

Bases de données documentaires et distribuées
Cours NFE04
Réplication et reprise sur panne

Auteurs : Raphaël Fournier-S'niehotta, Philippe Rigaux, Nicolas Travers
prenom.nom@cnam.fr

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Répliquer, à quoi ça sert ?

Outil indispensable, universel pour la robustesse des systèmes distribués.

- **Tolérance aux pannes**

Vous cherchez un document sur S_1 , qui est en panne ? On le trouvera sur S_2

- **Distribution des lectures**

Répartissons les lectures sur S_1, S_2, \dots, S_n pour satisfaire les millions de requêtes de nos clients.

- **Distribution des écritures ?**

Oui, mais attention, il faut **réconcilier** les données ensuite.

- et autres avantages, comme la construction d'un index sur un des serveurs sans affecter les autres,

Le bon niveau de réplication ? **Trois copies pour une sécurité totale ; deux au minimum.**

Méthode générale de réplication

Une **application** (le client) demande au **système** (le serveur) S_1 **l'écriture** d'un document (unité d'information).

- le serveur écrit le document sur le disque ;
- S_1 **transmet la demande d'écriture à un ou plusieurs autres serveurs** S_2, \dots, S_n , créant des **replicas** ou **copies** ;
- S_1 "rend la main" au client.

Essentiel, réplication synchrone/asynchrone ?

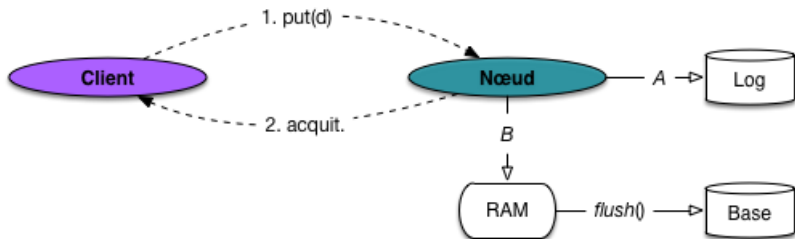
- **synchrone** : S_1 **attend** confirmation de S_2, \dots, S_n avant de rendre la main ;
- **asynchrone** : S_1 rend la main **sans attendre**.

Conséquences

Une réplication **asynchrone** est beaucoup plus rapide, mais elle favorise les **incohérences**, au moins temporaires.

Pour éviter la latence disque : le fichier journal

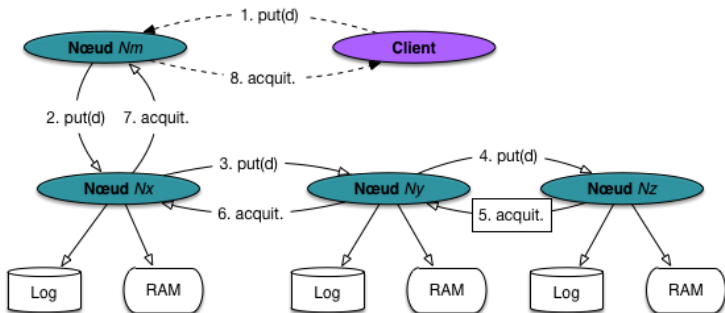
Technique universellement utilisée en centralisé



Permet de se ramener à des entrées/sorties **séquentielles**.

Réplication avec écritures synchrones

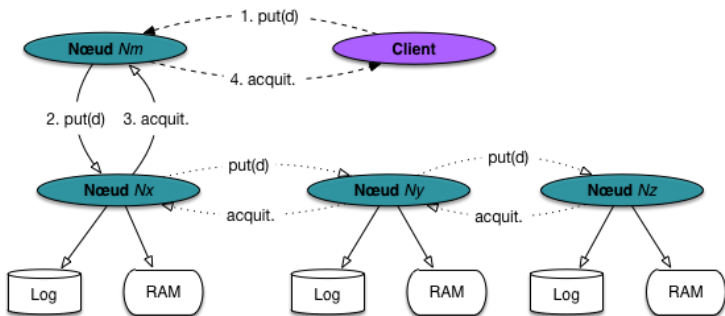
Le client est acquitté quand **tous** les serveurs ont effectué l'écriture.



Sécurité totale ; cohérence forte ; **très lent**.

Réplication avec écritures asynchrones

Le client est acquitté quand **un** des serveurs a effectué l'écriture. Les autres écritures se font indépendamment.



Sécurité partielle ; cohérence faible ; **Efficace**.

Parlons cohérence

La cohérence est la capacité d'un système de gestion de données à refléter fidèlement les opérations d'une application.

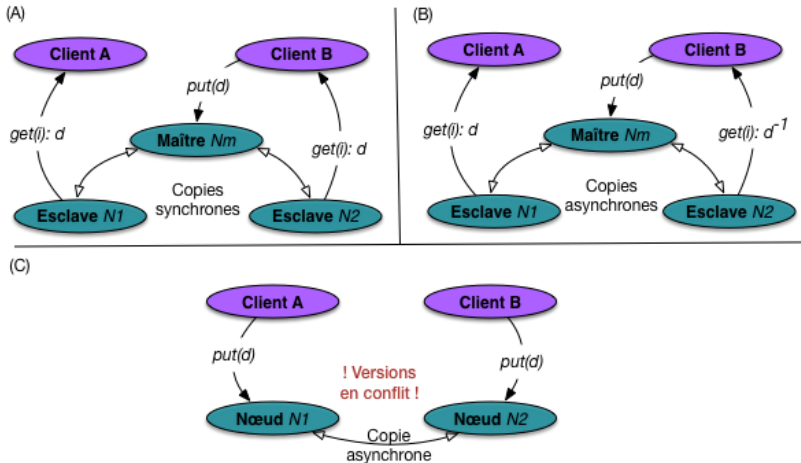
- toute opération (validée) est immédiatement visible et permanente.
- si je fais une écriture de d suivie d'une lecture, je dois constater les modifications effectuées ;
- si je refais une lecture un peu plus tard, ces modifications doivent toujours être présentes.

Cohérence forte = une des fonctionnalités marquantes des systèmes relationnels.

Dans les systèmes NoSQL

La cohérence forte est sacrifiée au profit des performances.

Architectures avec réplication : topologie et (a)synchronicité



Soyons cohérents (ou pas...)

Plusieurs niveaux de cohérence dans les systèmes distribués / NoSQL.

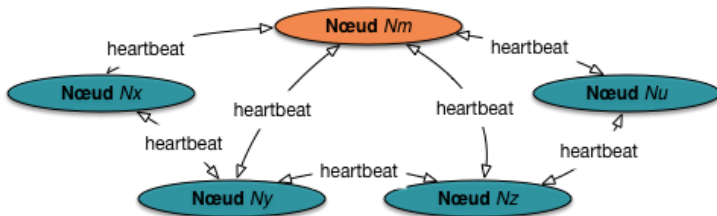
- **Cohérence forte** (ACID) – nécessite une réplication synchrone, et potentiellement des verrouillages complexes (*two phases commit*).
- **Cohérence faible** – on accepte le risque de lectures ne reflétant pas les mises à jour.
- **Cohérence à terme** (*Eventual consistency*) – le système garantit que les incohérences ne sont que transitoires.

Remarque

Les systèmes NoSQL dans l'ensemble abandonnent la cohérence forte pour favoriser la réplication asynchrone, et donc le débit en écriture/lecture.

Réplication et reprise sur panne

Principe général : tout le monde est interconnecté et échange des messages (*heartbeat*).

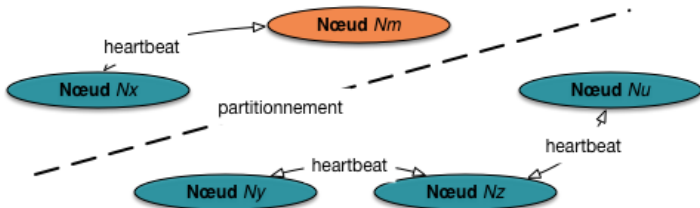


Si un **esclave** tombe en panne, le système continue à fonctionner, **tant qu'il est en contact avec une majorité d'esclaves**

Si le **maître** tombe en panne, les **esclaves** doivent **élire** un nouveau maître.

Cas du partitionnement réseau

Un **partitionnement** divise la grappe en deux groupes.



Principe de majorité

Seul le groupe représentant la **majorité** des participants initiaux peut élire un maître.

Election : algorithme de négociation, voir par exemple Paxos.

Culture, culture : le théorème CAP

Un “théorème” qui dit qu’un système distribué ne peut satisfaire à la fois

- 1 La cohérence (**C**)
- 2 La disponibilité (**A** *Availability*)
- 3 La tolérance au **P**artitionnement

Exprime une limitation intrinsèque aux systèmes distribués, applicable à tout système NoSQL, et expliquant qu’il faut toujours faire un compromis.