Université d'Évry-Val d'Essonne

université PARIS-SACLAY

Master 1 Informatique & Miage

Bases de Données Approfondies – 2e partie

Théorie et pratique de NoSQL

Philippe Declercq

2022/2023

Plan du cours et des TD

1	Introduction au NoSQL
2	Stockage et manipulation de données avec NoSQL
3	Réplication des données
4	Partitionnement des données
5	Recherches et calculs en NoSQL
6	Et l'avenir ?
TD1	Installation et manipulations simples NoSQL
TD2	Réplication et partitionnement
TD3	Recherche d'information – Moteurs de recherche

Université d'Évry-Val d'Essonne

université PARIS-SACLAY

Théorie et pratique de NoSQL

Chapitre 3

Réplication des données

Philippe Declercq

2022/2023

Réplication des données

- Scalabilité : aptitude d'un système à maintenir un même niveau de performance face à l'augmentation de charge ou de volumétrie, par augmentation des ressources matérielles et sans impact global sur le système.
- Scalabilité horizontale (ou externe) : possibilité d'ajouter des machines identiques en //.
- Scalabilité verticale (ou interne) : possibilité d'ajouter des ressources supplémentaire à une machine (CPU, RAM, disque…).
- Qu'est ce que la scalabilité pour les bases de données ?

Réplication des données

- Système distribué :
  - système logiciel qui permet de coordonner de nombreuses machines
  - souvent dans un même réseau local (LAN)
  - communiquant par l'échange de messages
  - avec des machines peu spécialisées pouvant être retirées ou ajoutées
- Pour les données distribuées :
  - un cas particulier de système distribué
  - accès efficaces avec des volumes de données très importants
  - accès même en cas d'indisponibilité des machines
  - possibilités d'extension (infinie !) des capacités de stockage et des volumétries d'accès
- Les systèmes NoSQL sont des systèmes distribués dédiés à la gestion de grandes masses de données.

Réplication des données

- En tant que système distribué, un système NoSQL va pouvoir mettre en œuvre la réplication des données qu'il stocke.
- Réplication : processus de partage d'informations pour assurer la cohérence de données entre plusieurs sources de données redondantes, pour améliorer la fiabilité, la tolérance aux pannes, ou la disponibilité. On parle de réplication de données si les mêmes données sont dupliquées sur plusieurs périphériques. *(source : wikipedia)*
- La réplication repose sur un ensemble de technologies qui permettent de copier et de distribuer des données et des objets de base de données d'une base vers une autre, puis de synchroniser ces bases afin de préserver leur cohérence.

Réplication des données

- Les systèmes relationnels peuvent aussi mettre en œuvre la réplication de données, cependant leurs caractéristiques transactionnelles leur impose le respect des contraintes ACID :

Réplication des données

- Les critères ACID :
  - **Atomique** : une transaction représente une unité de travail qui est validée intégralement ou totalement annulée. C'est tout ou rien.
  - **Cohérente** : la transaction doit maintenir le système en cohérence par rapport à ses règles fonctionnelles. Durant l'exécution de la transaction, le système peut être temporairement incohérent, mais lorsque la transaction se termine, il doit être cohérent, soit dans un nouvel état si la transaction est validée, soit dans l'état cohérent antérieur si la transaction est annulée.
  - **Isolée** : comme la transaction met temporairement les données qu'elle manipule dans un état incohérent, elle isole ces données des autres transactions de façon à ce qu'elle ne puisse pas lire des données en cours de modification.
  - **Durable** : lorsque la transaction est validée, le nouvel état est durablement inscrit dans le système.

Réplication des données

- Ces contraintes sont difficiles à maintenir dans un système distribué tout en conservant des performances correctes.
- En particulier la notion de cohérence : les systèmes relationnels imposent une règle stricte : d'un point de vue transactionnel, les lectures de données se feront toujours sur des données cohérentes. La base de données est visible en permanence sur un état cohérent.
- Cette propriété est facteur d'augmentation des temps de réponse…surtout si des données sont répliquées…
- La cohérence est cependant nécessaire, et même indispensable, dans certains cas. Mais pas dans tous…

Réplication des données

- On oppose parfois les propriétés ACID des SGBDR aux propriétés « BASE » des solutions NoSQL :
  - **Basic Availability** : le système garantit la disponibilité des données
  - **Soft-state** : L'état du système peut changer avec le temps. Ainsi, même pendant les périodes sans insertion ou mise à jour, des changements peuvent survenir en raison de traitements asynchrones exécutés pour la mise en cohérence du système.
  - **Eventual consistency** : la cohérence des données n'est pas assurée immédiatement, mais elle arrivera avec le temps.
- Ces contraintes, qui peuvent ne pas sembler suffisantes pour un système de persistance, permettent une bien meilleure disponibilité et scalabilité que les contraintes ACID.

Réplication des données

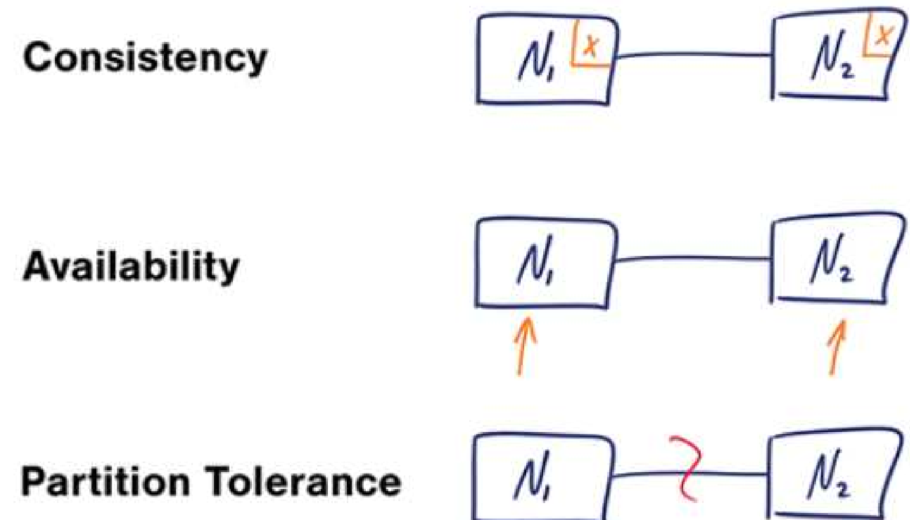
- Le théorème de CAP (Brewer, 2000) stipule qu'il est impossible, dans un système distribué, d'assurer les 3 propriétés suivantes en même temps :
  - **C comme Coherence ou Cohérence** : tous les nœuds du système voient exactement les mêmes données au même moment.

    Si j'écris une donnée dans un nœud et que je la lis dans un autre nœud, je dois retrouver ce que j'ai écrit sur le premier.
  - **A comme Availability ou Disponibilité** : la perte de nœuds n'empêche pas les survivants de continuer à fonctionner correctement.

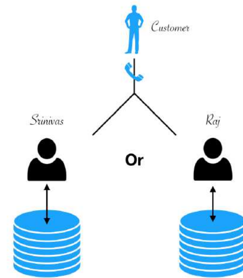
    Même en cas de panne, mes données restent accessibles.
  - **P comme Partition tolerance ou Résistance au partitionnement** : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement.

    Les données peuvent être partitionnées sans soucis de localisation.

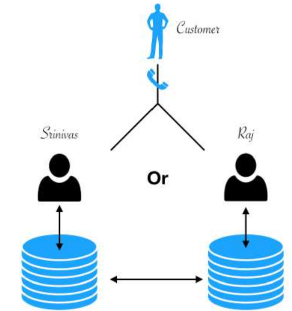
Réplication des données



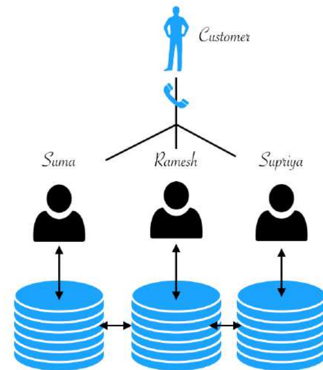
- Illustration du théorème de CAP
  - Imaginons une entreprise de livraison de repas à domicile.
  - Cette entreprise prend ses commandes par téléphone uniquement.
  - Actuellement, un seul employé répond au téléphone pour enregistrer les commandes. Mais… certains clients se plaignent d'attendre de longues minutes avant que leur appel aboutisse….
  - L'entreprise décide alors d'affecter 2 employés au standard téléphonique. L'entreprise espère avec cette solution perdre moins de clients en répondant plus rapidement à leur appel.
  - « Et s'il le faut, j'affecterai autant de personnes que nécessaire au standard pour qu'un client n'attende jamais plus de 1 minute ! » s'exclame le patron.

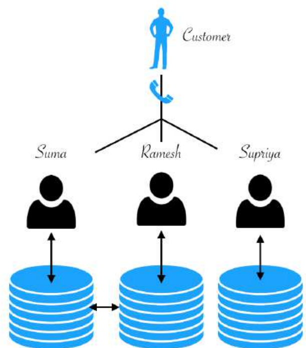


- Un beau jour, un client, qui vient de passer une commande auprès d'un employé, appelle quelques minutes plus tard pour savoir où en est sa commande.
- C'est un autre employé qui prend l'appel, et qui ne trouve pas trace de la commande sur son propre cahier….
- L'employé est alors obligé d'aller demander à ses collègues qui a connaissance de la commande….cela prend beaucoup de temps, et au final le client, furieux, décide d'annuler sa commande…
- Le patron, mécontent, impose alors aux employés de fournir une copie de chaque commande passée aux autres employés au standard téléphonique. Cela va couter du temps, mais au moins, chaque employé aura connaissance de l'ensemble des commandes des clients.



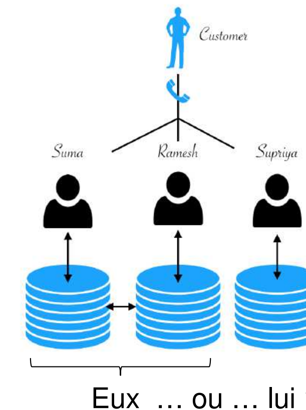
- L'organisation mise en place fonctionne bien, et devant l'augmentation des appels, le patron décide d'augmenter le nombre d'employés au standard.





- Mais après quelques temps….des tensions apparaissent dans l'équipe, et certains employés sont mis à l'écart et ne communiquent plus avec les autres….

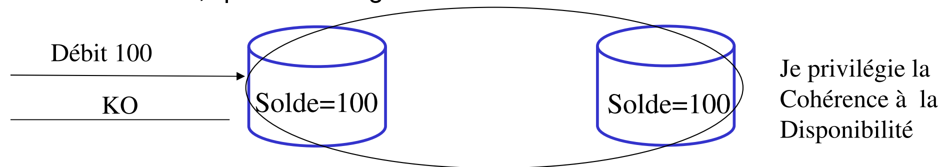
- La patron s'interroge : si les personnes ne peuvent plus se parler, comment faire pour conserver une organisation satisfaisante ?
- Après réflexion, le seul moyen qu'il trouve est de se séparer d'une partie des employés pour ne conserver que ceux qui acceptent de communiquer entre eux ! Et tant pis si le temps d'attente des clients augmente …



Eux … ou … lui ?

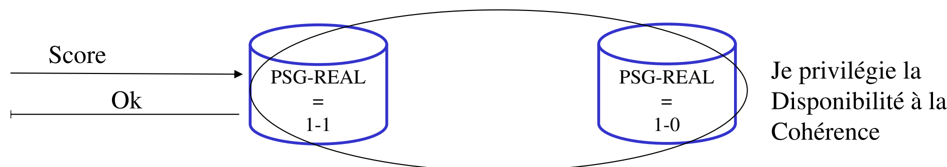
Réplication des données

- Dans le cadre d'un compte en banque sur lequel je peux faire des débits, quelle stratégie mettre en œuvre ?



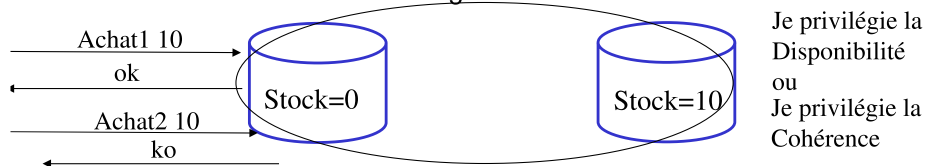
Je privilégie la Cohérence à la Disponibilité

- Et dans le cas d'un match de foot ?



Je privilégie la Disponibilité à la Cohérence

- Et dans le commerce en ligne ?



Je privilégie la Disponibilité
ou
Je privilégie la Cohérence

Réplication des données

- **Consistance (Coherence)** : Un système S est dit consistant si on peut garantir qu'une fois qu'on y enregistre un état (disons "x = y"), il donnera le même état à chaque opération suivante, jusqu'à ce que cet état soit explicitement changé par quelque chose en dehors de S.

Réplication des données

- Exemples de consistance :
  - Exemple 1 : une instance de la BD est automatiquement pleinement consistante puisqu'il n'y a qu'un seul nœud qui maintient l'état.
  - Exemple 2 : si 2 serveurs de BD sont impliqués, et si le système est conçu de telle sorte que toutes les clés de "a" à "m" sont conservées sur le serveur 1, et les clés "n" à "z" sont conservées sur le serveur 2, alors le système peut encore facilement garantir la cohérence.
  - Exemple 3 : Supposons 2 BD avec des répliques. Si une des BD fait une opération d'insertion de ligne, cette opération doit être aussi faite (commited) dans la seconde BD avant que l'opération soit considérée comme complète. Pour avoir une cohérence de 100% dans un tel environnement répliqué, les noeuds doivent communiquer, et plus le nb de répliques est grand plus les performances d'un tel système sont mauvaises.

Réplication des données

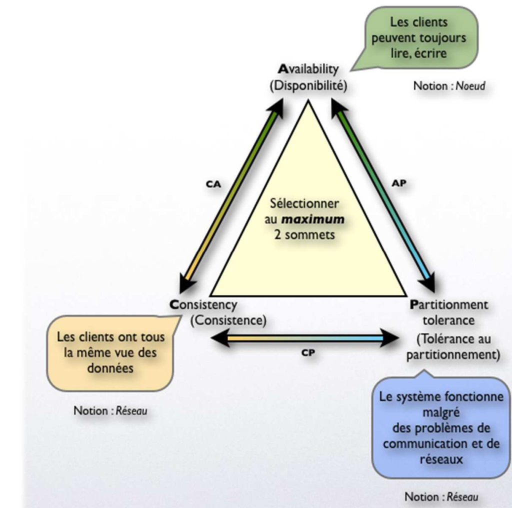
- **Disponibilité (Availability)**: Les BD dans Ex1 ou Ex2 ne sont pas fortement disponibles :
  - dans Ex1 : si le nœud tombe en panne, on perd 100% des données
  - dans Ex2 : si un nœud tombe en panne, on perd 50% des données
  - dans Ex3 : la réplication de la BD sur un autre serveur assure une bien plus forte probabilité que le système soit disponible.
- Donc :
  - Augmenter le nb de nœuds avec des copies des données (répliques) augmente directement la disponibilité du système, en le protégeant contre les défaillances matérielles
  - les répliques peuvent aussi aider à équilibrer la charge d'opérations concurrences, notamment en lecture.

Réplication des données

- **Résistance au partitionnement (Partition-tolerance)** :
  - Supposons que soit assuré par réplication « consistance » et « disponibilité ». Dans le cas Ex3, supposons 2 serveurs de BD dans 2 Data-Centers différents, et que l'on perde la connexion réseau entre les 2 Data-Centers, faisant que les 2 BD sont incapables de synchroniser leurs états.
  - Si vous parvenez à gérer les opérations de lecture/écriture sur ces 2 BD, il peut être prouvé que les 2 serveurs ne seront plus consistants.
  - Une application bancaire gardant à tout moment «l'état de votre compte » est l'exemple parfait du problème des enregistrements inconsistants : si un client retire 1000 euros à Marseille, cela doit être immédiatement répercuté à Paris, afin que le système sache exactement combien il peut retirer à tout moment.
  - Si les banques décident que la « consistance » est très importante, et désactivent les opérations d'écriture lors de la panne, alors la «disponibilité» du cluster sera perdue puisque tous les comptes bancaires dans les 2 villes seront désormais gelés jusqu'à ce que le réseau soit de nouveau opérationnel.

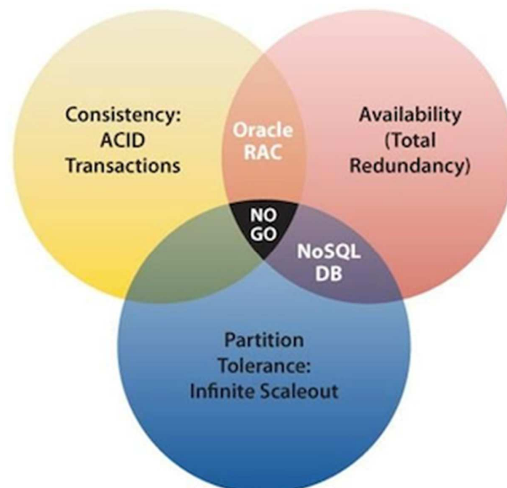
Réplication des données

- Pour un système de données distribué, il faut donc choisir et abandonner l'une des trois propriétés … oui mais … laquelle abandonner ? …



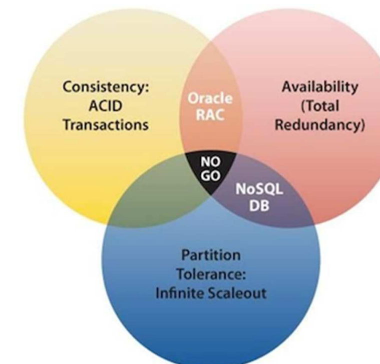
Réplication des données

- Les systèmes relationnels assurent les propriétés C(onsistency) + (A)vailability en vérifiant les propriétés ACID.



Réplication des données

- L'émergence des nouveaux besoins liés au big data a fait naitre des systèmes de gestion des données conçus de telles sortes qu'ils assurent les propriétés (A)vailability + (P)artition Tolerance, tant pis pour la (C)ohérence ….

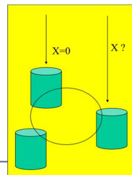


- C'est une des caractéristiques des systèmes NoSQL.

Réplication des données

- Trois niveaux de cohérence peuvent se rencontrer dans les systèmes de gestion des données :
  1. **cohérence forte** : toutes les copies sont toujours en phase, le prix à payer étant un délai pour chaque écriture.
  2. **cohérence faible** : les copies ne sont pas forcément en phase, et rien ne garantit qu'elles le seront.
  3. **cohérence à terme** : c'est le niveau de cohérence typique des systèmes NoSQL : les copies ne sont pas immédiatement en phase, mais le système garantit qu'elles le seront « à terme ».

- Dans la cohérence à terme, il existe donc un risque, faible mais réel, de constater de temps en temps un décalage entre une écriture et une lecture !



Réplication des données

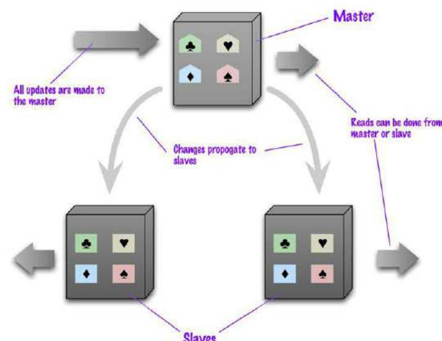
- Les systèmes NoSQL vont permettre un distribution des données et la mise en œuvre de mécanismes de réplication.

- Réplication : capacité à maintenir à jour une base de données distribuée sur plusieurs machines reliées en réseau, en recopiant à intervalles réguliers des morceaux ou l'intégralité de la base d'une machine à l'autre.

- Il existe 2 types principaux de réplication :
  - Réplication maitre-esclave (master slave replication),
  - Réplication multi-nœuds (peer-to-peer replication).

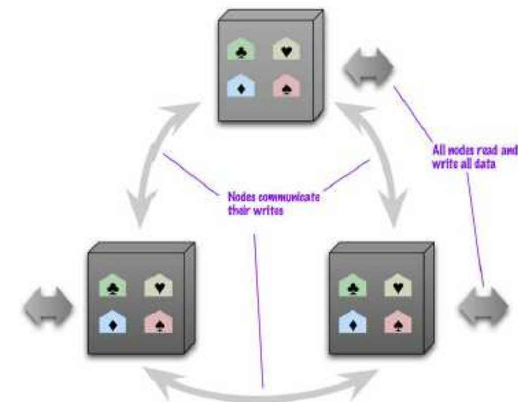
Réplication des données

- Réplication maitre-esclave (master slave replication) :
  - Le maître :
    - Est responsable des mises à jour des données,
    - Peut être désigné manuellement ou automatiquement.
  - Les esclaves :
    - Le processus de réplication les synchronise avec le maître,
    - Lorsque le maître tombe en panne, un esclave peut devenir maître.



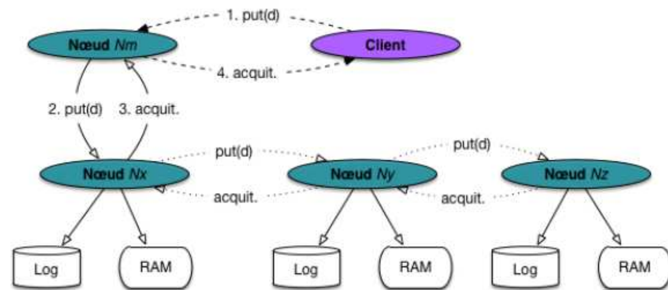
Réplication des données

- Réplication multi-nœuds (peer-to-peer replication) :
  - Tous les nœuds sont égaux, ils peuvent accepter les écritures,
  - La scalabilité est meilleure.



Réplication des données

- La réplication dans les systèmes NoSQL est en général asynchrone :



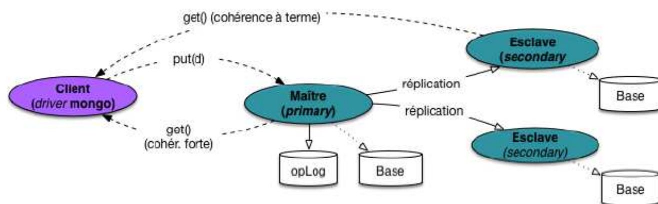
- Le client reçoit un acquittement alors que la réplication n'est pas complète. Pas de garantie complète de sécurité !
- Le client poursuit son exécution alors que toutes les copies ne sont pas encore mises à jour. Il se peut alors qu'une lecture renvoie une des copies obsolètes.

Réplication des données

- Mais … répliquer les données, quel(s) intérêt(s) ?
- Réponse : pour assurer la robustesse du système !
- La réplication de données dans les systèmes NoSQL permet d'atteindre les propriétés A et P du théorème de CAP :
  - Disponibilité ou Tolérance aux pannes. La réplication permet d'assurer la disponibilité constante du système. En cas de panne d'un serveur, d'un nœud ou d'un disque, la tâche effectuée par le composant défectueux peut être immédiatement prise en charge par un autre.
  - Scalabilité (lecture). Si une donnée est disponible sur plusieurs machines, il devient possible de distribuer les requêtes (en lecture) sur ces machines.
  - Scalabilité (écriture). On peut penser à distribuer aussi les requêtes en écriture, mais là on se retrouve face à de délicats problèmes potentiels d'écritures concurrentes et de réconciliation.

Réplication des données – En pratique sous MongoDB

- MongoDB gère la replication avec le concept de replica set : un ensemble d'instances mongoDB qui hébergent les mêmes données.
- Exemple de replica set à 3 nœuds : 1 maître, 2 esclaves :



  - Le maître est responsable du stockage de la donnée principale.
  - Un journal des transactions est maintenu par le maître (une collection spéciale nommée opLog).
  - La réplication vers les deux esclaves se fait en mode asynchrone.

Réplication des données – En pratique sous MongoDB

- 2 niveaux de cohérence sont possibles :
  - Cohérence forte : imposant au client d'effectuer toujours les lectures via le maître
  - Cohérence à terme : les clients peuvent effectuer des lectures sur les esclaves.
- La robustesse du système est assurée par l'échange de messages de surveillance entre les nœuds, et par la mise en œuvre d'un processus d'élection d'un nouveau maître par la majorité des nœuds survivants en cas de panne.
- Si le nombre de votants est pair, la désignation d'un nouveau maître peut-elle être impossible ?

Réplication des données – En pratique sous MongoDB

- Sous mongoDB, la création du replica set à 3 nœuds se fait, à partir de l'un des nœuds (« n1 » dans l'exemple), en lançant les commandes :
  - rs.initiate()
  - rs.add(« adresse IP du n2 »)
  - rs.add(« adresse IP du n3 »)
- mongoDB désigne automatiquement un maître. Pour le connaitre :
  - rs.isMaster() ou rs.status()
- mongoDB permet d'autoriser ou d'interdire les lectures sur un esclave

Réplication des données – En pratique sous MongoDB

- Configuration de mon replica set :

Invite de commandes - mongo --port 27018

```
        "localTime" : ISODate("2015-10-28T13:52:03.111Z"),
        "maxWireVersion" : 3,
        "minWireVersion" : 0,
        "ok" : 1
}
test:PRIMARY> rs.status()
{
        "set" : "test",
        "date" : ISODate("2015-10-28T13:54:02.852Z"),
        "myState" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "name" : "Compaq-Declercq:27018",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 1316,
                        "optime" : Timestamp(1446040306, 1),
                        "optimeDate" : ISODate("2015-10-28T13:51:46Z"),
                        "electionTime" : Timestamp(1446039611, 2),
                        "electionDate" : ISODate("2015-10-28T13:40:11Z"),
                        "configVersion" : 3,
                        "self" : true
                },
                {
                        "_id" : 1,
                        "name" : "Compaq-Declercq:27019",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 144,
                        "optime" : Timestamp(1446040306, 1),
                        "optimeDate" : ISODate("2015-10-28T13:51:46Z"),
                        "lastHeartbeat" : ISODate("2015-10-28T13:54:02.581Z"),
                        "lastHeartbeatRecv" : ISODate("2015-10-28T13:54:02.603Z"),
                        "pingMs" : 0,
                        "syncingTo" : "Compaq-Declercq:27018",
                        "configVersion" : 3
                },
                {
                        "_id" : 2,
                        "name" : "Compaq-Declercq:27020",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 136,
                        "optime" : Timestamp(1446040306, 1),
                        "optimeDate" : ISODate("2015-10-28T13:51:46Z"),
                        "lastHeartbeat" : ISODate("2015-10-28T13:54:02.581Z"),
                        "lastHeartbeatRecv" : ISODate("2015-10-28T13:54:02.620Z"),
                        "pingMs" : 0,
                        "configVersion" : 3
                }
        ],
        "ok" : 1
}
test:PRIMARY>
```

Client Application
Driver
Writes
Reads
Primary
Replication
Replication
Secondary
Secondary

Réplication des données – En pratique sous MongoDB

- Si l'on simule une panne du maitre, on observe qu'un nouveau maitre a été automatiquement désigné au sein du replica set :

Invite de commandes - mongo --port 27019

```
}
test:PRIMARY> rs.status()
{
        "set" : "test",
        "date" : ISODate("2015-10-28T14:38:01.295Z"),
        "myState" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "name" : "Compaq-Declercq:27018",
                        "health" : 0,
                        "state" : 8,
                        "stateStr" : "(not reachable/healthy)",
                        "uptime" : 0,
                        "optime" : Timestamp(0, 0),
                        "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
                        "lastHeartbeat" : ISODate("2015-10-28T14:37:58.431Z"),
                        "lastHeartbeatRecv" : ISODate("2015-10-28T14:36:44.047Z"),
                        "pingMs" : 1,
                        "lastHeartbeatMessage" : "Failed attempt to connect to Compaq-Declercq:27018
; couldn't connect to server Compaq-Declercq:27018 (192.168.0.5), connection attempt failed",
                        "configVersion" : -1
                },
                {
                        "_id" : 1,
                        "name" : "Compaq-Declercq:27019",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 3778,
                        "optime" : Timestamp(1446040306, 1),
                        "optimeDate" : ISODate("2015-10-28T13:51:46Z"),
                        "electionTime" : Timestamp(1446043008, 1),
                        "electionDate" : ISODate("2015-10-28T14:36:48Z"),
                        "configVersion" : 3,
                        "self" : true
                },
                {
                        "_id" : 2,
                        "name" : "Compaq-Declercq:27020",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 2774,
                        "optime" : Timestamp(1446040306, 1),
                        "optimeDate" : ISODate("2015-10-28T13:51:46Z"),
                        "lastHeartbeat" : ISODate("2015-10-28T14:38:00.010Z"),
                        "lastHeartbeatRecv" : ISODate("2015-10-28T14:37:59.903Z"),
                        "pingMs" : 0,
                        "configVersion" : 3
                }
        ],
        "ok" : 1
}
test:PRIMARY> _
```

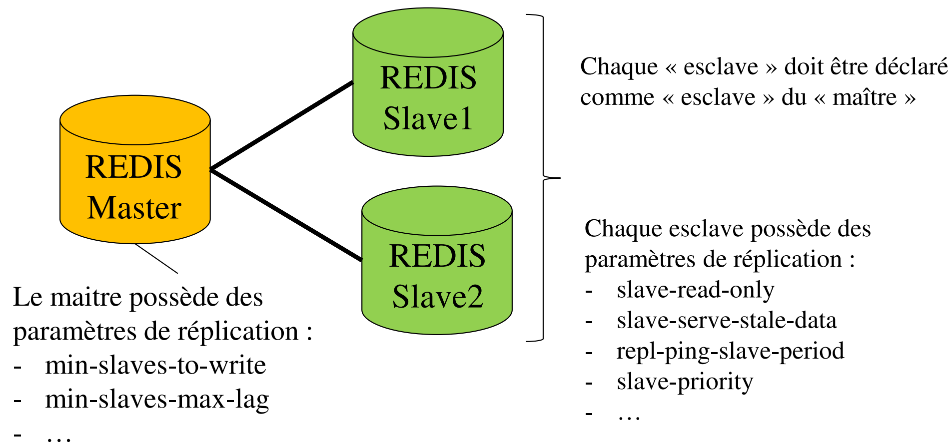
Election for New Primary
Secondary
Heartbeat
New Primary Elected
Primary
Replication
Heartbeat

Réplication des données – En pratique sous MongoDB

- mongoDB offre de nombreuses options de réplication qu'il convient de définir selon le contexte du système (nature des données hébergées, exigences de performance, localisation des différents nœuds…)
- Exemple : writeconcern permet d'indiquer sur combien de nœuds une donnée doit être mise à jour avant d'envoyer un acquittement au client.
  - Writeconcern = 1 : la mise à jour sur le maitre suffit à l'envoi de l' acquittement, les mises à jour sur les autres nœuds sont asynchrones.
  - Writeconcern = 2 : mise à jour sur le maitre + un autre nœud avant envoi de l'acquittement
  - Writeconcern = "majority" : la mise à jour est effective sur la majorité des nœuds avant envoi de l'acquittement

Réplication des données – En pratique sous Redis

- REDIS permet de très facilement mettre en place une réplication des données.



https://github.com/kubernetes/examples/blob/master/staging/storage/redis/image/redis-master.conf

Réplication des données – En pratique sous Redis

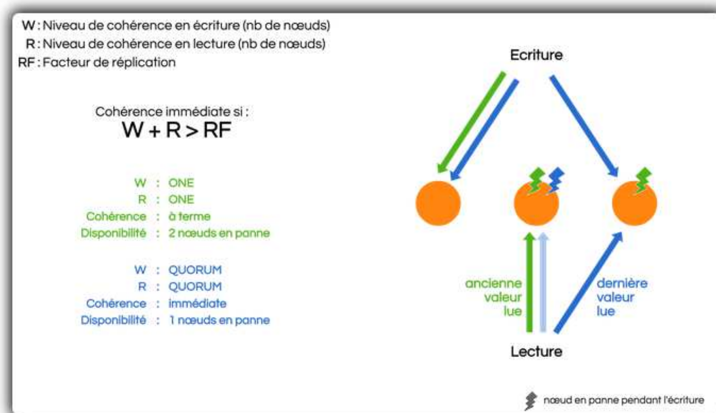
- REDIS fournit Sentinel, une solution de surveillance de l'état de la configuration afin d'assurer une haute disponibilité des données aux clients.

- Sentinel est une solution elle-même distribuée. De multiples proccess Sentinel sont actifs simultanément et échangent des informations sur l'état des bases répliquées. Chaque processus Sentinel vérifie constamment que le maître et les esclaves sont opérationnels. Un système d'alerting permet de remonter tout dysfonctionnement.

- Les processus Sentinel permettent aussi de modifier automatiquement la configuration de la réplication, en cas de défaillance du maître.

Réplication des données – En pratique

- Exemple sur Cassandra : plus de disponibilité ou plus de cohérence ?



  - Configuration verte : plus de disponibilité, moins de cohérence
  - Configuration bleue : plus de cohérence, moins de disponibilité

Réplication des données - Résumé

- La réplication est essentiellement destinée à pallier les pannes en dupliquant les données sur plusieurs serveurs et en permettant donc qu'un serveur prenne la relève quand un autre vient à faillir. Le fait de disposer des mêmes données sur plusieurs serveurs par réplication ouvre également la voie à la distribution de la charge et donc à la scalabilité.

- Ce n'est cependant pas une méthode applicable à grande échelle car, sur ce que nous avons vu jusqu'ici, elle implique la copie de toute la collection sur tous les serveurs.

Réplication des données - Résumé

- La mise au point de la réplication et des options de réplication se fait en tenant compte :
  - de l'usage des données : fréquences des lectures, des insertions, des mises à jour, …
  - des contraintes non-fonctionnelles sur ces données : taux de disponibilité, cohérence minimale requise, délais de mise à jour, …
  - Des risques à la non-cohérence et/ou à la non-disponibilité des données

- Nous étudierons au prochain chapitre le mécanisme de partitionnement des données, technique privilégiée pour obtenir une véritable scalabilité.

Université d'Évry-Val d'Essonne

université PARIS-SACLAY

Théorie et pratique de NoSQL

Chapitre 4

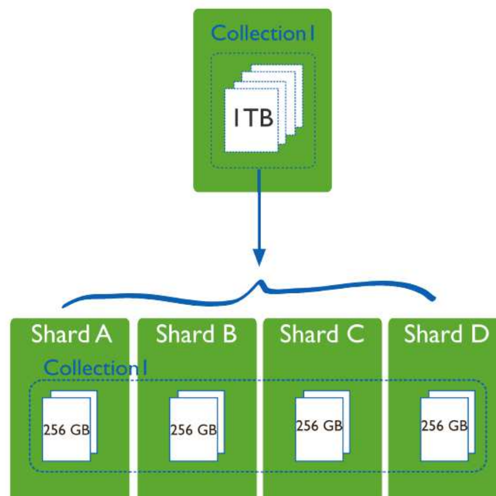
Partitionnement des données

Philippe Declercq

2022/2023

Partitionnement des données

- Pour augmenter la scalabilité du stockage des données, les systèmes NoSQL offrent des possibilités de partitionnement (sharding) des données sur différent nœuds.



Partitionnement des données

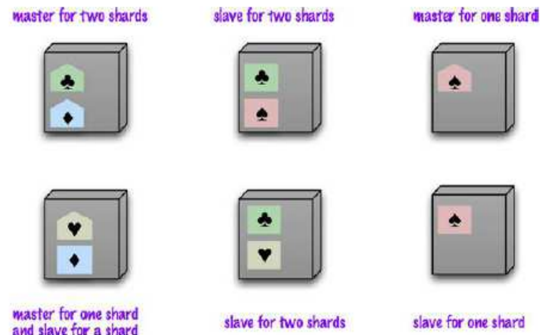
- Une partition d'un ensemble S est un ensemble $\{F1, F2, \cdots, Fn\}$ de parties de S, que nous appellerons fragments, tel que :
  - $\cup i\ Fi = S$
  - $Fi \cap Fj = \emptyset$ pour tout $i, j, i \neq j$
- Ou plus simplement : chaque élément de S est contenu dans un et un seul Fi.

- Dans notre cas, l'ensemble est une collection, les éléments de l'ensemble sont les documents.

- Les parties sont nommées fragments.

Partitionnement des données

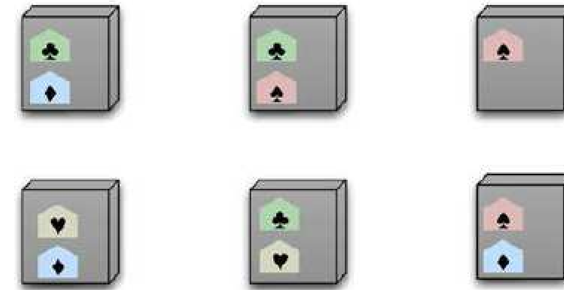
- Réplication et partitionnement en topologie Maître-esclave :
  - Plusieurs maitres, mais chaque donnée a un unique maitre,
  - 2 possibilités :
    - Un nœud peut être maitre pour certaines données et esclave pour d'autres :



    - Chaque nœud est dédié pour être maitre ou esclave.

Partitionnement des données

- Réplication et partitionnement en topologie multi-nœuds :
  - Chaque partition est présente sur 2 ou 3 nœuds



Partitionnement des données

- Partitionner les données nécessite de définir une clé de partitionnement.
- La clé est formée à partir d'un ou plusieurs attributs des documents.
- Une bonne clé doit permettre le partitionnement en fragments de taille comparable.
- Lorsque certaines partitions stockent plus de données ou reçoivent plus de requêtes que les autres, on parle de phénomène de hotspotting.

Partitionnement des données

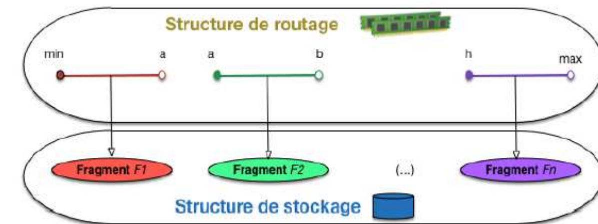
- La stratégie la plus simple pour avoir des fragments de taille comparable est de choisir de manière aléatoire un fragment pour chaque nouvelle information à stocker.
- Mais…cela risque de pénaliser les lectures qui devront se faire sur l'ensemble des nœuds du cluster !
- Il faut donc trouver un compromis entre une bonne répartition des données et l'optimisation des accès.

Partitionnement des données

- Il existe deux grandes approches pour déterminer une partition en fonction d'une clé : par intervalle et par hachage.
- Par intervalle, on obtient un ensemble d'intervalles disjoints couvrant le domaine de valeurs de la clé.
  - A chaque intervalle correspond un fragment,
  - Exemple : si la clé est une chaîne de caractères, on peut considérer que les chaînes dont la première lettre est comprise entre A et G verront leur document stocké sur le fragment 1, entre H et P sur le fragment 2, et entre Q et Z sur le fragment 3.
- Par hachage, une fonction appliquée à la clé détermine le fragment d'affectation.

Partitionnement des données

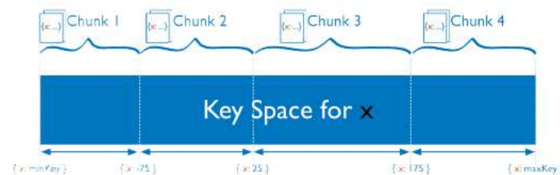
- Dans un partitionnement par intervalle, le domaine de valeur de la clé de partition est divisé en n intervalles définissant n fragments :



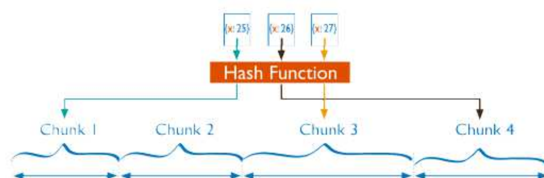
- À chaque intervalle est associé un fragment dans la structure de stockage.
- La structure de routage est constituée de paires (I, a) où I est la description d'un intervalle et a l'adresse du fragment correspondant.

Partitionnement des données

- Dans un partitionnement par hashage, une fonction de hash est appliquée sur les données de la clé.
- Le résultat de la fonction de hashage sert alors de clé de partitionnement dans les différents fragments.
- Part. par intervalle :



- Part. par hash :



Partitionnement des données

- Un partitionnement par intervalle donne de meilleurs résultats pour des requêtes basées sur des attributs de la clé. Dans de telles requêtes, le routeur sait facilement envoyer la requête aux partitions concernées.
- Seulement, le partitionnement par intervalle peut entraîner une répartition inégale des données et des requêtes, ce qui peut annuler certains des avantages du partitionnement.
- Le partitionnement par hash, par contre, assure une meilleure distribution des documents.
- Mais cette distribution rend plus probable qu'une requête sur les données de partitionnement ne puisse pas cibler quelques fragments mais plus probablement l'ensemble des fragments afin de retourner un résultat.

Partitionnement des données

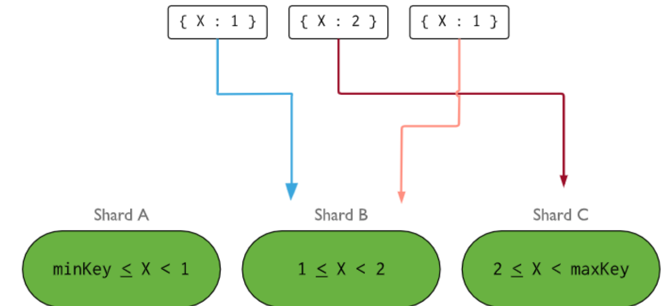
- Le choix de la stratégie de partitionnement des données est fondamental pour obtenir des performances correctes.

- Idéalement le partitionnement doit assurer :
  - Une répartition « équilibrée » des données stockées,
  - Une distribution « équilibrée » de l'ensemble des insertions, mises à jour et suppressions,
  - Une distribution « équilibrée » des interrogations,
  - L'envoi de chaque requête sur le minimum de partitions.

- A prendre en considération selon les usages désirés :
  - Scalabilité et performance en écriture ?
  - Scalabilité et performance en lecture ?
  - Les deux ?
- …. et faire des compromis…

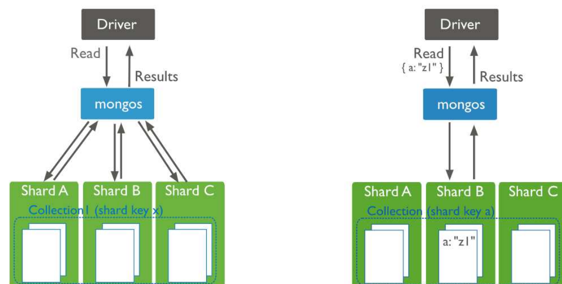
Partitionnement des données

- Comment choisir une bonne clé de partitionnement ?
  - La clé de partitionnement doit être immutable et présente dans toutes les demandes d'insertion.
  - Faire attention à la cardinalité de la clé de partitionnement
    - Il faut qu'il y ait un nombre suffisant de valeurs distinctes pour que chaque partition soit utilisée de façon uniforme.
    - Exemple : ici X a une cardinalité trop faible…



Partitionnement des données

- Comment choisir une bonne clé de partitionnement ?
  - Répartir les écritures
    - Attention aux clés croissantes type numéro, compteur, timestamp, … qui entraineraient une charge très déséquilibrée en écriture sur les partitions
  - Cibler les lectures
    - Les lectures les plus rapides sont celles sur lesquelles le système va pouvoir cibler une seule partition (ou un sous-ensemble).

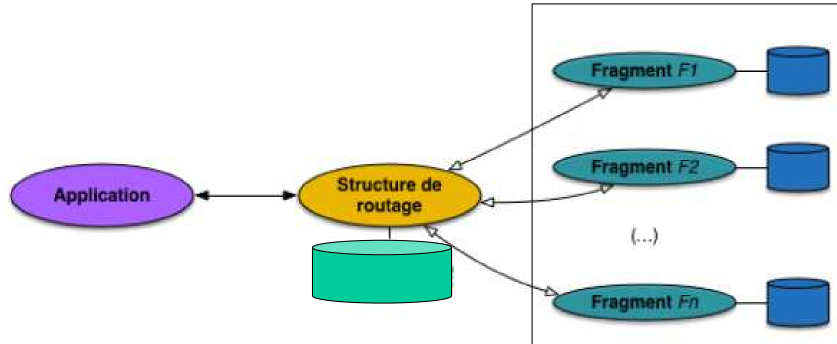


Partitionnement des données

- Comment choisir une bonne clé de partitionnement ?
  - Paralléliser les opérations
    - Exemple : répartition par utilisateur sur les site de e-commerce, webmail, …

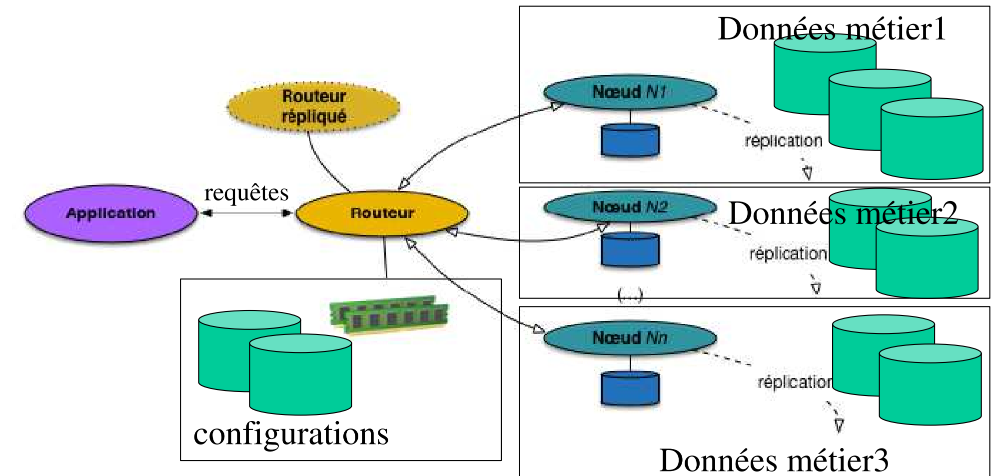
Partitionnement des données

- Un système partitionné est construit avec 2 structures :
  - La structure de routage établit la correspondance entre la valeur d'une clé et le fragment qui lui est associé (ou, très précisément, l'espace de stockage de ce fragment).
  - La structure de stockage est un ensemble d'espaces de stockages séparés, contenant chacun un fragment.



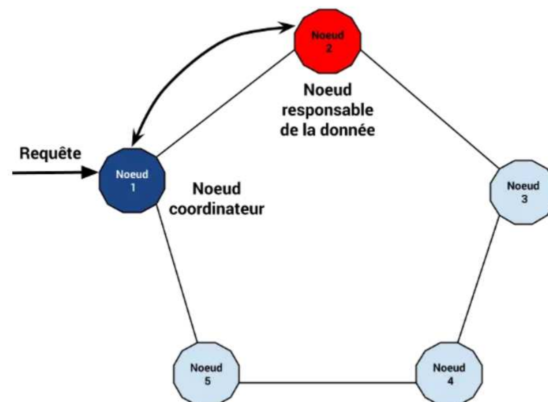
Partitionnement des données

- Dans un système distribué, cela donne :



Partitionnement des données

- Ou bien dans un système où les nœuds sont égaux, chacun des nœuds peut recevoir une requête et jouer le rôle de coordinateur dans le cluster :

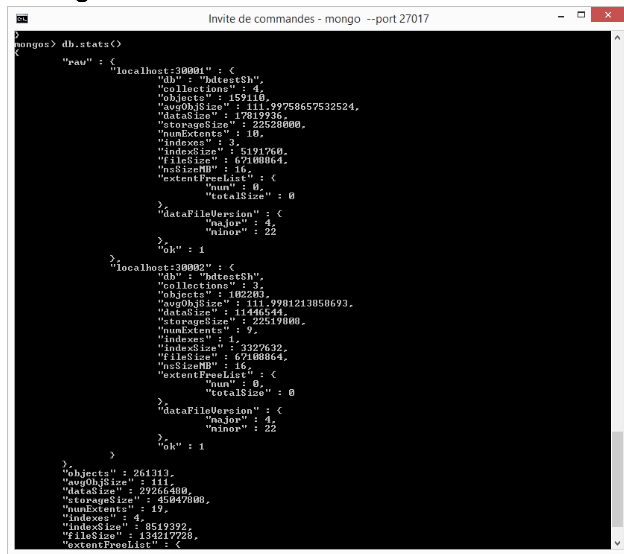


Partitionnement des données – En pratique sous MongoDB

- Sous mongodb, la structure à créer se compose des éléments suivants :
  - les routeurs (processus mongos) communiquent avec les applications clientes, se chargent de diriger les requêtes vers les serveurs de stockage concernés, et transmettent les résultats.
  - les replica set (processus mongod) servent au stockage des données (avec réplication). Un replica set est en charge d'un ou plusieurs fragments (shards) et gère localement la reprise sur panne par réplication.
  - les config servers (processus mongod avec option configsrv) stockent les données de routage et plus généralement la configuration complète du système : liste des replica sets (avec, pour chacun, le maître et les esclaves), liste des fragments et allocation des fragments à chaque replica set.

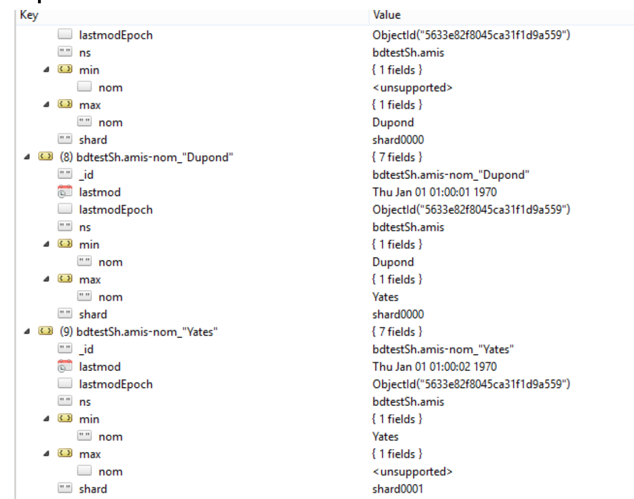
Partitionnement des données – En pratique sous MongoDB

- L'insertion en masse dans une collection entraîne alors un partitionnement du stockage des documents :



Partitionnement des données – En pratique sous MongoDB

- Si l'on décide de partitionner selon une clé correspondant à un ou plusieurs attributs des documents, mongodb partitionne automatiquement les documents sur les shards disponibles :



Partitionnement des données – Résumé

- Réplication et partitionnement des données assurent la scalabilité du système de stockage des données, et permettent :
  - De sécuriser le stockage des données,
  - De supporter des augmentations de charge sans impacts sur les performances.

- Exemple d'architecture distribuée : VITAM
  https://www.programmevitam.fr/ressources/Doc0.15.1/html/archi/technique/services/mongodb.htm

- Toutes les solutions NoSQL implémentent nativement ces mécanismes de réplication et partitionnement. C'est en effet l'une des caractéristiques des bases NoSQL !
- Des solutions du monde SQL tentent aussi de proposer ces solutions. Exemple : ***https://vitess.io/docs/overview/whatisvitess/***