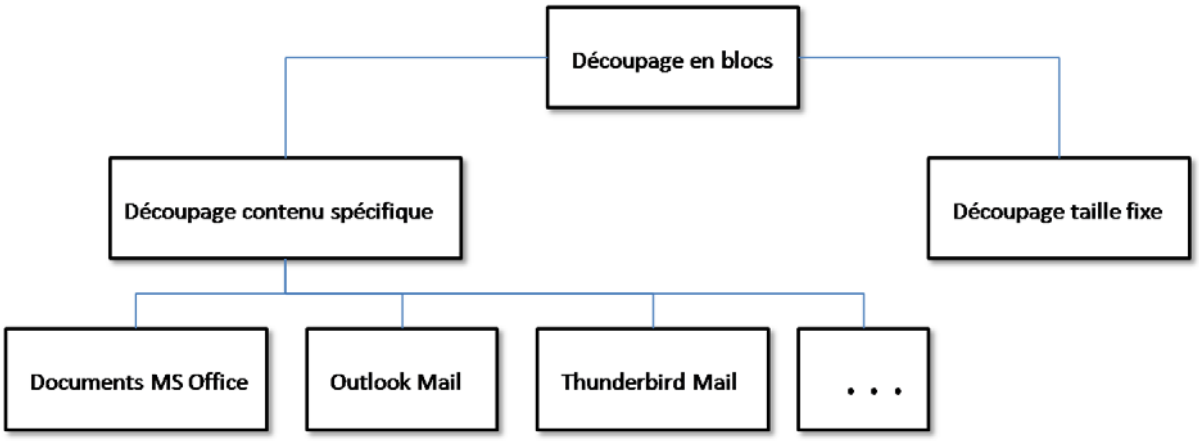


Figure 27 - Découpage en blocs

Le découpage en blocs de taille fixe ne prend pas en compte le contenu du fichier ou dossier à découper. La taille d'un bloc est un paramètre à configurer. Le but d'un découpage de contenu spécifique est de pouvoir, dans certains cas, séparer différents types de contenu. Par exemple Outlook et Thunderbird, qui sont deux clients de messagerie, stockent tout le contenu de la messagerie dans un seul fichier. Ce fichier contient les mails y compris les pièces jointes, les plannings des agendas, les messages envoyés, supprimés, les listes des contacts,…. Chacun des ces objets peut contenir ou peut être constitué de plusieurs autres objets. Par exemple, un mail est constitué de plusieurs objets qui sont : le sujet, les adresses (De, A, CC, BCC, Copie à, …), corps du mail, signatures, pièces jointes, …. Si on utilise le découpage classique pour ce genre d'application, on va se retrouver avec les objets dispersés dans plusieurs blocs de données, alors que le découpage de contenu spécifique permet de séparer ces objets et chaque objet garde bien ses propres sous-objets.

6.3.2.5 Les filtres

Le composant des filtres contient les filtres liés à la compression et au cryptage des données. La Figure 28 illustre l'arborescence d'un filtre avec quelques éléments de compression et de cryptage.

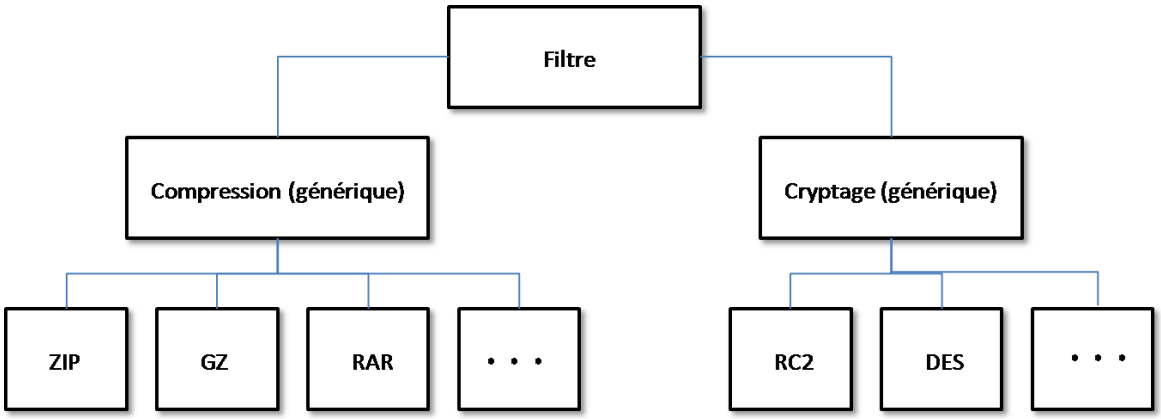


Figure 28 - Filtres (compression et cryptage)

Le noyau fait appel à ce composant pour la compression et le cryptage. Plusieurs types de compression et de cryptage existent. Le Filtre est de type extension ainsi que les différents types de compression et de cryptage qui sont aussi des extensions. Ce qui permet de rajouter ou de supprimer autant de types que l'on veut. Lors de la génération de l'application de sauvegarde, il est également possible de sélectionner les types de filtres qu'on souhaite intégrer à cette dernière ; ce qui permet dans certains cas de réduire sa taille.

6.4 Le configurateur de l'application de sauvegarde

Dans cette partie nous allons expliquer les différentes étapes pour configurer la sauvegarde des données. Cette configuration est réalisée à partir d'un logiciel qui fait partie de la solution de sauvegarde, c'est l'AlterBackup Configurator.

La Figure 29 illustre la première étape du configurateur qui consiste à saisir le nom de l'utilisateur et le mot de passe pour se connecter au serveur de configurations. Sur ce serveur distant, un web service installé gère tout ce qui est lié aux comptes utilisateurs et à leurs configurations de sauvegarde. L'intérêt de stocker ces informations sur un serveur distant est de garantir que ces informations ne se perdent pas en cas de problèmes ou d'incidents sur la machine de l'utilisateur.

Une fois l'utilisateur connecté au serveur, il devra choisir l'identifiant de la machine physique pour laquelle il souhaite saisir ou modifier la configuration. Pour utiliser la solution de sauvegarde Alter Backup, l'utilisateur doit remplir un contrat dans lequel il choisit le nombre de machines à intégrer pour la sauvegarde. Comme l'illustre la Figure 29, une fois la

machine choisie, le bouton « Suivant » s'active pour permettre à l'utilisateur de passer aux étapes de configuration de la machine sélectionnée.

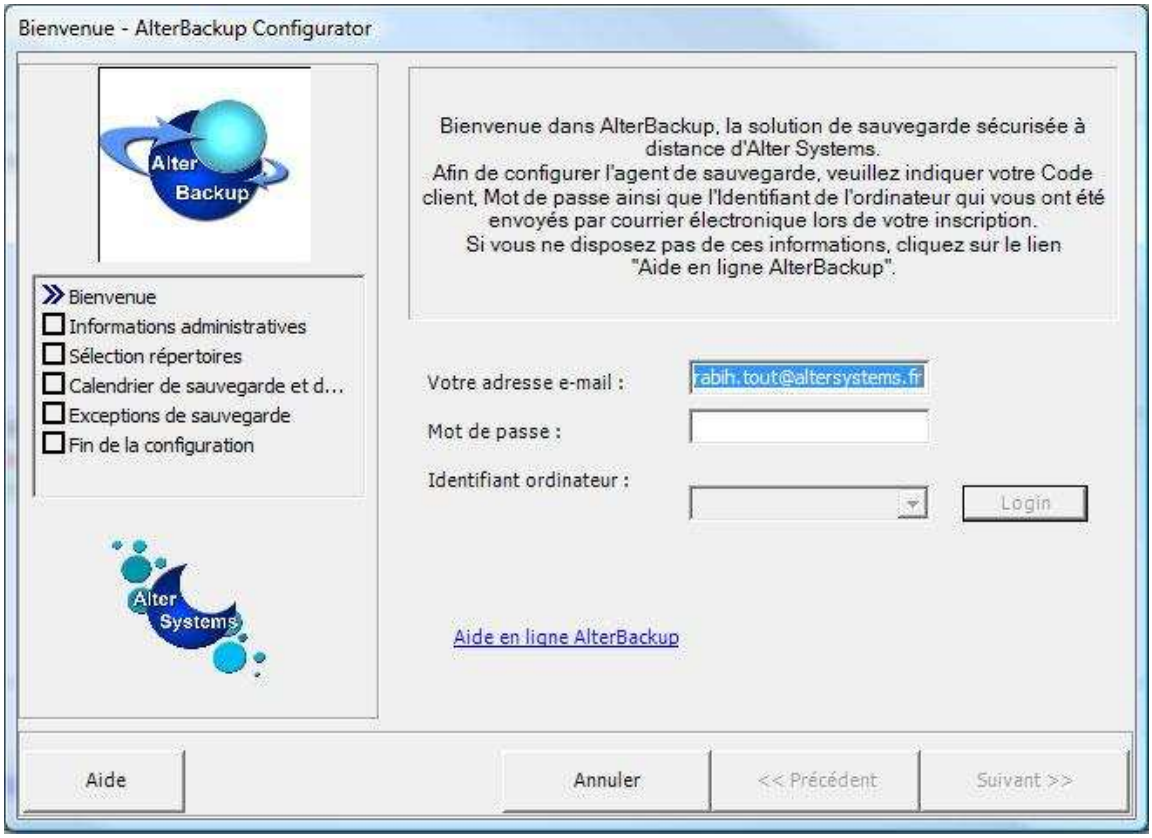


Figure 29 - Configurateur (1ère étape) – Saisie des informations de connexion

Une fois la machine choisie, l'utilisateur passe à la deuxième étape (Figure 30). Sur cet écran, s'affichent les informations administratives liées à l'utilisateur ainsi qu'à son contrat de sauvegarde. On peut bien voir le numéro de contrat, sa date de début et de fin, ainsi que toutes les machines physiques inscrites à ce contrat.

Le but de l'affichage de cet écran au départ est de rappeler à l'utilisateur des détails de son contrat à chaque fois qu'il veut modifier sa configuration. Cela permet de lui rappeler surtout la date de fin de son contrat. Il peut également signaler par mail à l'administrateur toute erreur dans les informations sur son contrat ou toute volonté de modification des données personnelles.

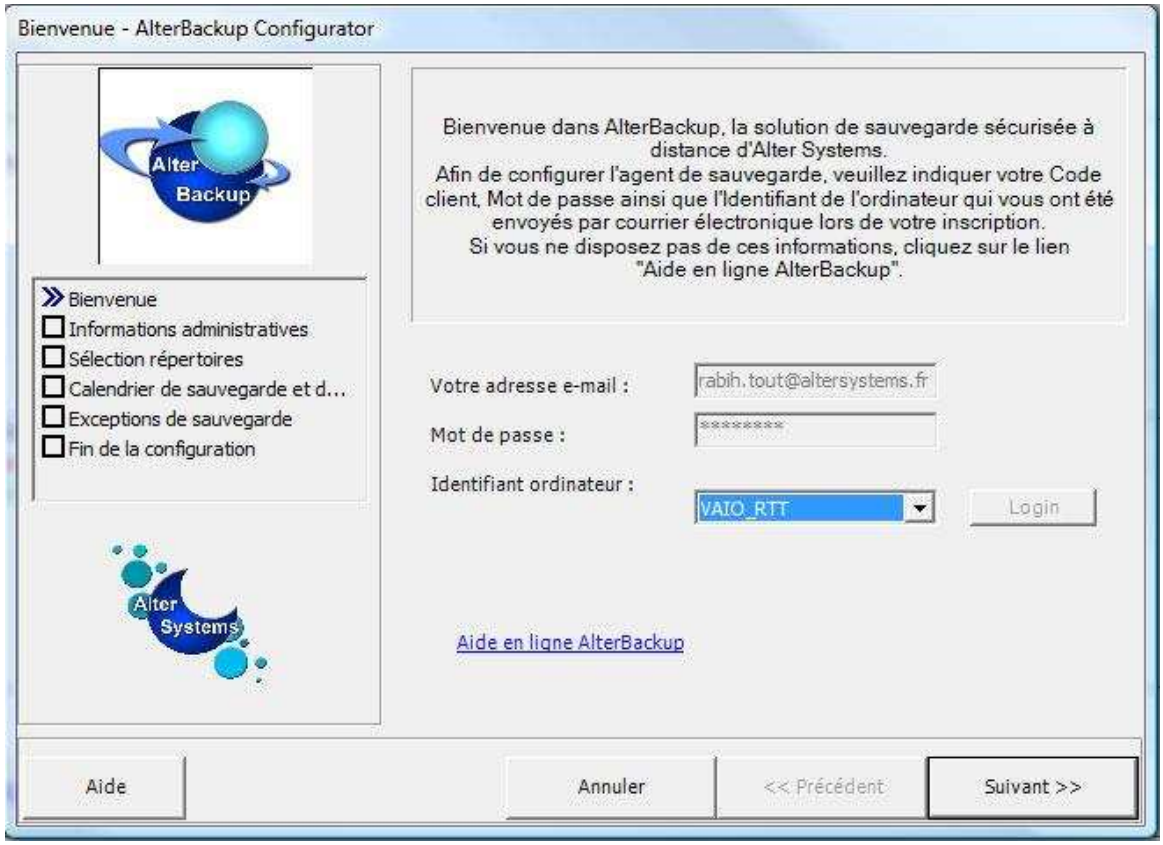


Figure 30 - Configurateur (1ère étape) – Sélection de la machine physique

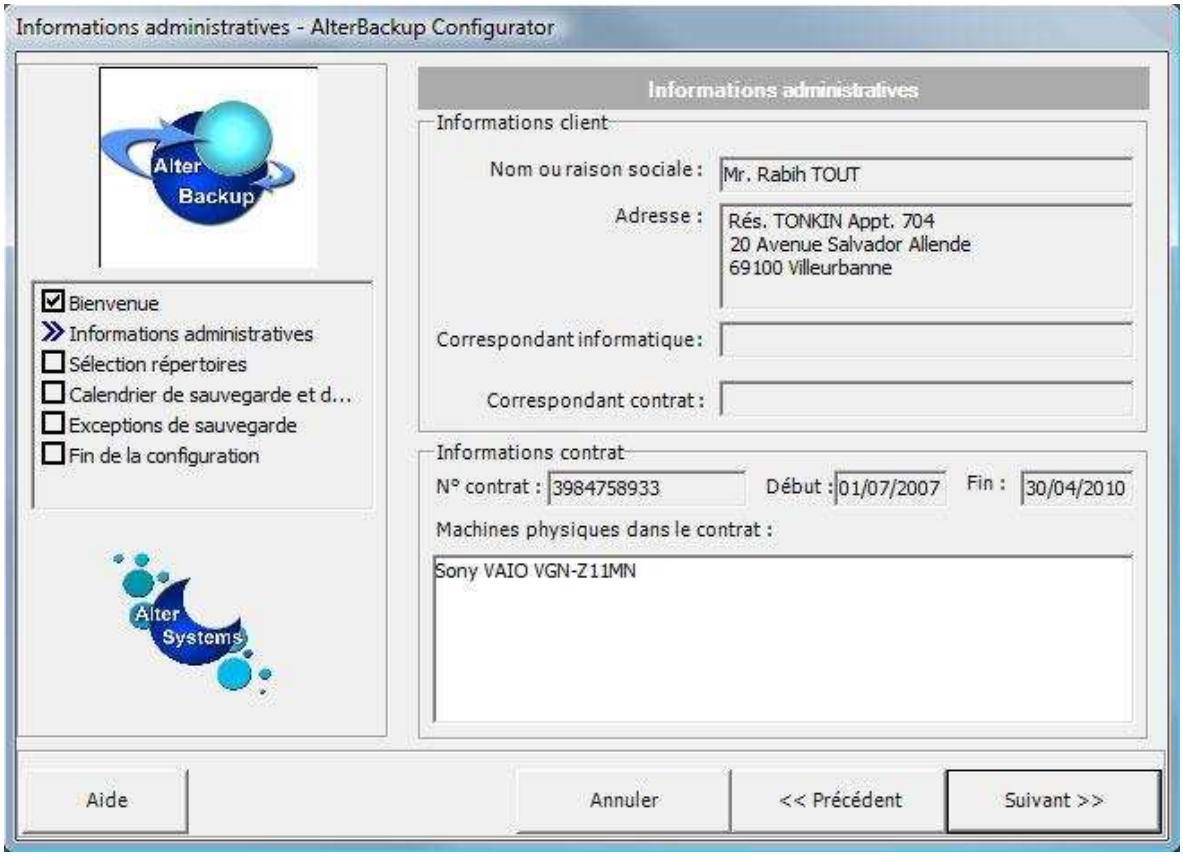


Figure 31 - Configurateur (2ème étape) – Informations administratives

La 3ème étape du configurateur (Figure 31) est la sélection des répertoires à sauvegarder. Le programme affiche à l'utilisateur tous les disques existants sur sa machine physique. Ce dernier peut parcourir les répertoires et sous-répertoires d'un disque sélectionné pour choisir les dossiers et fichiers à sauvegarder. La sélection est récursive (le choix d'un répertoire implique la sélection automatique de tous ses sous-répertoires). À chaque fois que l'utilisateur sélectionne un répertoire, le champ qui affiche la taille des données se met automatiquement à jour. Le but de ce champ est de permettre à l'utilisateur d'avoir une idée de la taille approximative des données qu'il a choisies pour sauvegarder. Si jamais la taille de ces données dépasse la taille maximale choisie par l'utilisateur dans son contrat de sauvegarde, la case se met en couleur rouge pour avertir l'utilisateur qu'il a dépassé la taille maximale autorisée et qu'il ne pourra donc pas passer à l'étape suivante tant qu'il n'aura pas modifié sa sélection.

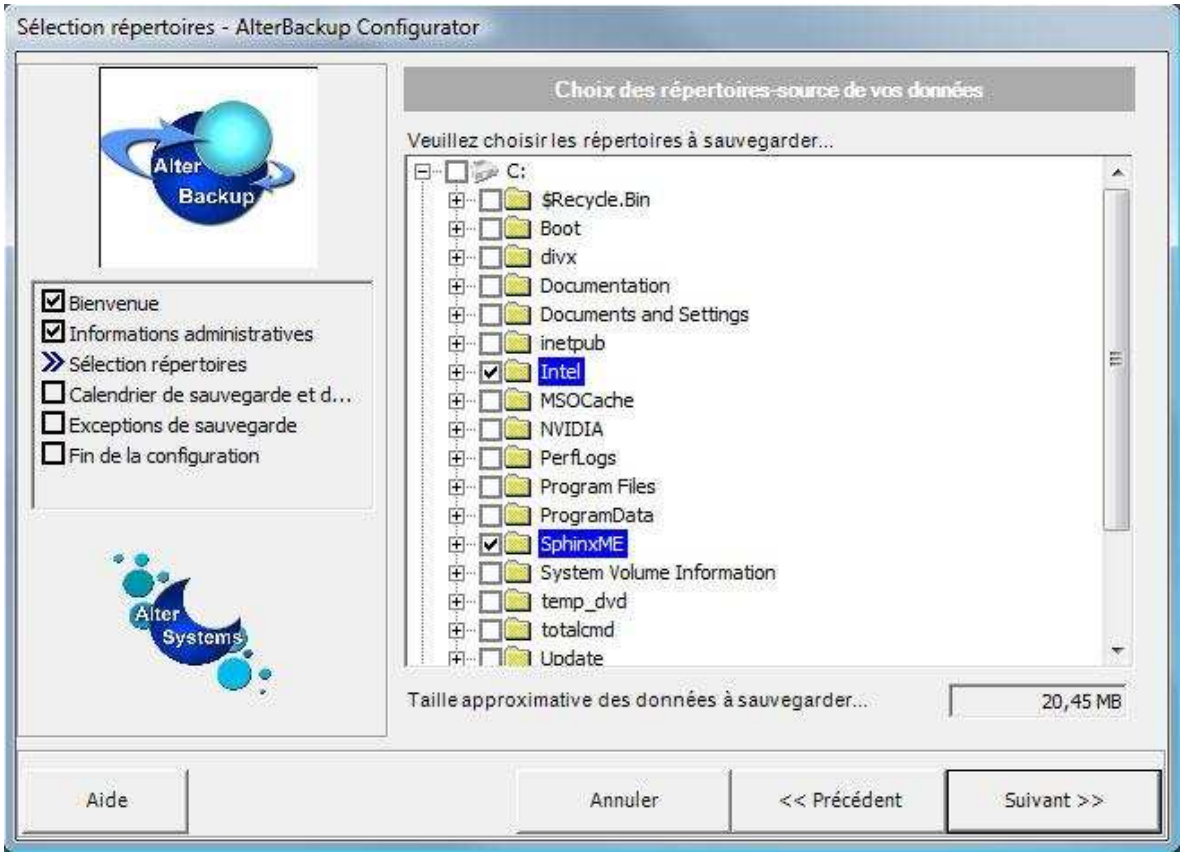


Figure 32 - Configurateur (3ème étape) – Sélection des répertoires

Le calcul de la taille des données se fait avec un algorithme récursif qui parcourt tous les sous-dossiers d'un répertoire sélectionné pour calculer la taille totale de tous les dossiers et fichiers existants.

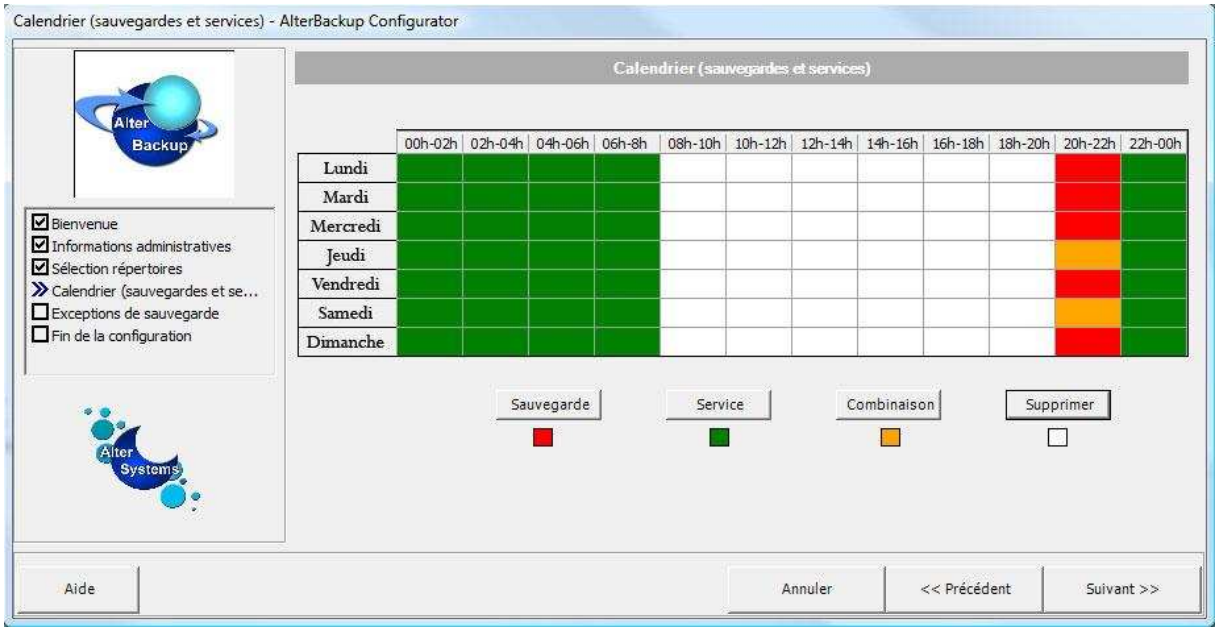


Figure 33 - Configurateur (4ème étape) – Saisie des plannings

L'étape 4 du configurateur illustrée dans la Figure 33, permet de saisir et de modifier les plannings de sauvegarde et de service. L'utilisateur coche les cases qu'il souhaite modifier et clique sur l'une des quatre touches situées en dessous du planning :

- La touche « Sauvegarde » change la couleur des cellules en rouge. Les créneaux en rouge sont les créneaux durant lesquels l'utilisateur souhaite sauvegarder ses propres données.
- La touche « Service » permet de sélectionner les créneaux durant lesquels l'utilisateur est prêt à présenter des services aux autres utilisateurs du réseau.
- La touche « Combinaison » permet de saisir les créneaux durant lesquels un utilisateur désire effectuer ses propres sauvegardes et est prêt en même temps à présenter des services. Ceci permet de saisir les créneaux sur un même planning.
- Finalement la touche « Supprimer » permet de supprimer un ou plusieurs créneaux sélectionnés.

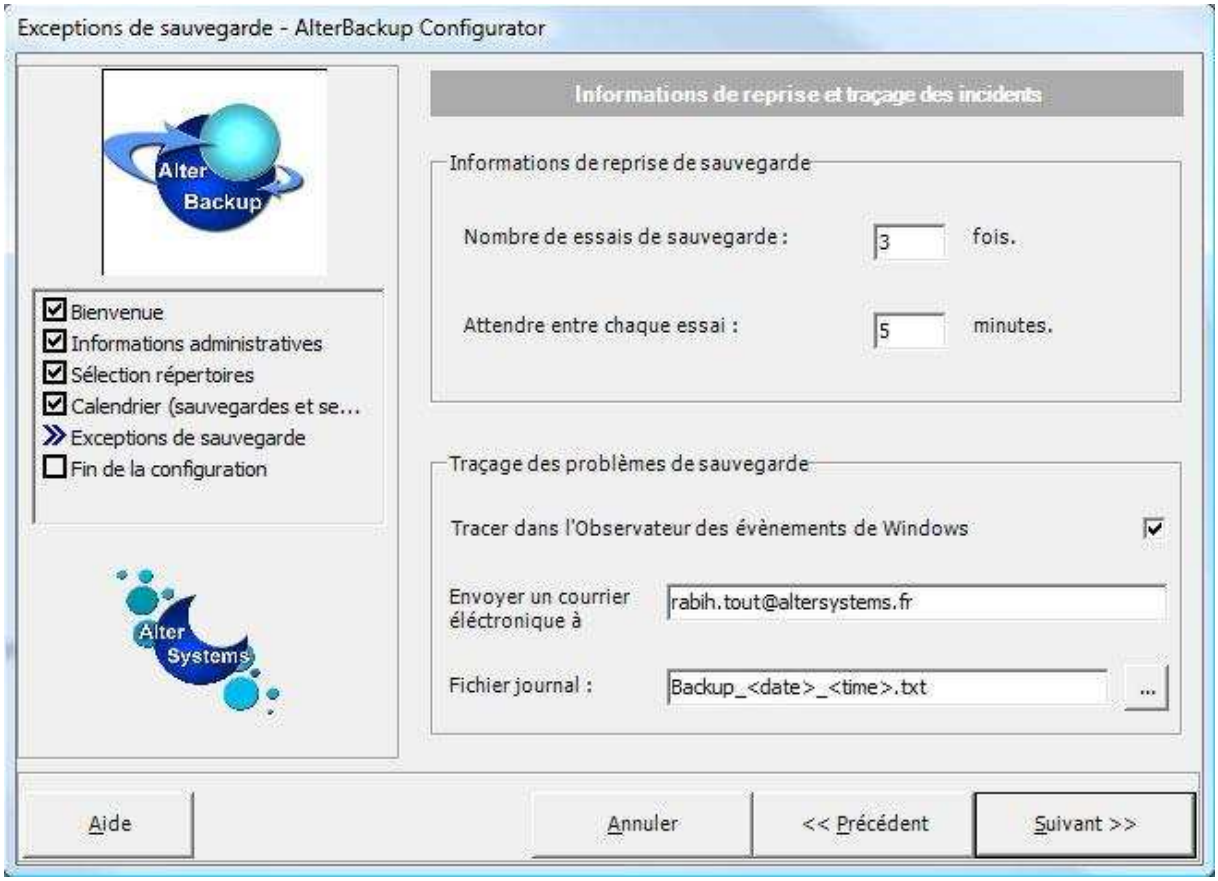


Figure 34 - Configurateur (5ème étape) – Saisie des exceptions de sauvegarde

La 5ème étape de la configuration (Figure 34) permet de saisir les exceptions de la sauvegarde. Le nombre des essais de sauvegarde est très important dans certains cas. Par exemple si on essaie de faire une sauvegarde vers un serveur FTP et que ce serveur est momentanément déconnecté ou indisponible, le fait d'avoir plusieurs essais, augmente la chance de se connecter au serveur et d'effectuer la sauvegarde.

L'utilisateur a la possibilité de tracer les événements de la sauvegarde dans l'observateur des événements de Windows, il a aussi la possibilité de recevoir un mail détaillé de ces événements. Il est également possible de générer un fichier journal contenant ces événements.

La dernière étape (Figure 35) permet d'enregistrer la configuration sur le serveur et d'envoyer un e-mail à l'utilisateur pour l'informer de sa nouvelle configuration.

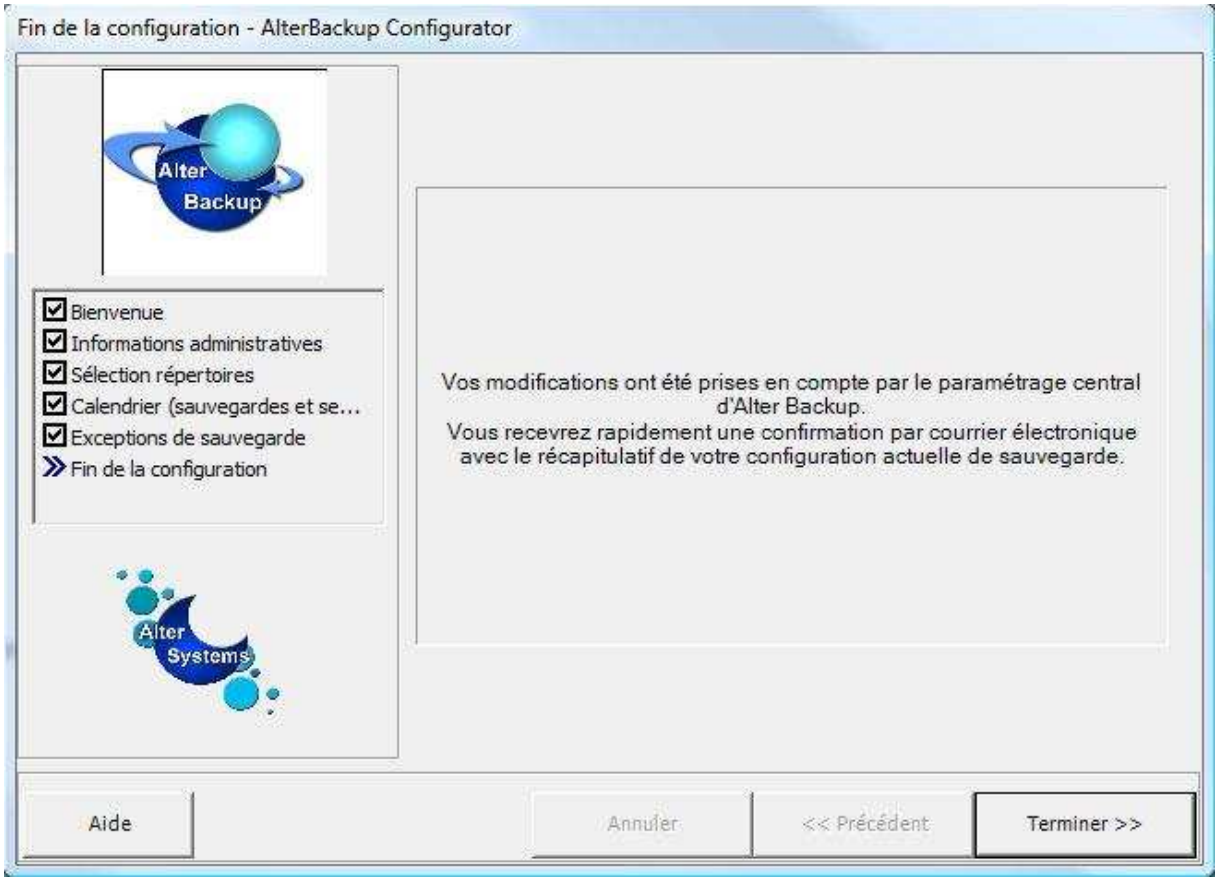


Figure 35 - Configurateur (6ème étape) – Fin de la configuration

Conclusion et Perspectives

Le nombre des personnes qui utilisent l'informatique augmente de plus en plus. L'usage du PC et d'internet devient indispensable et toute personne utilisant les outils informatiques, a besoin de stocker un volume de données de plus en plus important. Un simple accident pourra effacer toutes les données sans grand espoir de les retrouver. La meilleure solution pour éviter la perte des données est de faire une sauvegarde régulière. Les techniques de sauvegarde classiques présentent des limitations. La sauvegarde de données sur un serveur distant est payante pour les utilisateurs car elle exige des frais élevés pour le fournisseur comme le coût d'administration et du matériel dédié.

Le grand succès des réseaux P2P dans les applications de partages de fichiers a permis à différents type d'applications de se lancer dans ce type de réseau. Ces réseaux décentralisés et distribués n'exigent pas d'administration centralisé ni de matériels dédiés. Ces avantages ont attiré les applications de sauvegarde pour se lancer dans des applications basées sur le P2P. Le but étant d'exploiter l'espace disque non utilisé par l'utilisateur pour stocker les données sauvegardées. Cela paraît simple mais en utilisant les réseaux P2P, nous pourrons avoir plusieurs problèmes liés à ces réseaux comme la sécurité, le routage, l'organisation des sauvegardes, la synchronisation des données, la persistance des données, …

Pour sauvegarder les données dans ce genre de réseaux, il faut garantir que ces données vont persister pour que nous puissions nous en servir au moment d'une restauration. Dans cette thèse, nous nous sommes intéressés à la sauvegarde des données dans les réseaux P2P et surtout à deux problématiques qui sont l'organisation du réseau de sauvegarde et la persistance des données sauvegardées. Dans ce cadre, nous avons focalisé notre étude, dans le 1er chapitre, sur la sauvegarde des données en présentant les différentes stratégies de sauvegarde et les différents types de supports utilisés. Nous avons expliqué les différences entre les différentes architectures de réseaux en présentant les avantages et les inconvénients de chacune d'elles et en détaillant le fonctionnement de l'architecture P2P.

Dans le 2ème chapitre, nous avons présenté une étude bibliographique sur quelques systèmes de sauvegarde P2P. Nous avons commencé ce chapitre avec une présentation des applications de partage de fichier pour montrer le passage de ces applications vers le besoin d'applications de sauvegarde basées sur les réseaux P2P. Dans le chapitre 3, nous présentons une étude bibliographique sur les mécanismes de redondance.

Face à un certain nombre de problématiques que nous avons rencontré lors de la création de notre application de sauvegarde P2P, nous avons décidé de traiter les deux problématiques cités en dessus. Dans le 4ème chapitre, nous avons proposé un système de planification de sauvegarde P2P qui consiste à définir pour chaque nœud participant au système, un certain nombre de plannings qui permettent d'informer les autres nœuds du réseau de sa propre situation en consultant la DHT. Cette méthode nous a permis de mieux structurer notre réseau de sauvegarde et de diminuer le nombre de requêtes inutiles dans le réseau.

Dans le chapitre 5, nous proposons une approche probabiliste qui permet de calculer le nombre de répliques nécessaires pour garantir la persistance des données sauvegardées dans le réseau. Connaissant les propriétés du réseau comme le nombre des nœuds, le taux de mouvement, nous arrivons à calculer le nombre minimum de répliques à diffuser dans le réseau de manière à pouvoir récupérer au moins une réplique après un certain temps *t*. Nous simulons les états de chaque nœud en utilisant les chaînes de Markov. Dans le dernier chapitre, nous présentons l'architecture de notre application de sauvegarde P2P développée au sein de la société. Elle intègre des plugins et permet de basculer facilement entre plusieurs types de systèmes de fichiers.

Perspectives

Le travail réalisé dans cette thèse a été effectué dans un cadre précis où plusieurs extensions et améliorations peuvent être envisagées comme suit :

- ***Extension du modèle basé sur la chaîne de Marcov*** : Dans notre travail, nous avons considéré que les nœuds avec ou sans données peuvent basculer entre deux états (connectés ou déconnectés). Pour se rapprocher plus de la réalité, il est également nécessaire qu'un nœud puisse basculer entre les états (avec et sans données). Un nœud peut perdre ses propres donnés à n'importe quel moment (connecté ou déconnecté) et inversement, un nœud doit pourvoir acquérir des donnés à tout moment.
- ***La sécurité des sauvegardes*** : Dans le cadre de cette thèse, nous nous sommes intéressés à deux problématiques qui sont l'organisation du réseau de sauvegarde et la persistance des données. La sécurité, qui est aussi un sujet très important, n'a pas été étudiée. La sécurité des sauvegardes dans les réseaux P2P sera un sujet intéressant à évoquer et à étudier en détail.

- ***La pérennité des donnés*** : Même si la persistance des donnés est un sujet important, la pérennité des donnés pourra être aussi intéressante. Aujourd'hui, l'intérêt de notre approche est de garantir la persistance des donnés dans le système. La pérennité n'a pas été étudiée donc notre système ne garantit pas que ces donnés vont être disponible pour l'utilisateur à tout moment. Ce sujet est très important car il permet d'améliorer la qualité de service de notre application de sauvegarde.
- ***Application de sauvegarde mondiale*** : Nous avons développé une application de sauvegarde qui implémente les approches que nous avons proposées. Actuellement, notre application de sauvegarde s'adresse aux entreprises car c'était notre priorité à cause des demandes reçues de nos clients. Il sera très intéressant d'élargir cette application pour qu'elle s'adresse à tous les utilisateurs comme les applications de téléchargement de fichiers multimédias qu'on trouve actuellement. Cela permettra de créer une application de sauvegarde gratuite à large échelle.

Bibliographie

[1] Y. Zhang, Z. Li, Z. Hu, H. Tu, and H. Lin, "A P2P E-commerce Related Network Security Issue: P2P Worm," *ISECS '08: Proceedings of the 2008 International Symposium on Electronic Commerce and Security*, pp. 114-117, 2008.

[2] W. Yu, S. Chellappan, X. Wang, and D. Xuan, "Peer-to-peer system-based active worm attacks: Modeling, analysis and defense," *Comput. Commun.*, vol. 31, no. 17, pp. 4005-4017, Nov. 2008.

[3] J. Winter, "Routing of structured queries in large-scale distributed systems," *LSDS-IR '08: Proceeding of the 2008 ACM workshop on Large-Scale distributed systems for information retrieval*, pp. 11-18, Oct. 2008.

[4] K. Needels and M. Kwon, "Secure routing in peer-to-peer distributed hash tables," *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, 2009.

[5] C. Lu, Z. Li, H. Lin, and Q. Huang, "Fault-Tolerant Routing for P2P System with Structured Topology," *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, vol. 3, pp. 102-105, Dec. 2008.

[6] F. Atalla, D. Miranda, J. Almeida, M. A. Gon\c{c}alves, and V. Almeida, "Analyzing the impact of churn and malicious behavior on the quality of peer-to-peer web search," *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pp. 1137-1144, Mar. 2008.

[7] O. D. Sahin, "Supporting complex queries over structured p2p networks," *Doctoral Thesis*, 2005.

[8] V. Vishnumurthy and P. Francis, "A comparison of structured and unstructured P2P approaches to heterogeneous random peer selection," *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, pp. 1-14, 2007.

[9] R. Tout, N. Lumineau, P. Ghodous, and M. Tanasoiu, "Schedule-based P2P Network Organization," in *International Conference on Digital Information Management (ICDIM'07)*, Lyon, 2007.

[10] R. Tout, P. Ghodous, A. Ouksel, and M. Tanasoiu, "Backup Scheduling in Clustered P2P Network," in *16th ISPE International Conference on Concurrent Engineering (CE2009)*, Taipei, Taiwan, 2009.

[11] Napster. [Online]. http://www.napster.com

[12] O. Peres and T. Herault, "Table de hachage distribuée autostabilisante," in *9ème Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Ile d'Oléron, 2007, pp. 63-66.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001*

conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, California, United States, 2001, pp. 149-160.

[14] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, 2001, pp. 329-350.

[15] eMule. [Online]. http://www.emule.com

[16] Gnutella. [Online]. http://www.gnutella.com

[17] Kazaa. [Online]. http://www.kazaa.com

[18] Freenet. [Online]. http://freenetproject.org

[19] BitTorrent. [Online]. http://www.bittorent.com

[20] K. D. John, et al., "OceanStore: An Architecture for Global-Scale Persistent Storage," in *Proceedings of the Ninth international Conference on Architectural Support For Programming Languages and Operating Systems*, Cambridge, Massachusetts, 2000, pp. 190-201.

[21] P. Drusche and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility," in *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, Washington, 2001, pp. 75-80.

[22] C. Batten, K. Barr, A. Saraf, and S. Trepetin, *pStore: A Secure Peer-to-Peer Backup System*. 2001.

[23] Rsync. [Online]. http://www.samba.org/rsync

[24] M. Landers, H. Zhang, and K.-L. Tan, "PeerStore: Better Performance by Relaxing in Peer-to-Peer Backup," *P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, pp. 72-79, 2004.

[25] L. P. Cox, O. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making Backup Cheap and Easy," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[26] L. P. C. a. B. D. Noble, "Samsara: Honor Among Thieves in Peer-to-Peer Storage," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Bolton Landing, NY, USA, Oct. 2003, pp. 120-132.

[27] P. K. Gummadi, S. Saroiu, and S. D. Gribble, "A measurement study of Napster and Gnutella as examples of peer-to-peer file sharing systems," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 1, pp. 82-82, Jan. 2002.

[28] V. Gramoli, A.-M. Kermarrec, A. Mostéfaoui, M. Raynal, and B. Sericola, "Core Persistence in Peer-to-Peer Systems: Relating Size to Lifetime," in *Proceedings of the On The Move*

International Workshop on Reliability in Decentralized Distributed systems, 2006, pp. 1470-1479.

[29] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs," *SIGMETRICS Perform. Eval. Rev.*, vol. 28, no. 1, pp. 34-43, 2000.

[30] H. Weatherspoon and J. Kubiatowicz, "Erasure Coding Vs. Replication: A Quantitative Comparison," *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pp. 328-338, 2002.

[31] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pp. 109-116, 1988.

[32] H. F. Mattson and G. Solomon, "A new treatment of Bose-Chaudhuri codes. Journal of the Society of Industrial and Applied Mathematics," pp. 654-669, 1961.

[33] M. Luby, et al., "The Use of Forward Error Correction (FEC) in Reliable Multicast," 2002.

[34] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proc. of the 29th ACM Symposium on Theory of Computing*, 1997.

[35] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 56-67, 1998.

[36] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: system support for automated availability management," *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pp. 25-25, 2004.

[37] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in *Proceedings of IPTPS*, 2005.

[38] G. Chen, T. Qiu, and F. Wu, "Insight into redundancy schemes in DHTs," *J. Supercomput.*, vol. 43, no. 2, pp. 183-198, 2008.

[39] H. LI and G. Chen, "Data Persistence in Structured P2P Networks with Redundancy Schemes," *GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing*, pp. 542-549, 2007.

[40] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, Banff, Alberta, Canada, 2001, pp. 188-201.

[41] S. Ghamri-Doudane and N. Agoulmine, "Enhanced DHT-based P2P Architecture for Effective Resource Discovery and Management," *J. Netw. Syst. Manage*, vol. 15, no. 3, pp. 335-354, Sep. 2007.