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 pour assurer la survie d'au moins une copie en cas d'accident grave (incendie, tremblement de terre).

À défaut d'une solution aussi complète, **deux copies** constituent déjà une bonne protection. Il est bien clair que dès que la perte de l'une des copies est constatée, une nouvelle réplication doit être mise en route.

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) 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.

Essentiel, réplication synchrone/asynchrone ?

- **synchrone** : S1 attend confirmation de S2, . . . , Sn avant de rendre la main ;
- **asynchrone** : S1 rend la main sans attendre.

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

Temps d’attente

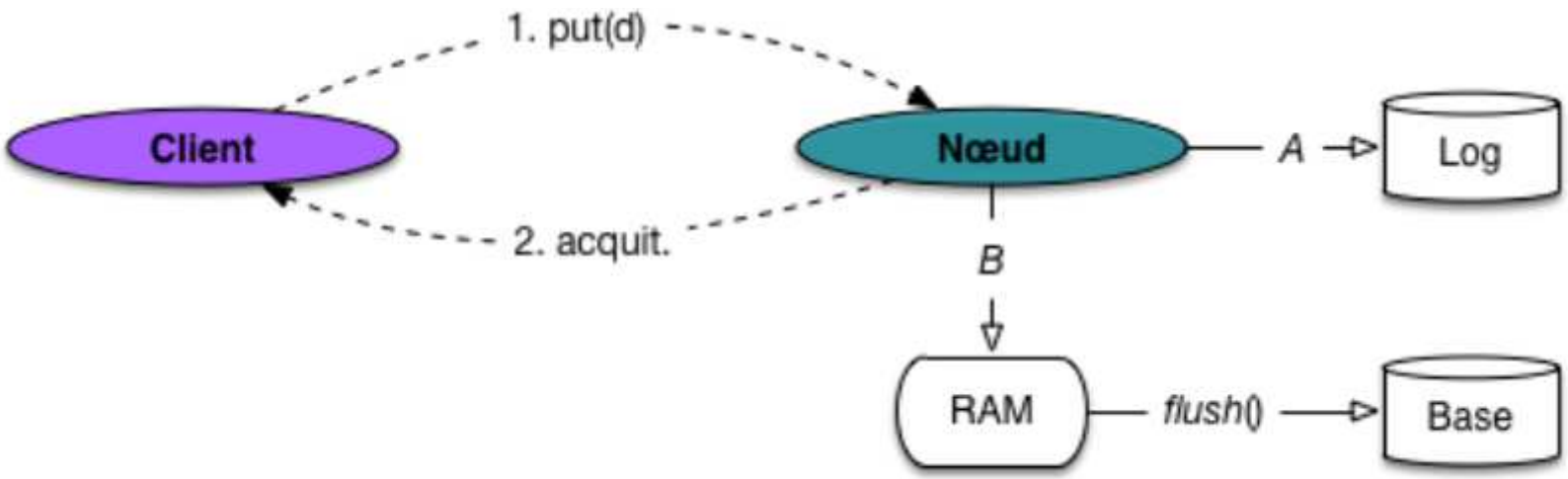
Deux techniques sont utilisées pour limiter le temps d’attente, toutes deux affectant (un peu) la sécurité des opérations:

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

La première technique

Pour éviter la latence disque : le fichier journal

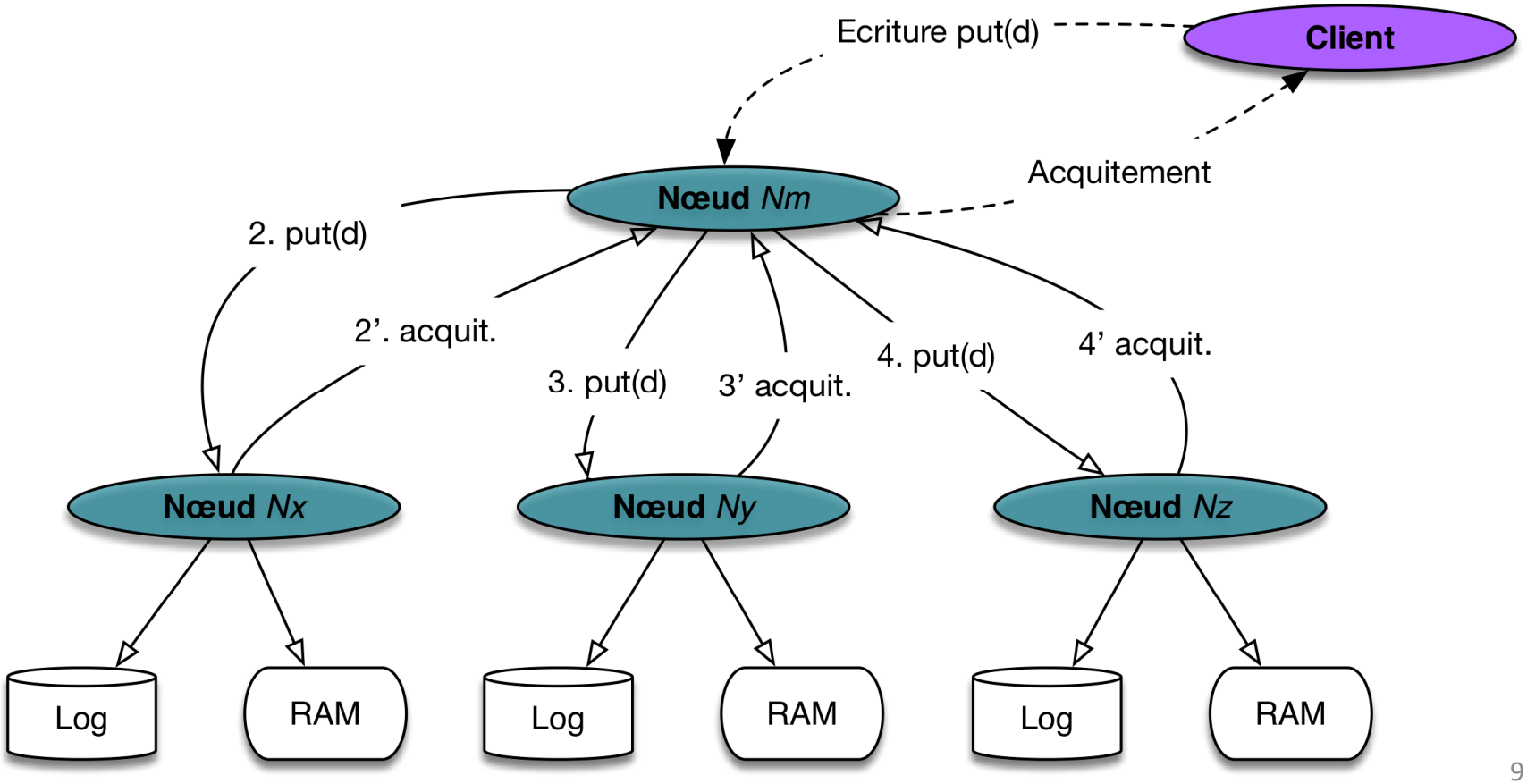
Technique universellement utilisée en centralisé



Réplication (avec écritures synchrones)

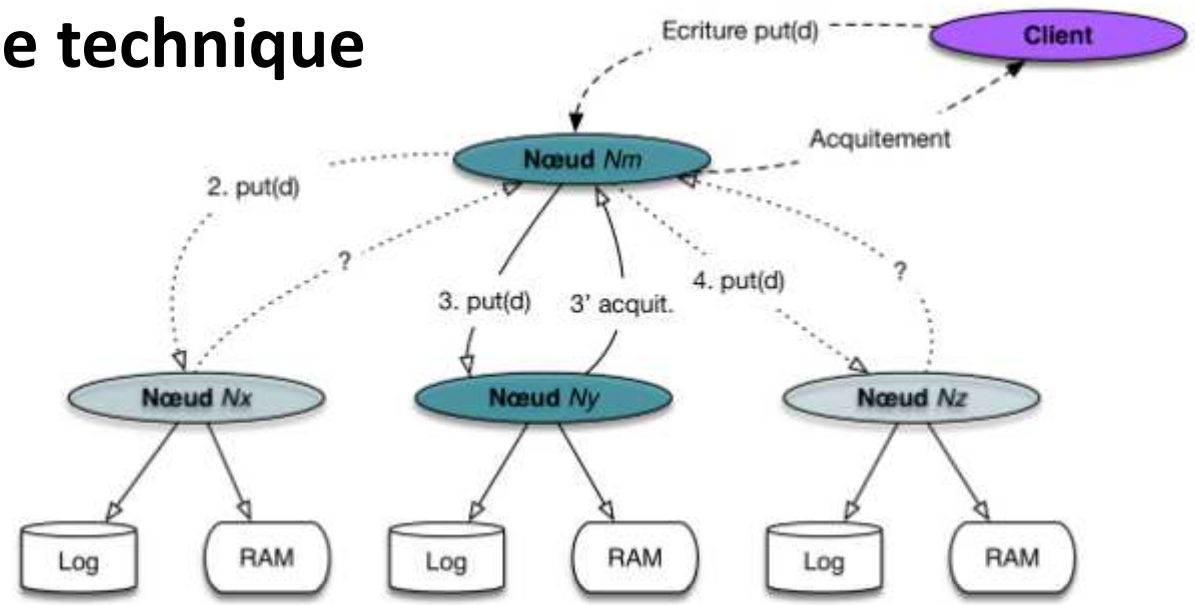
Le client est acquitté quand tous les serveurs ont effectué l'écriture

Sécurité totale ; cohérence forte ; très lent



Réplication avec écritures asynchrones

La 2eme technique



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

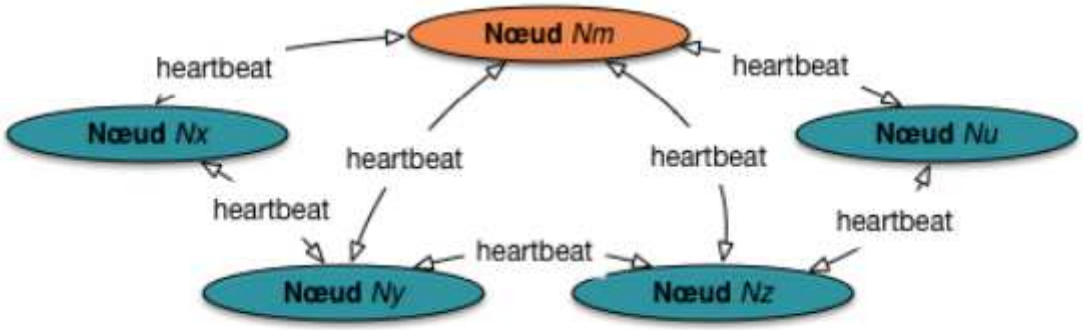
-Sécurité partielle ; cohérence faible ; Efficace.

-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.

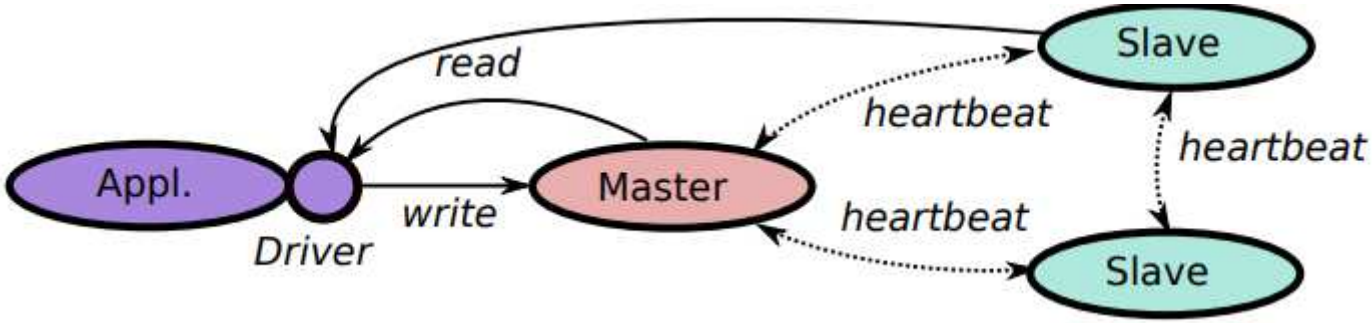
Si le maître tombe en panne, les esclaves doivent élire un nouveau maître.

Réplication dans MongoDB

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

Replica en Mongodb : Replica set

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

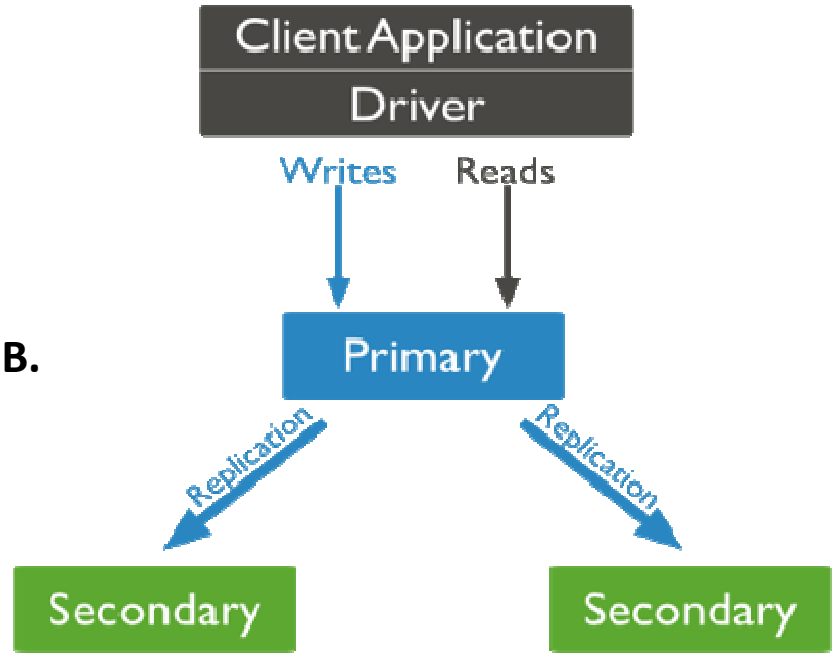


- Le **RS** comprend un maître (**primary**) et des esclaves (secondary)
- Les écritures ont toujours lieu sur le maître
- La réplication vers les deux esclaves se fait en mode **asynchrone**.
-Le **driver** associé à l'application peut lire sur le maître (cohérence forte) ou sur un secondary (cohérence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

Un RS contient typiquement trois nœuds, un maître et deux esclaves. C'est un niveau de réplication suffisant pour assurer une sécurité presque totale des données.

Réplication :

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



-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.

Commande Mongodb replicat set

```
> rs.help()
        rs.status()                                { replSetGetStatus : 1 } checks repl set status
        rs.initiate()                              { replSetInitiate : null } initiates set with default settings
        rs.initiate(cfg)                           { replSetInitiate : cfg } initiates set with configuration cfg
        rs.conf()                                  get the current configuration object from local.system.replset
        rs.reconfig(cfg)                           updates the configuration of a running replica set with cfg (disconne
cts)
        rs.add(hostportstr)                        add a new member to the set with default attributes (disconnects)
        rs.add(membercfgobj)                       add a new member to the set with extra attributes (disconnects)
        rs.addArb(hostportstr)                     add a new member which is arbiterOnly:true (disconnects)
        rs.stepDown([stepdownSecs, catchUpSecs])   step down as primary (disconnects)
        rs.syncFrom(hostportstr)                   make a secondary sync from the given member
        rs.freeze(secs)                            make a node ineligible to become primary for the time specified
        rs.remove(hostportstr)                     remove a host from the replica set (disconnects)
        rs.secondaryOk()                               allow queries on secondary nodes

        rs.printReplicationInfo()                  check oplog size and time range
        rs.printSecondaryReplicationInfo()             check replica set members and replication lag
        db.isMaster()                              check who is primary

        reconfiguration helpers disconnect from the database so the shell will display
        an error, even if the command succeeds.
>
```

Chaque nœud est un serveur mongod
Les serveurs doivent pouvoir communiquer entre eux par le réseau:

-Pour simuler un ReplicaSet localement avec un répertoire dédié

mongod --replSet *nomRS* **--port** *numprt*-**-dbpath** *chemin/repertoire*

-Pour connecter un client au serveur

mongo --host @IP --port numport

-Pour Initialiser le *replica set*, et ajoutez-lui le client.

```
rs.initiate()
rs.add (« adresseIP:port")
```

-Puis vérifiez l'état de votre replica set :

```
rs.status()
```

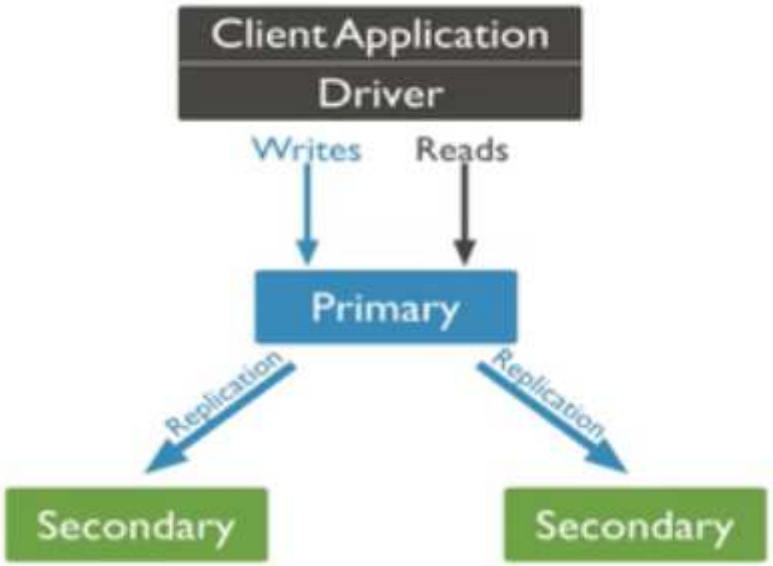
-Pour savoir quel nœud a été élu maître :

```
db.isMaster()
```

- insérer des données, qui devraient alors être répliquées.

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

Exemple d application



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