`btrsync`
Un `rsync` exploitant intelligemment les déplacements et les copies de sous-arborescences

Antoine Amarilli et Pablo Rauzy

École normale supérieure

Le problème

- On souhaite transférer une arborescence de fichiers d'une machine à une autre sur un lien de faible bande passante.
- Pour accélérer le transfert, on veut éviter de transférer à nouveau ce qui existe déjà sur la machine distante.
- Applications : backups, mirroring, etc.

rsync

- `rsync` est un logiciel très fréquemment utilisé pour cette tâche.
- `rsync` détecte quand un fichier cible est déjà présent et optimise le transfert pour ne pas retransférer entièrement les blocs communs du fichier (en utilisant des sommes de contrôle).
- En revanche, `rsync` ne sait pas détecter les déplacements et les copies de fichiers ou de sous-arbres.
- Ainsi, un simple `mv baz/ bar/` oblige à retransférer tout le répertoire. Ce n'est pas satisfaisant !

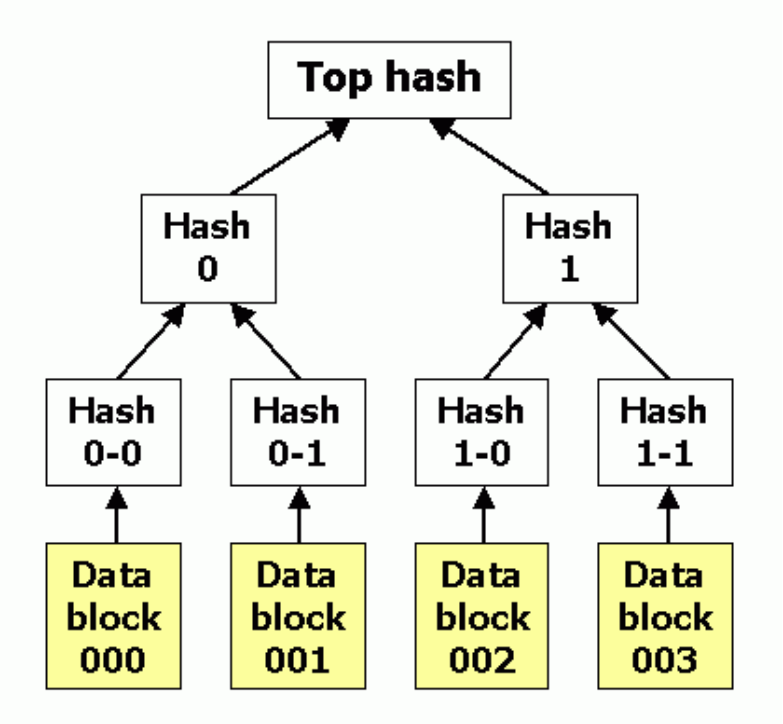
Votre mission

Votre mission, si vous l'acceptez :

Construire `btrsync`, un programme similaire à `rsync` (ou utilisant `rsync`) qui permet de transférer efficacement une arborescence en identifiant les sous-arbres communs.

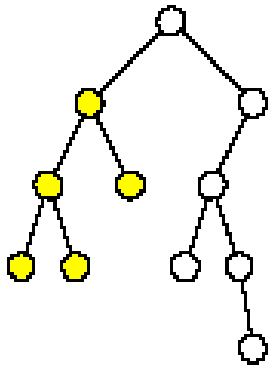
Approche suggérée

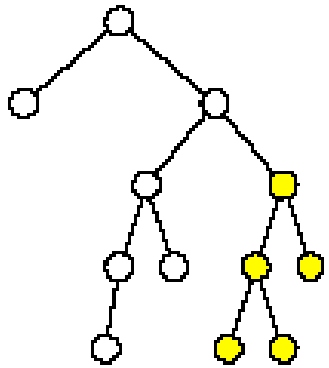
- Sur la source et sur la cible, calculer (récursivement) des sommes de contrôle pour les fichiers et les répertoires. (Par exemple, utiliser un arbre de Merkle.)
- Identifier les sommes de contrôle communes et déterminer ce qu'il faut transférer.



Idées pour l'implémentation

- Il y a peut-être moyen de détourner un logiciel existant plus compliqué pour faire ce travail (par exemple `git`).
- Sinon, il peut être intéressant de faire certains transferts et déplacements puis de rappeler `rsync` pour faire le reste du travail (et s'assurer que tout a bien marché).
- Pour gérer l'authentification et le chiffrement, ne rien implémenter soi-même, et utiliser `ssh` et `scp`.





Prolongements

- Laisser l'utilisateur choisir entre des sommes de contrôle sur les données (sûr, et détecte davantage de collisions, mais plus lent) ou sur les métadonnées comme le nom, la taille, etc. (peu sûr, mais suffisant dans certains cas et très rapide).
- Sérialiser les hashes calculés dans un fichier et le réutiliser au prochain transfert pour ne pas recalculer le hash de ce qui n'a pas été modifié.

Pourquoi c'est cool ?

- `rsync` est *vraiment* utilisé et *vraiment* pourri de ce point de vue.
- Si vous faites bien ce projet, des gens vont *vraiment* s'en servir !
- Si vous voulez chercher de la théorie autour, il y a sûrement des choses à trouver et à dire.