Réduction de l'impact des fautes corrélées dans les réseaux pair-à-pair en utilisant des nuées d'agents mobiles

Benoît Romito, François Bourdon

▶ **To cite this version:**

Benoît Romito, François Bourdon. Réduction de l'impact des fautes corrélées dans les réseaux pair-à-pair en utilisant des nuées d'agents mobiles. 20ème Journées Francophones sur les Systemes Multi-Agents (JFSMA'12), 2012, Honfleur, France. hal-00973549

HAL Id: hal-00973549

https://hal.science/hal-00973549

Submitted on 4 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Réduction de l'impact des fautes corrélées dans les réseaux pair-à-pair en utilisant des nuées d'agents mobiles

B. Romito[a] F. Bourdon[a]
benoit.romito@unicaen.fr francois.bourdon@unicaen.fr

[a]Laboratoire GREYC, ENSICAEN, CNRS UMR 6072
Université de Caen Basse-Normandie, France

Résumé

Cet article introduit MINCOR*, un algorithme de recuit simulé décentralisé pour le placement de données dans les réseaux pair-à-pair. Il a pour objectif de réduire l'impact des fautes corrélées sur ces systèmes de stockages. Le placement est réalisé à l'aide d'un système multi-agents transformant les documents en nuées d'agents mobiles. Après une phase de clustering du réseau regroupant les pairs fortement corrélés entre eux, les nuées exécutant* MINCOR *arrivent à trouver un placement qui minimise le nombre d'agents sur les mêmes clusters. Ce placement est trouvé de manière décentralisée grâce à la capacité d'exploration de l'environnement des nuées. Un ensemble d'expériences ont été réalisées sur ce système lorsqu'il est soumis à des fautes corrélées. Ces expériences montrent que, en pratique, le placement attendu est effectivement obtenu. Elles montrent également que les nuées exécutant* MINCOR *subissent moins de fautes concurrentes en présence de fautes corrélées qu'un simple placement aléatoire.*

Mots-clés : *fautes corrélées, flocking, agents mobiles, stockage décentralisé, réseaux pair-à-pair, recuit simulé distribué*

Abstract

This paper introduces MINCOR*, a decentralized simulated annealing algorithm designed for the data placement in peer-to-peer networks. Its goal is to reduce the correlated failures impact in such data storage systems. This data placement is realized using a multi-agent system which turns the documents into mobile agents flocks. After a network clustering step where highly correlated peers are regrouped together, the flocks executing* MINCOR *are able to find a placement minimizing the number of agents on the same clusters. This placement is obtained in a decentralized way thanks to the environment exploration capabilities of the flocks. A set of experiments are performed on this system in presence of correlated failures. They show that, in practice, the expected placement is well obtained. They also show that, flocks using the* MINCOR *algorithm suffer less concurrent losses in presence of correlated failures than a mere random placement.*

Keywords: *correlated failures, flocking, mobile agents, decentralized data storage, peer-to-peer networks, distributed simulated annealing*

1 Introduction

Le stockage robuste de l'information dans les réseaux pair-à-pair repose sur la mise en place de schémas de l'information efficaces. En effet, la caractéristique principale de ce type d'architectures ouvertes est que les pairs peuvent se déconnecter à tout moment en privant le reste du réseau des données qu'ils hébergent. D'une manière générale, la tolérance aux fautes est obtenue par l'application de trois mécanismes :

1. introduire de la redondance dans les données [23] pour tolérer les fautes ponctuelles. Cette redondance peut être obtenue par réplication ou par des codes d'effacement [15] ;
2. assurer un placement adéquat de cette redondance pour en faciliter la supervision et la recherche [9] ;
3. réparer la donnée lorsque trop de répliques ont été perdues [6]. Les réparations sont indispensables pour assurer la durabilité du système.

De fait, chaque système défini son propre schéma de l'information et les performances de ce schéma en terme de tolérance aux fautes et de disponibilité sont évaluées sur des modèles de fautes. Cependant, la plupart des modèles de fautes employés comme c'est le cas dans CFS [4] ou dans Farsite [1] font l'hypothèse que les fautes sont indépendantes. C'est à dire que les pairs se déconnectent indépendamment les uns des autres. Hors, en réalité, certains pairs

partagent des traits communs les faisant covarier [24]. Par exemple des pairs peuvent être situés dans le même bâtiment, ils peuvent partager les mêmes routeurs ou posséder les même vulnérabilités. Héberger plusieurs répliques d'un même document sur des pairs corrélés expose le document à la perte simultanée de ces répliques alors que son modèle de fautes ne l'envisage pas. La disponibilité espérée s'en trouve alors faussée pouvant conduire dans le pire des cas à la perte de la donnée. Nous proposons dans cet article un placement des répliques tenant compte des corrélations entre les pairs. Cette approche tire avantage de la mobilité d'un système multi-agents pour le stockage pair-à-pair que nous avons proposé dans [16, 17]. Dans cette architecture, les documents sont fragmentés et transformés en nuées d'agents mobiles qui se déplacent dans le réseau. Après une étape de clustering qui regroupe les pairs fortement corrélés entre eux [24], nous proposons un algorithme de recuit simulé distribué (MINCOR) qui optimise le placement des nuées sur ces clusters. Cet algorithme est en fait un filtrage des candidats dans le déplacement en flocking qui va guider les déplacement de chaque nuée afin qu'un maximum de ses fragments soient répartis sur des clusters distincts. Cette répartition a pour but de minimiser le nombre de pertes d'agents simultanées lorsque les fautes sont corrélées. Nous avons conduit un ensemble d'expérimentations par la simulation afin d'évaluer cet algorithme. La première évaluation que nous proposons mesure la qualité du placement d'une nuée sur l'ensemble des clusters. Dans un second temps, nous évaluons le gain à utiliser cet algorithme en comparaison à un simple placement aléatoire de nuée. Ce gain est mesuré en comptant le nombre moyen de fautes simultanées subies par chacune des nuées exécutant les différents algorithmes de placement.

La Sec 2 est consacrée aux travaux connexes sur les fautes corrélées dans les réseaux pair-à-pair et sur l'architecture de flocking. Puis, à la Sec 3, nous présentons notre algorithme de placement sur les clusters de corrélations. Enfin nous évaluons notre approche par des simulations à la Sec 5.

2 Travaux connexes

2.1 Fautes corrélées

Une grande quantité de plateformes de stockage de documents en pair-à-pair ont vu le jour dont [4, 18, 22, 1, 20, 25]. Toutes ces plateformes ont été étudiées dans le détail à l'aide de modèles de comportement en cas de fautes. Mais ces modèles de comportement font l'hypothèse (implicite ou non) que les fautes apparaissent de manière indépendantes. Pourtant, les pannes sont bel et bien corrélées [2, 10, 24, 12, 14, 5]. Partant de ce constat, deux grande idées sont proposées dans la littérature pour prendre en compte ce problème des corrélations :

1. La première consiste à décrire le système finement pour effectuer un clustering de machines similaires. Une fois ce clustering effectué, la répartition des répliques d'un même document est faite sur des clusters distincts. Dans [24], un *Model builder* construit un modèle du réseau à partir d'informations renseignées par les administrateurs/utilisateurs et par l'observation de l'uptime des pairs. Chaque pair est représenté par un vecteur d'attributs et l'information mutuelle entre les pairs est calculée. À partir de ces mesures de l'information mutuelle, un *SetCreator* génère un ensemble de clusters de machines non-corrélées . Lorsqu'un document est ajouté au système, un serveur central *Disseminator* choisi un cluster pour y stocker les données. Dans [12], les auteurs ont étudié les coréllations sur le réseau de calcul BOINC [1]. Un clustering des pairs est calculé avec k-means à partir de traces collectées sur leur disponibilité. À l'issue de ce clustering, les tâches de calcul sont distribuées dans des clusters en fonction de leur durée. Une tâche longue sera affectée à un cluster ayant une bonne disonibilité alors qu'une tâche plus courte pourra être affectée dans des clusters moins fiables.

2. La seconde idée part du principe qu'un clustering réaliste est trop difficile à obtenir et qu'il faut plutôt essayer de modéliser le degré de corrélation pour pouvoir augmenter la redondance en conséquence. C'est notamment la position défendue par [2, 14, 5]. Glacier [10] est un exemple d'application pair-à-pair qui combine codes d'effacement, répliques et ramasse miette pour moduler subtilement les niveaux de redondances des données en fonction du nombre de fautes que le système subis à un instant donné.

1. http://boinc.berkeley.edu/

2.2 Modèle de flocking d'agents mobiles

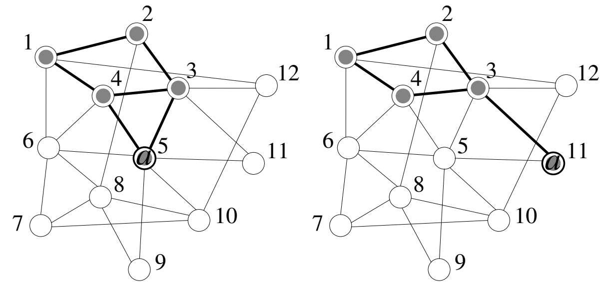
Nous présentons à présent le modèle de flocking détaillé dans [17]. Ce modèle construit un système multi-agents dans lequel chaque donnée est découpée en fragments. Les fragments sont obtenus par codes d'effacement [15]. Un (m, n)-code d'effacement découpe une donnée en m blocs. Après une étape d'encodage, $m+n$ fragments sont générés. Avec ce type d'encodage, la donnée initiale peut être reconstruite à partir de n'importe quel ensemble de m fragments pris parmi les $m + n$. Dans ce cas le système est en mesure de tolérer n pertes de fragments. Après cette étape d'encodage, chaque fragment est encapsulé dans un agent mobile cognitif capable de pendre ses propres décisions. Cette cognition permet aux agents de se déplacer de manière autonome de pairs en pairs en suivant des règles de flocking similaires à celles proposées par Reynolds [19]. Reynolds souhaitait trouver des règles simples, facilement applicables par chaque agent, permettant de reproduire le vol en nuée des oiseaux en faisant émerger le comportement global du groupe par l'application asynchrone de règles locales. Reynolds a identifié trois règles qu'un agent doit suivre :

- cohesion : les agents veillent à ne pas être trop éloignés de leur voisinage ;
- separation : les agents veillent à ne pas être trop proche de leur voisinage ;
- alignement : les agents gardent la même direction que leur voisinage.

Chaque document est alors représenté par la nuée composée de l'ensemble de ses agents fragments. Une mesure de distance entre deux agents a été introduite. Elle est donnée par le temps d'aller-retour (RTT) d'un paquet entre les pairs hébergeurs. Les règles de Reynolds sont ensuite transposée dans un réseau et appliquées de manière asynchrone par chaque agent :

1. un pair ne peut héberger qu'un seul fragment de la même nuée (règle de séparation) ;
2. les agents se déplacent vers l'agent le plus éloigné de leur voisinage (règle de cohésion associée à la distance RTT).

La Fig. 1 présente un réseau de 12 pairs qui héberge une nuée de 5 fragments représentés par les disques grisés. La perception d'un agent est limitée au voisinage du pair qui l'héberge. Ce voisinage est accessible pour l'agent en interrogeant la plateforme agents mobiles. L'algorithme de flocking qui est décrit dans [17] fonctionne comme suit. La première étape pour



(a) agent a avant son déplacement (b) agent a après son déplacement

FIGURE 1 – Nuée de 5 fragments dans un réseau de 12 pairs. Les agents sont représentés par les disques grisés. L'agent a sur le pair 5 se déplace sur le pair 11 et choisi le pair 3 comme pivot.

un agent qui souhaite se déplacer, par exemple l'agent a hébergé sur le pair 5, est de construire $busy$, l'ensemble des pairs voisins qui hébergent des fragments du même document et $free$ les autres voisins. Dans l'exemple, on a $busy_a = \{3, 4\}$ et $free_a = \{6, 9, 10, 11\}$. Ensuite, l'agent construite $anchors$, le sous-ensemble de $busy$ qui contient les pairs les plus éloignés. On considère dans l'exemple que seul le pair 3 est suffisamment éloigné de 5 (au sens du RTT) pour être choisi. On a donc $anchors_a = \{3\}$. Un ensemble de candidats, contenu dans $candidates$, est alors construit en sélectionnant les éléments de $free$ qui sont voisins des éléments de $anchors$. Dans l'exemple, $candidates_a = \{11\}$ car 11 est un voisin de 3 et de 5. La dernière étape de cet algorithme consiste à choisir un candidat dans $candidates$ et à se déplacer sur celui-ci. Dans l'exemple, a décide donc de se déplacer sur le pair 11. Mais une nuée peut perdre des fragments lorsque les pairs qui les hébergent tombent en panne. C'est pourquoi, chaque nuée est dotée d'un mécanismes de supervision et de réparation [21] distribué lui permettant de se régénérer lorsqu'elle a subit un trop grand nombre de pertes d'agents. Un seuil $m \leq r < m + n$ est défini et détermine la taille critique à partir de laquelle une nuée lance la procédure de réparation. Ce mécanisme à seuil est utile pour tolérer les fautes temporaires dues à des ruptures de cohésion.

3 Placement de nuées sur des clusters de corrélations

Ce papier se place dans le cadre de l'approche par clustering. Notre proposition consiste à supprimer l'entité centrale *Disseminator* de [24] et à tirer partie de la mobilité du modèle de Flo-

cking [17] pour permettre à chaque document de trouver la meilleure place par une exploration des clusters. Le tout de manière décentralisée. Nous supposons qu'à l'issue du processus de clustering [24, 12], un ensemble de clusters de pairs correllés est construit et que les pairs d'un même cluster se connaissent entre eux formant un second réseau logique. Notre contribution consiste à proposer un algorithme de déplacement de nuée qui :

1. Minimise le nombre de fragments d'une nuée sur les mêmes clusters pour amortir l'impact des fautes correllées.
2. Répartisse le coût de stockage d'une nuée sur l'ensemble du réseau.
3. Conserve les propriétés du flocking.

Ce placement multi-critère est réalisé à l'aide d'une procédure de recuit-simulé selon la dynamique de Metropolis [13]. Il requiert la définition d'une fonction donnant l'énergie d'une nuée dans une certaine configuration. Cette fonction d'énergie doit contenir les différents critères que nous venons d'énoncer. La force de cet algorithme vient du fait que la fonction d'énergie globale peut être transformée pour être calculée localement par les pairs nous permettant ensuite de construire un algorithme totalement décentralisé. Le principe de l'algorithme consiste à ajouter une couche de prise de décision au dessus du flocking pour conserver ses propriétés. C'est-à-dire que les possibilités de déplacements d'un agent sont toujours celles résultantes de l'algorithme de flocking dans $candidates$. Mais le choix dans cet ensemble est dirigé de sorte que les critères énoncés plus haut soient optimisés.

3.1 Définition du modèle

Le modèle que nous définissons est constitué du graphe non-orienté $\Gamma = (V, E)$ modélisant le réseau pair-à-pair avec V l'ensemble des pairs et E l'ensemble des arrêtes. Un ensemble d'agents mobiles fragments F sont en évolution dans Γ sous la forme d'un ensemble de nuées N tel que $\bigcap_{n_i \in N} n_i = \emptyset$ et $\bigcup_{n_i \in N} n_i = F$. Il ne peut donc y avoir ni de fragments orphelins ni de fragments dans plusieurs nuées. A l'issue du clustering, un ensemble G de clusters de corrélations est généré. Les éléments $g \in G$ sont des sous-ensembles de V tel que pour tout $u, w \in g$, u est corrélé avec w. On considère dans la suite de ce papier que les clusters sont disjoints. Nous définissons un certain nombre de fonctions pour manipuler ce modèle :

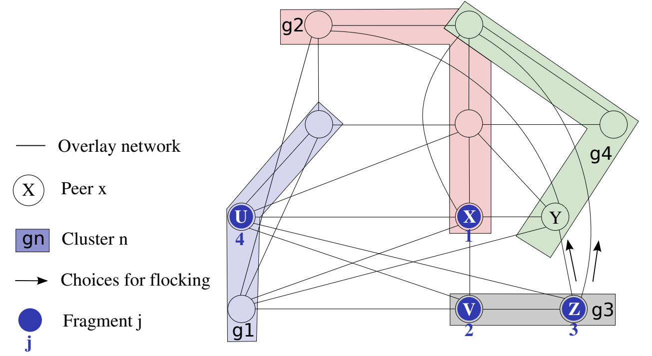


FIGURE 2 – Nuée de 4 fragments dans un réseau de 11 pairs avec 4 clusters de corrélations.

- Soit $v : V \rightarrow 2^V$ la fonction qui retourne le voisinage d'un pair x. On a $y \in v(x) \Leftrightarrow \{x, y\} \in E, x \neq y$.
- Soit $a : F \rightarrow V$ la fonction qui retourne le pair hébergeant un fragment. $u = a(f)$ ssi f est hébergé par u. Soit $\{f_1, \ldots, f_m\} \in N$ une nuée de m fragments. On note par f_j^u que f est le fragment d'identifiant j et qu'il est hébergé sur le pair u. On a donc $a(f_j^u) = u$. On appelle configuration α d'une nuée $n \in N$ une certaine affectation des fragments de n sur les pairs de V que l'on note $n_\alpha = \{(f, u) \in F \times V \,|\, u = a(f), \forall f \in n\}$.
- Soit $t : G \times N \rightarrow 2^V, g, n \mapsto g \cap \{a(f), \forall f \in n\}$. La fonction qui retourne les pairs du cluster g qui hébergent des fragments de la nuée n.
- Soit $clus : V \rightarrow G$. La fonction qui retourne le cluster g dans lequel se trouve un pair u. On a $g = clus(u)$ ssi $u \in g$.
- C un ensemble de fonctions de coût définies de $F \rightarrow \mathbb{R}$ représentant le coût d'hébergement d'un fragment. A titre d'exemple, on peut définir le coût de stockage $\mathcal{V} \in C$ d'un fragment f^u comme suit :

$$\mathcal{V}(f^u) = \frac{\mathcal{O}(u) + \mathcal{T}(f^u)}{\mathcal{E}(u)}$$

  avec $\mathcal{O}$, $\mathcal{E}$ et $\mathcal{T}$ des fonctions qui retournent respectivement : l'espace occupé d'un pair, l'espace total disponible sur un pair et la taille d'un fragment.

Example 1 *La Fig. 2 présente un exemple d'une nuée de 4 fragments répartie sur un réseau de 11 pairs eux même répartis sur 4 clusters de corrélations. La nuée est donc dans la configuration* $\alpha = \{(1, x), (2, v), (3, z), (4, u)\}$.

3.2 Dynamique de Metropolis et fonction d'énergie

La mécanique du recuit selon la dynamique de Metropolis [13] est une procédure itérative qui consiste à trouver une configuration minimisant une fonction d'énergie $\mathbb{E}$, définie sur un espace de configurations D. En partant d'une configuration initiale quelconque $d_0 \in D$, une configuration candidate $d_1 \in D$ choisie dans le voisinage de d_0 est acceptée avec une certaine probabilité qui décroît en fonction de $\mathbb{E}(d_1) - \mathbb{E}(d_0)$. Cette étape est répétée jusqu'à ce qu'un minimum global de $\mathbb{E}$ soit trouvé. L'acceptation de configuration ayant une énergie plus forte permet à l'algorithme d'éviter de rester bloqué dans des minima locaux. Nous introduisons la fonction d'énergie globale d'une nuée $n \in N$ dans un réseau pair-à-pair suivante :

$$\mathbb{E}(n) = \sum_{f \in n} \sum_{c \in C} c(f) + w \sum_{g \in G} |t(g,n)|^\gamma \quad (1)$$

L'énergie d'une nuée dans une configuration α est une quantité qui est égale au coût d'hébergement de ses fragments ajoutée à la proportion de ses fragments sur des pairs corrélés. Plus le nombre de fragments d'une même nuée sur le même cluster (le terme $|t(g,n)|^\gamma$) est élevé et plus l'énergie résultante est élevée. La constante $\gamma \in \mathbb{R}^+$ permet d'affecter une pénalité aux clusters contenant beaucoup de fragments de la même nuée. $w \in \mathbb{R}^+$ est une constante permettant au concepteur du système de moduler l'importance du critère de répartition sur des clusters disjoints par rapport au critère d'équilibrage de charge. Le principe du changement de configuration d'une nuée nécessaire à l'algorithme de recuit-simulé est présenté à la Fig 3. Le passage de la configuration α à la configuration β est réalisé en déplaçant le fragment 3 du pair z au pair y. Les candidats possibles pour les changements de configurations sont ceux retournés par l'algorithme de flocking de la Sec. 2.2. La décentralisation de l'algorithme de recuit-simulé est possible puisque le changement de configuration est local et matérialisé par le déplacement d'un seul agent de la nuée à la fois. La fonction $\mathbb{E}$ peut donc être calculée de manière asynchrone par chaque agent sur une vision locale de sa nuée qui est limitée au voisinage du pair qui l'héberge. Dans la Fig. 3, le fragment f_3 hébergé par le pair z peut calculer localement en z : $\Delta(\mathbb{E}_z) = \mathbb{E}_z(n_\beta, f_3^y) - \mathbb{E}_z(n_\alpha, f_3^z)$. De manière plus générale, on exprime $\mathbb{E}_z(n_\alpha, f_i^z)$ et $\mathbb{E}_z(n_\beta, f_i^y)$ calculés par un fragment f_i comme suit :

$$\begin{aligned}\mathbb{E}_z(n_\alpha, f_i^z) = \sum_{f \in \mathcal{F}} \sum_{c \in C} c(f) + \sum_{c \in C} c(f_i^z) \\ + w \sum_{g \in \mathcal{G}} |t(g, n_\alpha)|^\gamma + w|t(clus(y), n_\alpha)|^\gamma \\ + w|t(clus(z), n_\alpha)|^\gamma \quad (2)\end{aligned}$$

$$\begin{aligned}\mathbb{E}_z(n_\beta, f_i^y) = \sum_{f \in \mathcal{F}} \sum_{c \in C} c(f) + \sum_{c \in C} c(f_i^y) \\ + w \sum_{g \in \mathcal{G}} |t(g, n_\beta)|^\gamma + w|t(clus(y), n_\beta)|^\gamma \\ + w|t(clus(z), n_\beta)|^\gamma \quad (3)\end{aligned}$$

avec

$$\mathcal{F} = \{f \in n \,|\, a(f) \in v(z) \cup v(y)\} \setminus f_i$$

l'ensemble des fragments de n sur les voisinages de z et y qui restent immobiles. Et

$$\begin{aligned}\mathcal{G} = \{clus(a(f)) \,|\, a(f) \in v(z) \cup v(y), \\ \forall f \in n\} \setminus \{clus(z), clus(y)\} \quad (4)\end{aligned}$$

l'ensemble des clusters qui possèdent des fragments de n dans le voisinage de y et z mais qui ne sont ni la cible ni la source du déplacement. On élimine les termes constants et on obtient :

$$\begin{aligned}\Delta(\mathbb{E}_z) = \sum_{c \in C} c(f_i^y) + w|t(clus(y), n_\beta)|^\gamma \\ + w|t(clus(z), n_\beta)|^\gamma \\ - \sum_{c \in C} c(f_i^z) - w|t(clus(y), n_\alpha)|^\gamma \\ - w|t(clus(z), n_\alpha)|^\gamma \quad (5)\end{aligned}$$

Example 2 *Sur la Fig. 3 si l'on considère* $C = \emptyset$, $w = 1$, $\gamma = 3$, *et que* f_3^z *est le fragment souhaitant se déplacer on a :* $\mathcal{F} = \{f_1, f_2, f_4\}$, $\mathcal{G} = \{g_1, g_2\}$, $\Delta(\mathbb{E}_z) = 4 - 10 = -6$. *Donc un gain d'énergie à déplacer* f_3 *en* y.

3.3 Algorithme de placement

Pour trouver le minimum de la fonction $\mathbb{E}$, l'algorithme de Metropolis utilise la distribution de Gibbs, que l'on note $G_T(s)$. Cette loi est issue de la thermodynamique statistique et probabilise un espace S en prenant comme densité de probabilité la fonction :

$$G_T(s) = \frac{1}{Z_T} \exp\left(\frac{-\mathbb{E}(s)}{T}\right) \quad (6)$$

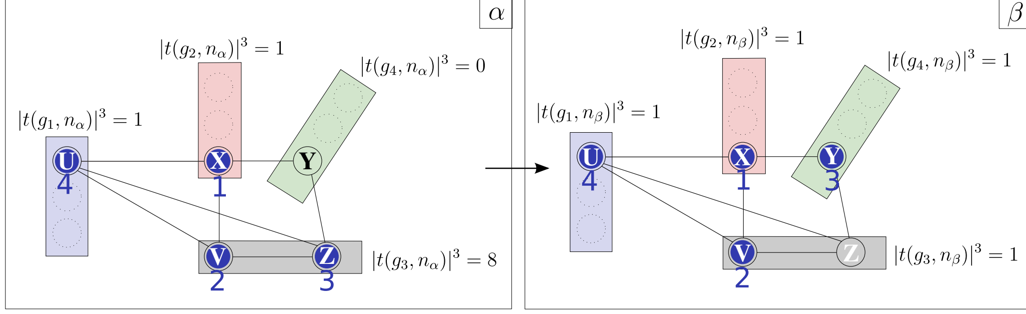


FIGURE 3 – Transistion de la configuration α à la configuration β par déplacement de f_3. Le graphe présenté est le même que celui de la Fig. 2 mais dans une forme filtrée.

avec Z_T la constante de normalisation :

$$Z_T = \sum_{s_i \in S} \exp\left(\frac{-\mathbb{E}(s_i)}{T}\right) \quad (7)$$

et avec le paramètre $T > 0$ appellé température du système. Lorsque T tends vers 0, la distribution G_T se concentre uniformément sur l'ensemble de ses minima globaux. L'algorithme de Metropolis repose sur une chaîne de Markov $x_T(\tau)$ définie sur l'ensemble des configurations S dans laquelle la température $T(\tau)$ est constante. L'ensemble des états correspond à l'ensemble des configurations. Les états s'_i accessibles à partir d'un état s sont propres au problème et nécessitent la définition d'une fonction de voisinage sur les configurations. Si l'état courant de la chaine $x(\tau)$ est égale à i, le choix de n'importe quel voisin candidat j de i est tiré aléatoirement et est notée q_{ij}. La probabilité de transition entre deux états $i, j \in S$ dans la chaîne est donnée par :

$$\mathbb{P}[x(\tau+1) = j | x(\tau) = i] = q_{ij} min\left(\exp\left(-\frac{\mathbb{E}(j) - \mathbb{E}(i))}{T}\right), 1\right) \quad (8)$$

si $i \neq j$. $x_T(\tau)$ est reversible, irreductible et apériodique et sa distribution de probabilité est donnée par l'Eq.(6). Cette chaine possède la propriété suivante :

$$\lim_{\substack{\tau \to +\infty \\ T\text{fixé}}} \mathbb{P}_T(x(\tau) = s) = G_T(s), \quad s \in S \quad (9)$$

L'algorithme de Metropolis converge donc vers une configuration d'énergie minimale. Pour plus de détails se référer à [3]. Nous revenons sur le placement des nuées de fragments. L'algorithme 1 décrit MINCOR, la procédure de déplacement par recuit-simulé exécutée par chaque agent sur un pair z quelconque de manière asynchrone. La décision de changer de configuration est prise en suivant la dynamique de Metropolis à température constante T de l'Eq. 8 et utilise la fonction d'énergie $\mathbb{E}_z$ définie dans l'Eq. 5. Les variables t_y et t_z comptent les fragments de $\{n\} \backslash f_i$ qui sont respectivement dans le cluster de y et dans le cluster de z. C'est pourquoi $|t(clus(y), n_\alpha| = t_y$ et $|t(clus(y), n_\beta| = t_y + 1$ parceque dans la configuration β, f_i est passé dans le cluster de y. De même, $|t(clus(z), n_\beta| = t_z$ et $|t(clus(z), n_\alpha| = t_z + 1$ parceque dans la configuration α, f_i est dans le cluster de z. L'appel de la fonction RANDOMFLOCKINGCANDIDATES(Z) exécute l'algorithme de flocking et retourne un pair tiré aléatoirement dans la liste des pairs candidats pour le déplacement. L'appel de PEERSOFCLUSTER(y) retourne la liste des pairs du cluster de y. Cette fonction nécessite l'envoie de messages par le réseau entre le pair z et le pair y.

4 Structure du réseau

Dans [17, 21], les agents évoluent dans SCAMP (Scalable Membership Protocol) [8], un protocole totalement décentralisé pour la gestion de groupes multicasts. Ce protocole a été initialement conçu pour supporter la diffusion fiable dans des groupes multicasts à l'aide de protocoles épidémiques. Nous l'utilisons dans nos travaux pour ses bonnes propriétés en terme de tolérance aux fautes et de passage à l'échelle qui sont indispensables pour assurer la sûreté de fonctionnement requise au niveau réseau. SCAMP construit un graphe aléatoire orienté suivant le modèle de Erdős et Rényi [7] et possédant un degré moyen qui converge vers $(\phi + 1)\log(|V|)$ avec $|V|$ le nombre de pairs du réseau et ϕ un paramètre du système utilisé pour

Algorithm 1: MINCOR

Input: f_i^z l'agent fragment en z exécutant l'algorithme, n la nuée de f_i, C un ensemble de fonctions de coût, γ la constante de pénalité, w le facteur de gain

$y \longleftarrow$ RANDOMFLOCKINGCANDIDATES(Z)
$cluster_z \longleftarrow$ PEERSOFCLUSTER(z)
$cluster_y \longleftarrow$ PEERSOFCLUSTER(y)
$cost_z \longleftarrow 0$
$cost_y \longleftarrow 0$
foreach $c \in C$ **do**
  $cost_z \longleftarrow cost_z + c(f, z)$
  $cost_y \longleftarrow cost_y + c(f, y)$
$t_z \longleftarrow 0$
$t_y \longleftarrow 0$
foreach $peer \in cluster_z$ **do**
  if *peer héberge un fragment* f_j *de* n **then**
    if $j \neq i$ **then**
      $t_z \longleftarrow t_z + 1$
foreach $peer \in cluster_y$ **do**
  if *peer héberge un fragment* f_j *de* n **then**
    if $j \neq i$ **then**
      $t_y \longleftarrow t_y + 1$

$$\Delta(\mathbb{E}_z) \longleftarrow cost_y + w\big((t_y+1)^\gamma + t_z^\gamma\big) - cost_z - w\big(t_y^\gamma - (t_z+1)^\gamma\big)$$

$$p \leftarrow \min\left(\exp\left(\frac{-\Delta(\mathbb{E}_z)}{T}\right), 1\right)$$

$tirage \leftarrow$ Random float choice in $[0, 1]$
if $tirage \leq p$ **then**
  AGENTMOVE(y)

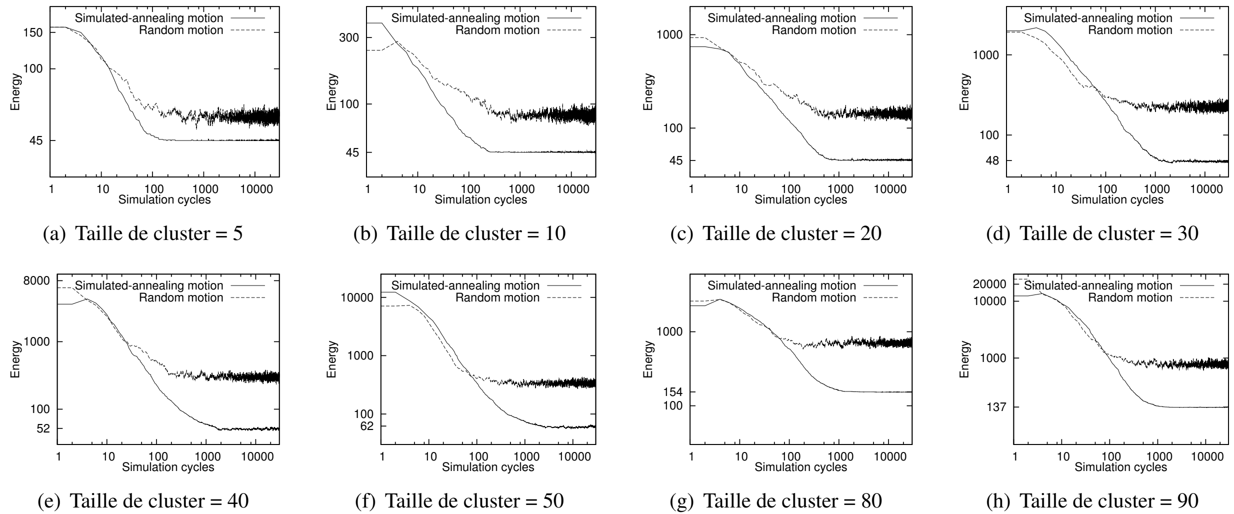
moduler le degré du réseau. Cette propriété permet au graphe de rester connecté si la proportion d'arcs perdus reste inférieure à $\dfrac{c}{c+1}$. De plus, le réseau passe à l'échelle en terme de taille de voisinage puisque son degré droit logarithmiquement. Nous décidons d'utiliser dans nos expérimentation les même paramètres réseau que ceux présentés dans [21] pour conserver une cohérence dans les résultats.

5 Résultats expérimentaux

Nous proposons dans cette section d'évaluer l'algorithme MINCOR. Dans un premier temps nous évaluons la qualité du placement par recuit simulé par rapport à un placement aléatoire. Cette évaluation est faite en collectant les valeurs d'énergie de nuées exécutant ces deux algorithmes et en les comparant. Dans un second temps nous évaluons le nombre de fautes concurrentes obtenues avec MINCOR lorsque le système subis des fautes corrélées. L'architecture a été implémentée sur oRis [11], un simulateur multi-agents. La durée d'une simulation est fixée à 30000 cycles. Chaque cycle correspond à 60 secondes en temps réel. La simulation s'étend donc sur une période d'approximativement 21 jours. Chaque expérience a été reproduite 70 fois et ce sont les valeurs moyennes qui sont données. Le réseau pair-à-pair supportant l'architecture est de type SCAMP avec 1000 pairs et la constante $\phi = 20$. Les taux de transfert réseau sont contenus entre 64kb/s et 640kb/s pour le débit montant et la taille d'un fragment est fixée à 64Mo.

5.1 Qualité du placement

Dans cette expérience, deux nuées de 45 fragments sont déployées dans le réseau. Ce choix de paramètres résulte de l'étude menée dans [21]. Il détermine le nombre de fragments nécessaires pour que la cohésion d'une nuée soit préservée en fonction de la taille et du degré du réseau dans lequel elle évolue. Les décisions de mouvement sont prise par chacun des agents à chaque cycle de simulation avec une probabilité de 0.1. La première nuée se déplace avec l'algorithme classique (déplacement aléatoire) et l'autre utilise l'algorithme MINCOR avec les paramètres $w = 1$, $\gamma = 3$, $C = \emptyset$ et $T = 1$. Nous souhaitons mesurer les performances du placement, c'est pour cette raison que la composante de coût d'hébergement dans la fonction d'énergie est ignorée. Nous mesurons l'énergie des deux nuées au cours de la simulation pour des clusters de tailles 5, 10, 20, 30, 40, 50, 80, 90 et 100. Une taille de cluster de 5 dans un réseau de 1000 pairs signifie qu'il y a un total de 200 clusters. Les résultats de la Fig. 4, en échelle logarithmique, montrent l'évolution moyenne de l'énergie des deux nuées au cours des simulations. Nous observons d'une manière générale sur l'ensemble des graphiques, que l'énergie des nuées évoluant avec MINCOR est plus faible que celles évoluant de manière aléatoire avec l'algorithme de flocking classique. Nous remarquons également que lorsque le nombre de clusters est supérieur ou égal au nombre de fragments Fig. 4(a), Fig. 4(b) et Fig. 4(c), l'énergie du placement par recuit est égale au nombre de

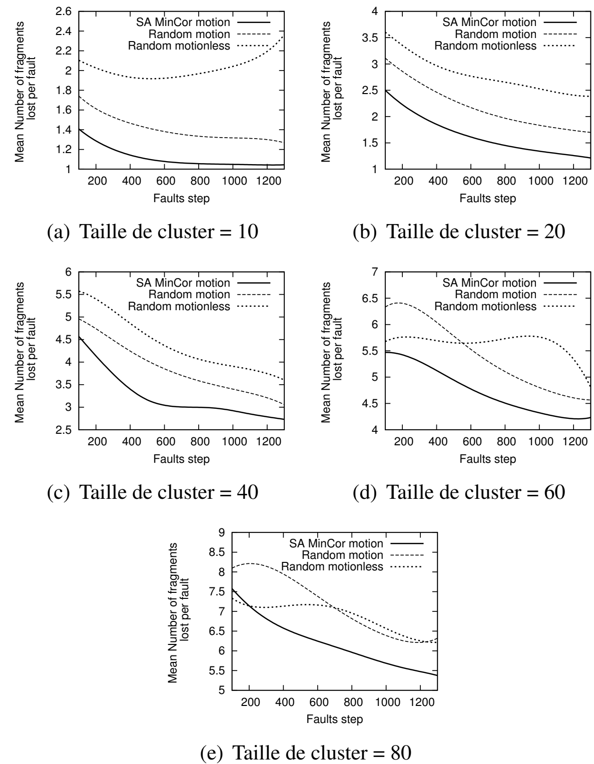


(a) Taille de cluster = 5 (b) Taille de cluster = 10 (c) Taille de cluster = 20 (d) Taille de cluster = 30

(e) Taille de cluster = 40 (f) Taille de cluster = 50 (g) Taille de cluster = 80 (h) Taille de cluster = 90

FIGURE 4 – Evolution de l'énergie moyenne des nuées pour le placement par flocking aléatoire et pour le flocking par recuit.

fragments de la nuée. Ceci montre que la nuée a été capable de placer chacun de ses fragments dans des clusters distincts tout en conservant sa localité (i.e., trouver une configuration d'énergie minimale). On constate à la Fig. 4(d) et à la Fig. 4(e) que lorsque le nombre de clusters devient inférieur à 45, le minimum d'énergie atteint avec MINCOR est supérieur à 45. Dans ces cas, la nuée n'a pas réussi à placer la totalité de ses fragments sur des clusters distincts. D'une manière générale, plus la taille des clusters augmente et plus le minimum d'énergie atteint par la nuée est élevé. On constate également que la convergence de MINCOR est relativement rapide et varie entre 100 et 1000 cycles, caractéristique d'une descente par l'algorithme de Metropolis. Finalement, on remarque de fortes valeurs d'énergie initiales. Cette configuration initiale vient du fait que dans l'implémentation, le déploiement des nuées est effectuée à partir d'un pair initiateur. De ce fait, tous les fragments au démarrage se retrouvent dans le même cluster avant de commencer à se déplacer générant une valeur d'énergie anormalement élevée. Avec cet ensemble d'expérimentations nous montrons qu'une nuée exécutant l'algorithme MINCOR est capable de trouver une répartition qui minimise le nombre de fragments sur des clusters de pairs corrélés de manière asynchrone et décentralisée.



(a) Taille de cluster = 10 (b) Taille de cluster = 20

(c) Taille de cluster = 40 (d) Taille de cluster = 60

(e) Taille de cluster = 80

FIGURE 5 – Nombre moyen de fragments perdu à chaque fois qu'un nuée subit une faute.

5.2 Mesure de pertes simultanées

Nous proposons dans cette section de mesurer le nombre de pertes simultanées de fragments ob-

tenus pour des nuée de 45 fragments lorsque les fautes sont corrélées. La première nuée se déplace en suivant l'algorithme de flocking classique, la seconde cherche un placement initial et ne se déplace plus en suite. La troisième utilise MINCOR pour ses déplacements. Nous introduisons un modèle de fautes qui déconnecte un pair p tous les $faultstep$ cycles. La faute a pour conséquence la déconnexion de tous les pairs du cluster de p. La première faute survient après le 1000e cycle pour laisser le temps à MINCOR de converger. Le réseau est ensuite placé dans un état stationnaire : pour chaque pair déconnecté au cycle τ, un nouveau pair est reconnecté au cycle $\tau + 1$. Le nombre de clusters est préservé par un clustering périodique du réseau lorsque suffisamment de pairs se sont reconnectés. Les mécanismes de supervision et de réparation décrit dans [21] sont activés. Ce qui signifie que chaque nuée est fragmentée avec un $(10, 35)$-code d'effacement et qu'un seuil de réparation $r = 13$ est défini. En d'autres termes, lorsque la taille d'une nuée atteint r à la suite de plusieurs fautes, une procédure de réparation est lancée pour restaurer les 45 fragments. Nous mesurons dans les graphiques de la Fig. 5 le nombre moyen de pertes simultanées de fragments pour chaque type de nuée. i.e., lorsqu'une nuée subit une faute, on compte le nombre de fragments qu'elle a perdu suite à cette faute. On constate que MINCOR est le meilleur algorithme de placement et qu'une nuée prenant ses décisions avec cet algorithme perd moins de fragments lorsqu'elle subie une faute que les autres nuées dans quasiment toutes les situations. Le comportement erratique, dans certains cas, de la courbe représentant le placement immobile s'explique par le fait qu'une nuée immobile ayant un grand nombre de ses fragments placés dans le même cluster a de plus grandes chances de perdre plus de fragments d'un coups. Le résultat est donc dépendant du choix de placement initial. Ce phénomène est amplifié pour les clusters de grande taille. On remarque également que les courbes sont décroissantes en fonction du $faultstep$. L'explication provient du fait que les nuées sont beaucoup plus perturbées durant leur phase de réparation lorsqu'il y a beaucoup de fautes. En effet, le processus de réparation [21] duplique m (provenant du (m, n)-code d'effacement) agents et les envoie sur un pair $cible$ pour reconstruire les fragments manquants. Mais si $cible$ est perdu, les agents dupliqués qui sont déjà arrivé dessus sont également perdu résultant en un nombre de pertes plus important. Finalement, on observe qu'une nuée qui se déplace aléatoirement selon l'algorithme classique fourni de meilleurs résultats qu'une placement immobile dans tous les cas à l'exception d'une taille de cluster de 80. Sa mobilité à elle seule semble suffisante à éviter certaines corrélations. On note qu'il est normal d'avoir un nombre de fautes décroissant en fonction du $faultstep$ puisque la durée de la simulation est fixée.

6 Conclusion

Nous avons présenté MINCOR, un algorithme permettant de réduire l'impact des fautes corrélées dans les systèmes de stockage pair-à-pair. Cet algorithme repose sur un système multi-agents qui transforme les données en nuées d'agents mobiles capables de trouver un placement qui minimise le nombre d'agents d'une même nuée sur des pairs corrélés. Cet algorithme est décentralisé et exécuté de manière asynchrone par chaque agent sur une vision locale de son environnement. Les expériences que nous avons conduites ont confirmé que MINCOR trouve le placement attendu et que lorsque les fautes sont corrélées et que le système est dans un état stationnaire, les nuées exécutant MINCOR subissent moins de pertes simultanées de fragments que les autres méthodes de placement. Réduisant ainsi l'impact des fautes corrélées sur le système. En dehors de cette application aux fautes corrélées, nous donnons un algorithme décentralisé qui permet de diriger une nuée d'agents mobiles dans un réseau à l'aide d'une fonction d'objectif sans altérer la propriété de flocking. Dans des travaux futurs il sera intéressant d'évaluer l'équilibre nécessaire entre le terme de répartition et le terme de coût dans la fonction d'énergie pour obtenir un bon placement sur les clusters et un bon équilibrage de charge.

Références

[1] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger Wattenhofer. Farsite : Federated, available, and reliable storage for an incompletely trusted environment. In *OSDI*, 2002.

[2] Mehmet Bakkaloglu, Jay J Wylie, Chenxi Wang, and Gregory R Ganger May. On Correlated Failures in Survivable Storage Systems. *Information Systems*, 2002.

[3] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical Science*, 8(1) :10–15, February 1993.

[4] Frank Dabek, M. Frans Kaashoek, David R. Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *SOSP*, pages 202–215, 2001.

[5] Olivier Dalle, Frédéric Giroire, Julian Monteiro, and Stephane Perennes. Analysis of Failure Correlation Impact on Peer-to-Peer Storage Systems. In *P2P*, pages 184–193, 2009.

[6] Abdulhalim Dandoush. *Analysis and optimization of peer-to-peer storage/backup systems*. PhD thesis, Université de Nice Sophia-Antipolis, 2010.

[7] P Erdős and A Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5 :17–61, 1960.

[8] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. SCAMP : Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication. In *Proc. NGC*, pages 44–55, 2001.

[9] Frédéric Giroire, Julian Monteiro, and Stéphane Pérennes. P2P storage systems : How much locality can they tolerate ? In *IEEE LCN*, pages 320–323, 2009.

[10] Andreas Haeberlen, Alan Mislove, and Peter Druschel. Glacier : Highly durable, decentralized storage despite massive correlated failures. In *NSDI*, 2005.

[11] F Harrouet. *oRis : s'immerger par le langage pour le prototypage d'univers virtuels à base d'entités autonomes*. PhD thesis, Université de Bretagne Occidentale, 2000.

[12] Derrick Kondo, Artur Andrzejak, and David P. Anderson. On correlated availability in internet-distributed systems. In *GRID*, pages 276–283, 2008.

[13] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 1953.

[14] Suman Nath, Haifeng Yu, Phillip B. Gibbons, and Srinivasan Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. In *NSDI*, 2006.

[15] James S. Plank. A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. Technical Report CS-96-332, University of Tennessee, July 1996.

[16] Hugo Pommier and François Bourdon. Agents mobiles et réseaux pair-à-pair vers une gestion sécurisée de l'information répartie. *Revue d'Intelligence Artificielle*, 23(5-6) :697–718, 2009.

[17] Hugo Pommier, Benoît Romito, and François Bourdon. Bio-inspired Data Placement in Peer-to-Peer Networks - Benefits of using Multi-agents Systems. In *WEBIST*, pages 319–324, 2010.

[18] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.

[19] Craig W. Reynolds. Flocks, herds and schools : A distributed behavioral model. In *SIGGRAPH*, pages 25–34, 1987.

[20] Sean C. Rhea, Patrick R. Eaton, Dennis Geels, Hakim Weatherspoon, Ben Y. Zhao, and John Kubiatowicz. Pond : The oceanstore prototype. In *FAST*, 2003.

[21] Benoît Romito, Hugo Pommier, and François Bourdon. Repairing flocks in peer-to-peer networks. In *IAT*, 2011.

[22] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[23] Hakim Weatherspoon and John Kubiatowicz. Erasure coding vs. replication : A quantitative comparison. In *Proc. IPTPS*, pages 328–338. Springer-Verlag, 2002.

[24] Hakim Weatherspoon, Tal Moscovitz, and John Kubiatowicz. Introspective failure analysis : Avoiding correlated failures in peer-to-peer systems. In *SRDS*, pages 362–, 2002.

[25] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz. Tapestry : a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1) :41–53, 2004.