Sauvegarde collaborative entre pairs

Ludovic Courtès, Yves Deswarte,
Marc-Olivier Killijian, David Powell, Matthieu Roy
LAAS-CNRS

Introduction

Pourquoi une approche coopérative ?

- Utilisation de **ressources excédentaires**
- Exemple : les capacités de stockage augmentent plus vite que les données pour les remplir
- **Simplicité** de mise en œuvre
- **Confiance** dans un système décentralisé : respect de la vie privée, pas de « tiers de confiance »

S'inspirer des systèmes pair-à-pair pour MoSAIC

- Des similitudes, des différences
- Ce qui peut nous être utile

- **Répartition des données dans les systèmes PàP**
  - Répartition des données
  - Localisation des données sur le réseau
  - Fragmentation + décentralisation = CAN
  - Algorithmes de localisation et routage
  - Condensé + chiffrement = CHK
  - Codage des fichiers
- Application à la sauvegarde coopérative
- Et MoSAIC ?

Répartition des données

- **Approche naïve** (Napster, Gnutella, etc.)
  - stockage de fichiers entiers
  - → **Problème** : disponibilité, égale répartition difficile, migration des données problématique, aucune résistance à la censure
- **Fragmentation, dissémination (redondance)** (e.g. GNUnet, Freenet)
  - Fragmentation en blocs de taille fixe (e.g. 1Ko) ou pas
  - Dissémination lors d'interactions entre pairs
  - Résout les problèmes évoqués ci-dessus

Localisation des données sur le réseau

- **Approche centralisée** (e.g. Napster)
  - un serveur de désignation central (ou plusieurs)
  - puis échange de fichiers entiers entre pairs
  - → **Passage à l'échelle, disponibilité**
- **Approche décentralisée** (e.g. Gnutella, GNUnet, Freenet, etc.)
  - chaque nœud **participe au service de désignation**
  - pas de point central

Fragmentation + décentralisation = CAN

- Considérer répartition et localisation des données **conjointement**
- **Idée : adresser les données par leur contenu**
  - ***Content Addressable Network*** (CAN)
  - Comme dans une table de hachage, **calcul d'un condensé** (*hash*)
  - Données **indexées** par leur condensé
- **Table de hachage répartie** (*distributed hash table*, DHT)
  - Mêmes primitives qu'une table de hachage
  - Chaque nœud responsable d'un intervalle d'index (une « *zone* »)
  - Bon condensé ⇒ Bonne répartition

Algorithmes de localisation et routage

- **Utilisation de *réseaux virtuels* (ou *overlay networks*)**
  ⇒ Service de désignation réparti
  - Chacun connaît ses « voisins » et pratique le téléphone arabe
  - Comment faire passer une requête à un pair λ ?

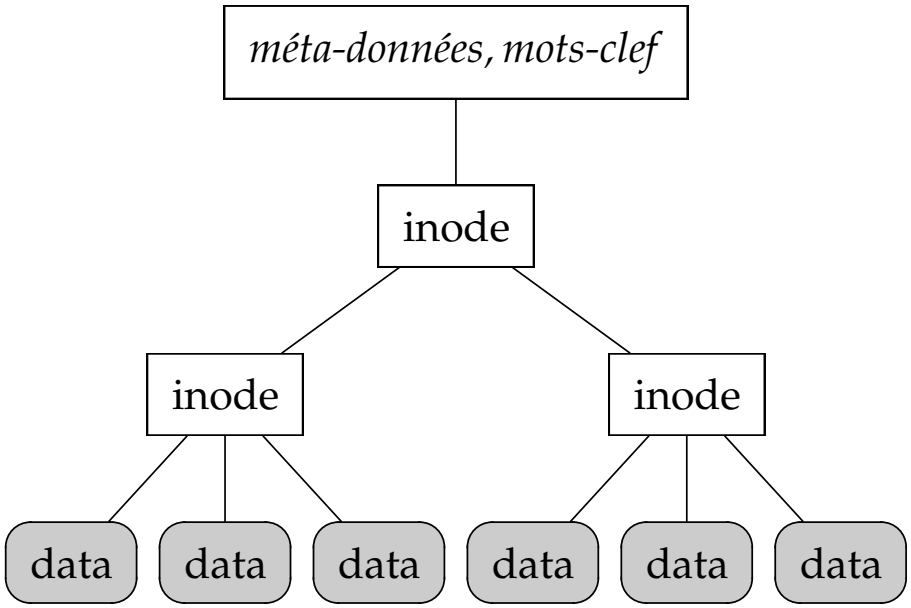
- **Techniques utilisées**
  - Réseaux **structurés**
    + Placement des pairs/données très contraint (DHT)
    + Routage/localisation déterministes, performants
  - Réseaux **non structurés**
    + Placement non contraint
    + Routage/localisation non déterministes, dynamiques
    + Résistants aux malveillances (GNUnet, Freenet, Gnutella, etc.)

Condensé + chiffrement = CHK

- **L'idée : le chiffrement convergent (*convergent encryption*)**
  - Objectif : **suppression de la redondance**
  - Utiliser une **clef de chiffrement dépendant du contenu**
  - Permet d'utiliser des **méthodes de chiffrement « légères » (symétriques)**
  - Utilisé pour le partage de fichiers (GNUnet, Freenet) et l'archivage (Venti)

- **En pratique : *Content-Hash Key* (CHK)**
  - Soit un bloc B, sa **clef de chiffrement** est $K=H(B)$
  - Le **bloc chiffré** est $C=E_K(B)$
  - Le couple $(H(C), K)$ suffit à **désigner** puis **déchiffrer** le bloc B

Codage des fichiers



- Représentation en **arbre** (GNUnet, CFS, Venti)
- Pas de distinction entre **méta-données** et **données**
- Permet d'utiliser des **blocs de taille fixe**
- La CHK de l'inode supérieur est un **point d'entrée suffisant**

- Répartition des données dans les systèmes PàP
- **Application à la sauvegarde coopérative**
  - Introduction
  - Les différentes approches au stockage
  - Les différentes approches (suite)
  - Exemple d'architecture : Pastiche/PeerStore
  - Choix des partenaires
  - S'assurer de la sauvegarde de données
  - Gérer les déconnexions de partenaires
  - Assurer la justice des contributions
- Et MoSAIC ?

Introduction

Quelques travaux

- Cooperative backup scheme (2002, Rice Univ.), Pastiche/Samsara (2002, MIT), PeerStore (2004, TU München), Venti-DHash (2003, MIT), ABS (2004, MIT), etc.
- Accent sur **les incitations à la contribution** et la **résistance aux malveillances**
- Également **élimination des redondances**, **performances de la sauvegarde et récupération**

Des techniques issues des réseaux pair à pair

- Routage entre pairs et découverte décentralisée de pairs
- Chiffrement convergent
- Indexation du contenu

Les différentes approches au stockage

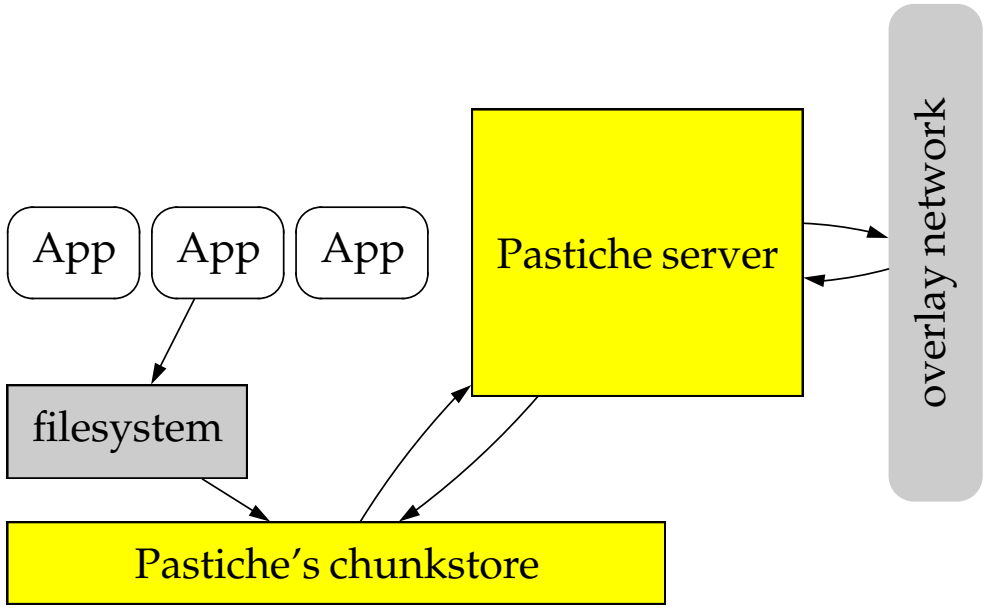
- **Approche simple**
  - Elnikety et. al (2002) : un **serveur central** permet de trouver des pairs, établissement de **plusieurs relations deux-à-deux**, données rassemblées (≈ `tar`) et **chiffrées avec une clef publique**
  - Mécanismes pour **résister aux malveillances**
  - ⇒ Sauvegarde et restauration **lentes et coûteuses**, n'exploite pas la **redondance entre versions**, nécessite de **stocker sa clef privée ailleurs**

- **Approche pair à pair utilisant une DHT**
  - **Efficace** (stockage et temps d'accès)
  - Très **sensible aux malveillances**
  - **Difficulté d'établir des contributions justes** (tout le monde contribue autant)

Les différentes approches (suite)

- **Approches pair à pair « évoluées »**
  - **chiffrement convergent** et algorithmes de **découverte de sous-ensembles communs** entre versions
  - Choix des pairs **en fonction de la redondance et de la localité**
  - **Pastiche**
    - + ***Overlay network*** utilisé pour la découverte (avec **distance ≡ latence** ou **distance ≡ taux de couverture**)
    - + Ensuite, **établissement de liens directs entre partenaires**
  - **PeerStore**
    - + hybride : utilise une **DHT pour stocker les méta-données** et des **relations directes entre pairs** pour les données
    - + ⇒ Permet de rapidement savoir si un **bloc est déjà sauvegardé dans le système**

Exemple d'architecture : Pastiche/PeerStore



- Couche de **stockage par bloc avec indexation par condensé (*chunkstore*)**
- Chaque bloc est stocké sous la forme d'un **journal** (séquence de versions du bloc)
- **Méta-données également sous forme de bloc**
- Données locales sauvegardées dans *chunkstore*
- Données distantes sauvegardées dans *chunkstore*
- Le serveur est **connecté à des partenaires**
- Le serveur **émet périodiquement les blocs à ses amis**

Choix des partenaires

Dans *Pastiche* : partenaires **choisis en fonction des similitudes avec leurs données**

1. Chaque participant **calcule un *résumé* de ses données** (un sous-ensemble des condensés des blocs de données)
2. Le nouveau nœud **diffuse son résumé** et obtient en réponse celui des autres
3. Le nœud peut **choisir parmi ceux-là** ou **joindre un réseau dont la métrique est la couverture des données**

⇒**Seules les données qui n'existent pas chez le partenaire sont sauvegardées**

S'assurer de la sauvegarde de données

- **L'idée : lancer régulièrement un défi aux partenaires**
- L'approche simple : **envoyer une requête de lecture d'*un* bloc au hasard**
- L'approche « évoluée » (Pastiche, PeerStore) :

  1. Le nœud envoie **une valeur** h_0 et une **liste de *n* blocs à vérifier** à son partenaire
  2. Le partenaire renvoie **un condensé** h_n calculé comme suit :
     $h_{i+1} = H(concat(b_{i+1}, h_i))$

  $\Rightarrow$ **Peu coûteux, permet de quantifier les risques**

Gérer les déconnexions de partenaires

- Problème : **tolérer les déconnexions temporaires de partenaires**
- L'idée : en cas de non disponibilité d'un pair, lui assurer une **période de grâce** (Elnikety et al.)
- Problème : **éviter que des nœuds profitent de la période de grâce sans contrepartie**
- Solution de base : **la période de grâce est suivie d'une période pendant laquelle les restaurations ne sont pas autorisées** (Elnikety et al.)
- Solution « évoluée » (Pastiche/Samsara)
  - Objectif : **avoir une punition progressive**
  - Hypothèse : **celui qui sauvegarde connaît le nombre de répliques dont dispose celui qui ne répond pas**
  - Méthode : **à chaque non réponse d'un nœud, celui qui sauvegarde pour lui efface au hasard un bloc avec une probabilité bien choisie**
  - Conclusion : un nœud ne perd ses données que s'il n'a pas répondu **pendant une longue période**

Assurer la justice des contributions

- **Échanges symétriques** (Elnikety et al.)
  - Problème : **ajout d'une contrainte au placement des données**

- **Création d'échanges symétriques** (Samsara, PeerStore)
  - Samsara
    1. Le demandeur **donne la garantie qu'il donne autant de ressources**
    2. En pratique, le créancier fait stocker des données au débiteur qui **dont le contenu ne peut être deviné que par le créancier**
    3. La garantie peut être **vérifiée régulièrement**, comme les autres données
    4. En cas de manque d'espace, on peut **faire suivre une garantie**

  - PeerStore : un nouveau nœud **diffuse une offre et reçoit des offres de ses pairs** qu'il peut refuser ou accepter (en fonction de leur équilibre)

- Répartition des données dans les systèmes PàP
- Application à la sauvegarde coopérative
- **Et MoSAIC ?**
  - Choix des données à effacer
  - Problème de la restauration dans MoSAIC
  - Problématiques dans MoSAIC (suite)
  - Problématiques dans MoSAIC (suite)
  - Assurer la justice des contributions

Choix des données à effacer

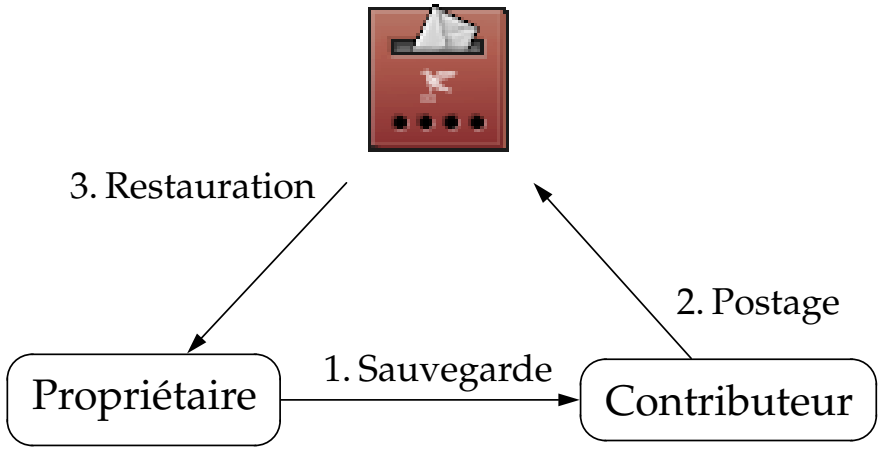
- Le dilemme
  1. Le propriétaire **sait quoi effacer** mieux que le contributeur
  2. Le contributeur **ne fait pas confiance** au propriétaire

- Un problème ouvert
  - Sujet **peu traité** dans les systèmes de sauvegarde PàP
  - Dans MoSAIC : les pairs **se perdent de vue**

Problème de la restauration dans MoSAIC

- **Dans l'article pour *MPAC 2004***
  - **Mode *push*** : le contributeur envoie les données *vers* leur propriétaire
  - **Mode *pull*** : le propriétaire fait une requête auprès des contributeurs

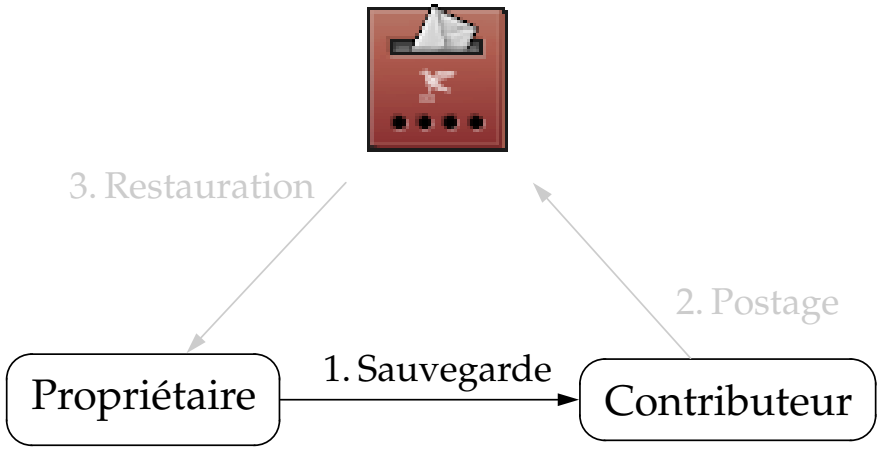
- **Plus généralement : mécanisme de boîte aux lettres**
  1. Données **stockées chez le contributeur**
  2. Données **stockées dans la boîte aux lettres** du propriétaire
  3. Récupération des données par leur propriétaire **à partir de la BàL**



Problématiques dans MoSAIC (suite)

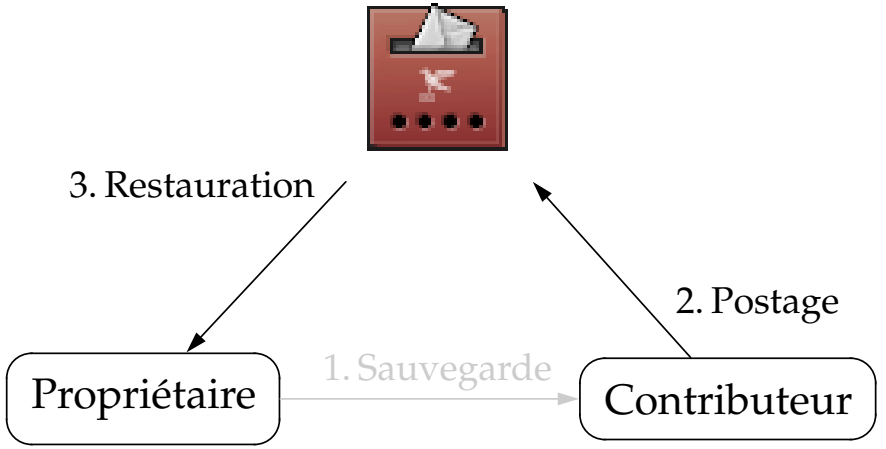
Sauvegarde

1. Découverte de ressources
2. Allocation de ressources
3. Identification
4. Stockage chez le contributeur



Restauration

1. Identification
2. Postage
3. Restauration
4. Maintenance des relations de confiance



Problématiques dans MoSAIC (suite)

- **Identification** des données
  - **Identification unique** (mode *push*)
  - Identification **par des méta-données** : « *les données sauvegardées par Bob jeudi dernier* » (mode *pull*)

- **Stockage des données**
  - En mode ***ad hoc***
  - En mode **connecté** (à une infrastructure)
  - En mode **intermittent** (connexion brève à une infrastructure)

- **Modèle de confiance**
  - En mode *ad hoc et* en mode connecté
  - **Faire le lien** entre ces deux « communautés » de participants

Assurer la justice des contributions

- **Dans MoSAIC : fort taux de renouvellement**
  - Relations **éphémères** entre mobiles
  - **Relations symétriques impossibles**
  - Impossibilité de connaître tous les participants
    ⇒ **Réputation**

- **Dans MoSAIC : plusieurs partitions**
  - Coexistence de **plusieurs partitions/réseaux de participants**
  - Passage de la réputation **entre partitions**
  - Passage de la réputation **entre médiums** (e.g. ad hoc et Internet)

La Fin

Questions ?