Les BD NOSQL
Réplication

-La **réplication** (des données) est une caractéristique commune aux systèmes NoSQL.

- s'exécutent dans un environnement sujet à des pannes fréquentes et répétées.

-Il est donc indispensable, pour assurer la **sécurité** des données, de les **répliquer**

Replication avec MongoDB, ElasticSearch et Cassandra…

Répliquer, à quoi ça sert ?

Outil indispensable, universel pour la robustesse des systèmes distribués.

- Tolérance aux pannes Vous cherchez un document sur **S1**, qui est en panne ? On le trouvera sur **S2**
- Distribution des lectures Répartissons les lectures sur **S1, S2, . . . , Sn** pour satisfaire les millions de requêtes de nos clients.
- Distribution des écritures ? Oui, mais attention, il faut réconcilier les données ensuite.

Le **niveau de réplication** dépend notamment du budget qu'on est prêt à allouer à la sécurité des données.

On peut considérer que **3 copies** constituent un bon niveau de sécurité.

Ex : Une stratégie possible est par exemple de placer un document sur un serveur dans une baie, une copie dans une autre baie pour qu'elle reste accessible en cas de coupure réseau, et une troisième dans un autre centre de données

Méthode générale de réplication

Une application (le client) demande au système (le serveur) S1 l'écriture d'un document (unité d'information).

- le serveur écrit le document sur le disque ;
- S1 transmet la demande d'écriture à un ou plusieurs autres serveurs S2, . . . , Sn, créant des replicas ou copies ;
- S1 "rend la main" au client.

Temps d’attente

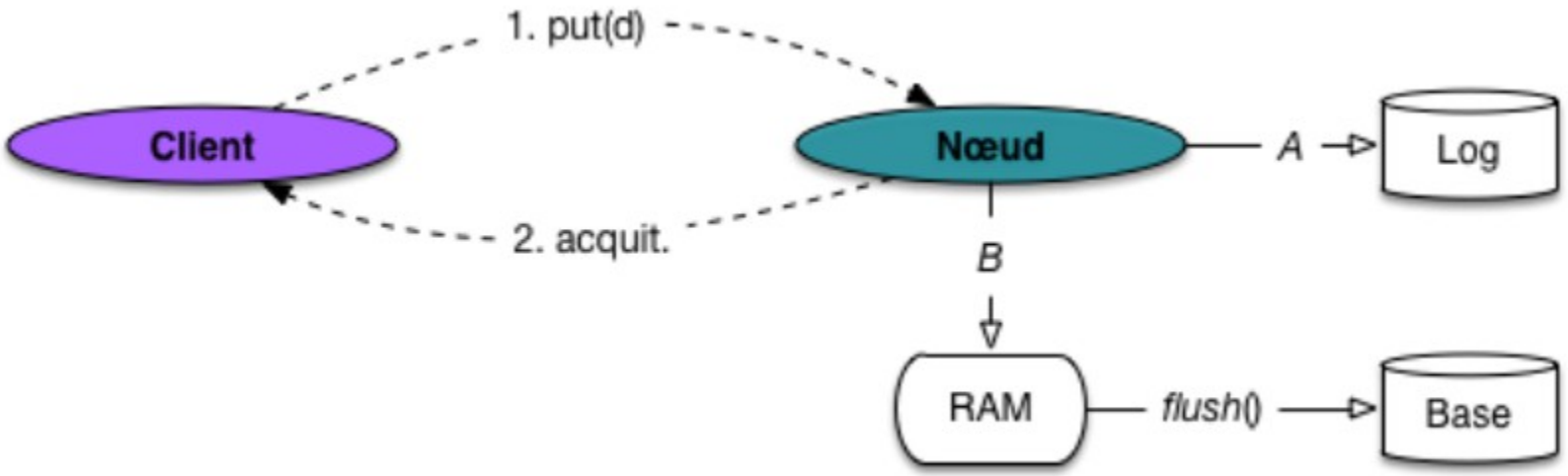
Deux techniques sont utilisées pour limiter le temps d’attente, toutes deux affectant (un peu) la sécurité des opérations:

- **écriture en mémoire RAM**, et fichier journal (*log*);
- **réplication asynchrone**.

La première technique

Pour éviter la latence disque : le fichier journal

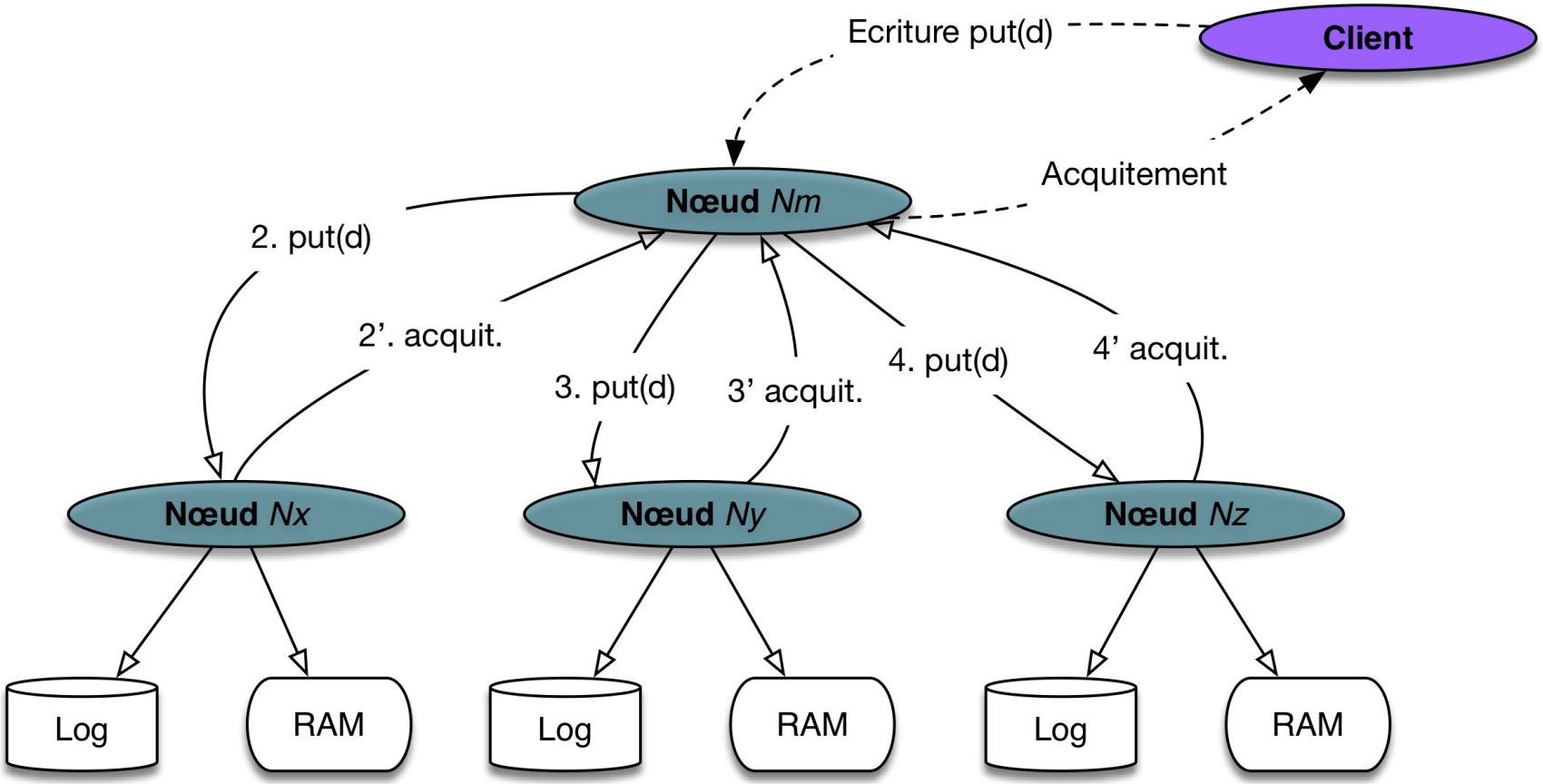
Technique universellement utilisée en centralisé



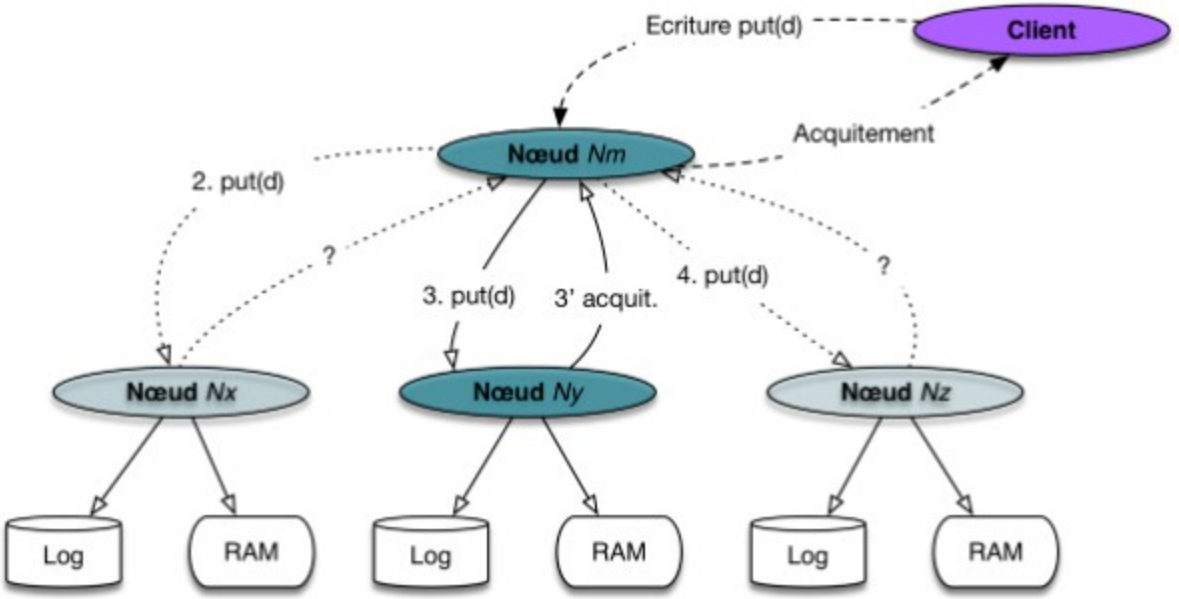
Réplication (avec écritures synchrones)

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.

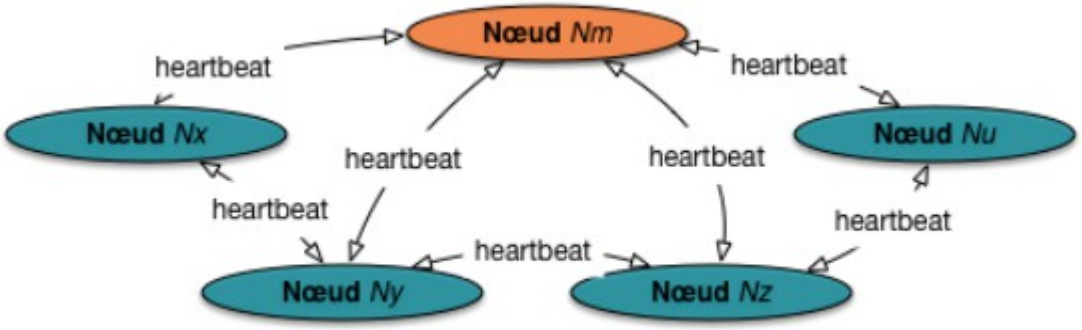
- **Coherence (Consistency)**:les modifications apportées à la b ase doivent
  être valides, en accord avec l'ensemble de la base et de ses contraintes d'intégrité. Par conséquent, une transaction fait passer une base de données d'un état cohérent à un autre état cohérent. En cas d'échec, l'état cohérent initial doit être restauré.

-Cohérence forte = une des fonctionnalités marquantes des systèmes relationnels (ACID).

-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.

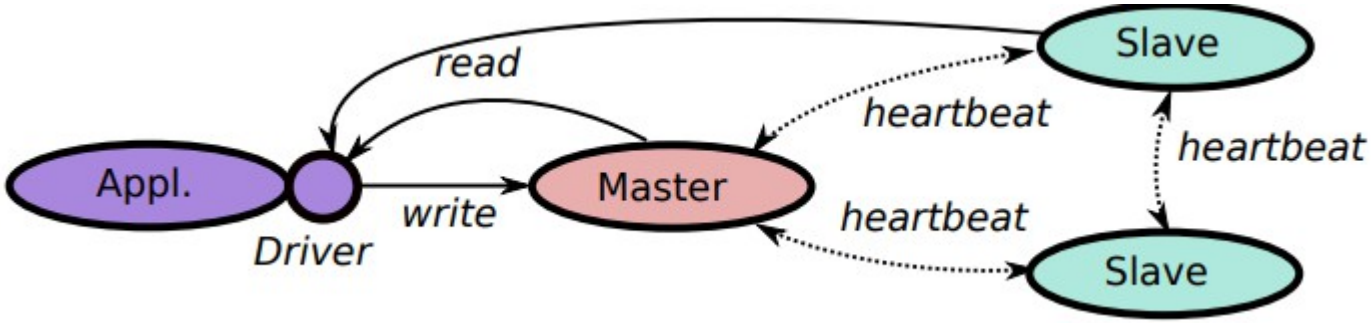
Si le maître tombe en panne, les esclaves doivent élire un nouveau maître.

Réplication dans MongoDB¶

MongoDB présente un cas très représentatif de gestion de la réplication et des reprises sur panne.

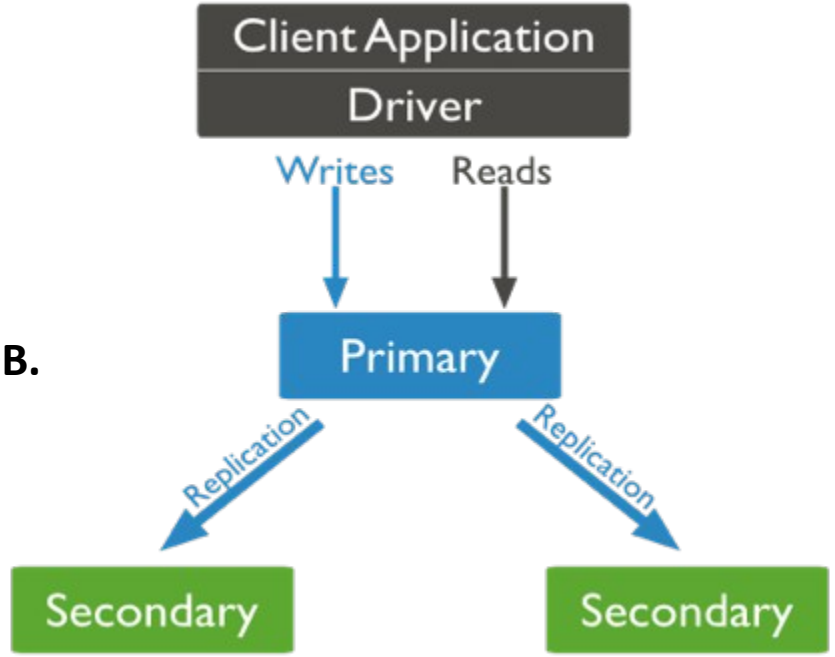
Réplica en Mongodb : Réplica set

Une grappe de serveurs partageant des copies d'un même ensemble de documents est appelée un ***replicat set*** (RS) dans MongoDB.



- Le **RS** comprend un maître (**primary**) et des esclaves (secondary)
- Les écritures ont toujours lieu sur le maître
- La réplication vers les deux esclaves se fait en mode **asynchrone**.
- Le **driver** associé à l'application peut lire sur le maître (cohérence forte) ou sur un secondary (cohérence faible)

Réplication :



Un Replica Set permet de gérer :
la tolérance aux pannes à l'intérieur de MongoDB.

- Si nombre **pair** : ajout d'un processus supplémentaire (arbiter)
- Un arbitre permet une majorité absolue en cas de partitionnement

Election d’un maître dans MongoDB

Une élection est déclenchée dans les cas suivants :

• Initialisation d’un nouveau RS

• Un esclave perd le contact avec le maître

• Le maître perd son statut car il ne voit plus qu’une minorité de participants

• Tous les clients (drivers) connectés au RS sont réinitialisés pour dialoguer avec le nouveau maître.

On crée trois conteneurs avec des serveurs MongoDB

```
docker run --name mongo1 --net host mongo mongod --replSet mon-rs --port 30001
docker run --name mongo2 --net host mongo mongod --replSet mon-rs --port 30002
docker run --name mongo3 --net host mongo mongod --replSet mon-rs --port 30003
```

Connectons un client au premier serveur (ou Studio3D si vous voulez).

```
mongo --host 192.168.99.100 --port 30001
```

Et initialisons le RS

```
rs.initiate()
```

On peut alors ajouter le second nœud (remplacer l'IP de la machine).

```
rs.add ("192.168.99.100:30002")
```

Pour savoir quel nœud a été élu maître :

db.isMaster()

Et pour tout savoir sur le *replica set*:

rs.status()

mongoimport -d nfe204 -c movies --file movies.json --jsonArray --host <hostIP> --port <xxx

on peut supposer que la réplication s'est effectuée. Vérifions:
connectez-vous au maître et regardez le contenu de la collection movies.

- http://docs.mongodb.org/manual
- http://camillepradel.fr/

http://b3d.bdpedia.fr/