



Master IASD 2019-2020

Projet NoSQL

Daniel OMOLA

Table des matières

I.	Introduction.....	3
A.	Motivation du choix des moteurs NoSQL.....	3
B.	MongoDB.....	3
C.	Couchbase	3
D.	Le jeu de données.....	3
II.	Installation des outils et chargement des données	4
A.	MongoDB.....	4
a.	Installation, lancement et création de la base	4
b.	Chargement et préparation des données	6
B.	Couchbase	7
a.	Installation, lancement et création de la base	7
b.	Chargement et préparation des données	9
III.	Test des bases sur des cas d'usage.....	10
A.	Requêtes Analyste de données	10
a.	Créer un nouveau champ isodate contenant la date au format unix	10
b.	Trouver l'ensemble des articles relatifs aux bourses mondiales, dont le texte contient l'un ou plusieurs des mots clés suivant : fall, crash, crack, decline, panic, fear.....	12
c.	Comptabiliser le nombre de documents par pays (ordre décroissant).....	13
B.	Requêtes utilisateur standard	15
a.	Lire les 10 derniers articles publiés sur la bourse de Tokyo (ou une autre bourse)	15
b.	Trouver l'ensemble des articles relatifs aux bourses mondiales publiés à une date donnée (ici le jeudi noir).....	16
IV.	Analyse théorique	18
A.	L'architecture des clusters MongoDB et Couchbase.....	19
B.	La gestion de la réplication.....	19
C.	La gestion du partitionnement.....	20
D.	La reprise sur panne	21
E.	La gestion des transactions	22
F.	La gestion de la cohérence, de la disponibilité et la tolérance au partitionnement.....	23
V.	Conclusion	25
A.	Cas d'usage et analyse théorique.....	25
B.	Difficultés rencontrées dans la réalisation du projet	25

I. Introduction

A. Motivation du choix des moteurs [NoSQL](#)

Parmi les bases de données NoSQL orientées document, MongoDB et Couchbase occupent respectivement la 1^{ère} et 2^{ème} place au sein de l'industrie. Ces deux outils, open source, sont largement plébiscités dans le développement et le déploiement d'applications Web, mobiles et IoT de premier plan. Parmi leurs clients, on compte des multinationales telles qu'Adobe, Amadeus, Barclays, BBVA, Bosch, Cisco...

Compte tenu de leur popularité, il m'apparaît pertinent de se former sur ces deux outils en vue d'accroître mon employabilité et l'intérêt des recruteurs potentiels.

B. MongoDB

Lancée par MongoDB Inc en 2007, MongoDB est une base de données sans schéma, qui stocke les données dans des fichiers de type BSON, un format binaire de JSON, qui permet une intégration facile et rapide des données.

C. Couchbase

Lancée par Couchbase Inc en 2010 sous le nom de Membase, Couchbase est également une base de données sans schéma, qui stocke les données dans des fichiers de type JSON.

D. Le jeu de données

Mon background de financier et mon intérêt pour la catégorisation textuelle appliquée à la finance m'ont conduit à choisir le jeu de données [Reuters-21578](#), publié par Carnegie Group, Inc. et Reuters, Ltd (fournisseur d'informations financières). Ce jeu de données est constitué d'articles journalistiques de la société Reuters au cours de l'année 1987 et couvre notamment les périodes ayant précédé et suivi le krach d'octobre 1987 et le lundi 19 octobre 1987, communément connu sous le nom du "[lundi noir](#)".

D'un point de vue recherche financière et journalistique, ce jeu de données est intéressant à de nombreux égards et notamment pour l'analyse du sentiment ([lien](#) vers un article de [torwarddatascience.com](#)).

Télécharger les données : [reuters_26-09-1987.json](#)

Les données sont constituées d'objets imbriqués affichant un niveau d'imbrication à deux niveaux, comme présenté ci-dessous.

```
{
  "_id":1,
  "date":"26-FEB-1987 15:01:01.79",
  "topics":"cocoa",
  "places":"el-salvador usa uruguay",
  "people":"",
  "orgs":"",
  "exchanges":"",
  "companies":"",
  "text":
    {
      "title":"BAHIA COCOA REVIEW",
      "dateline":"SALVADOR, Feb 26 -",
      "body":"Showers continued throughout the week in the Bahia cocoa zone, alleviating the drought since early January and improving prospects for the coming temporao..."
    }
}
```

II. Installation des outils et chargement des données

A. MongoDB

a. Installation, lancement et création de la base

i. Installation

Pour installer la version Community (Community Edition) de MongoDB sous Windows, j'ai suivi la procédure fournie par MongoDB sur le lien ci-dessous.

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>

ii. Lancement

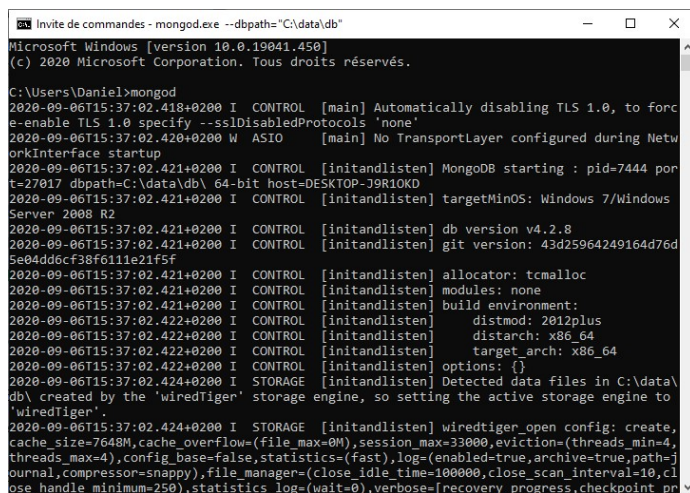
Une fois l'installation terminée, lancer MongoDB depuis le terminal :

1. Ouvrir un terminal et exécuter l'instruction :

`mongod.exe --dbpath="C:\data\db"`

ou

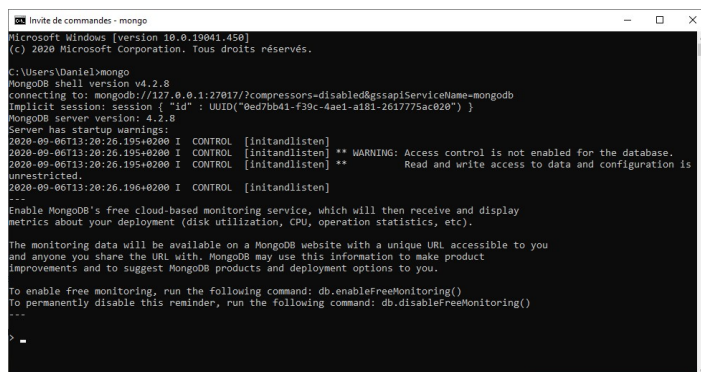
`mongod`



```
Microsoft Windows [version 10.0.19041.450]
(c) 2020 Microsoft Corporation. Tous droits réservés.

C:\Users\Daniel>mongod
2020-09-06T15:37:02.418+0200 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2020-09-06T15:37:02.420+0200 W ASIO [main] No TransportLayer configured during NetworkInterface startup
2020-09-06T15:37:02.421+0200 I CONTROL [initandlisten] MongoDB starting : pid=7444 port=27017 dbpath=C:\data\db\ 64-bit host=DESKTOP-J9R1OKD
2020-09-06T15:37:02.421+0200 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2020-09-06T15:37:02.421+0200 I CONTROL [initandlisten] db version v4.2.8
2020-09-06T15:37:02.421+0200 I CONTROL [initandlisten] git version: 43d25964249164d76d5e04dd6cf38f611e21f5f
2020-09-06T15:37:02.421+0200 I CONTROL [initandlisten] allocator: tcmalloc
2020-09-06T15:37:02.421+0200 I CONTROL [initandlisten] modules: none
2020-09-06T15:37:02.421+0200 I CONTROL [initandlisten] build environment:
2020-09-06T15:37:02.422+0200 I CONTROL [initandlisten] distmod: 2012plus
2020-09-06T15:37:02.422+0200 I CONTROL [initandlisten] distarch: x86_64
2020-09-06T15:37:02.422+0200 I CONTROL [initandlisten] target arch: x86_64
2020-09-06T15:37:02.422+0200 I CONTROL [initandlisten] options: {}
2020-09-06T15:37:02.424+0200 I STORAGE [initandlisten] Detected data files in C:\data\db\ created by the 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTiger'.
2020-09-06T15:37:02.424+0200 I STORAGE [initandlisten] wiredtiger_open config: create, cache_size=7648M,cache_overflow=(file_max=0M),session_max=33000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000,close_scan_interval=10,close_handle_minimum=250),statistics_log=(wait=0),verbose=(recovery_progress,checkpoint_pr
```

2. Ouvrir un nouveau terminal et exécuter l'instruction : `mongo`



```
Microsoft Windows [version 10.0.19041.450]
(c) 2020 Microsoft Corporation. Tous droits réservés.

C:\Users\Daniel>mongo
MongoDB shell version v4.2.8
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("8ed7bb41-f39c-4ae1-a181-2617775ac028") }
MongoDB server version: 4.2.8
Server has startup warnings:
2020-09-06T13:20:26.195+0200 I CONTROL [initandlisten] 
2020-09-06T13:20:26.195+0200 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-09-06T13:20:26.195+0200 I CONTROL [initandlisten] **      Read and write access to data and configuration is
unrestricted.
2020-09-06T13:20:26.196+0200 I CONTROL [initandlisten] 
...
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
>
```

iii. Création d'une base

1. Afficher les différentes bases disponibles : `show dbs`

```
Invite de commandes - mongo
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
```

2. Créer une nouvelle base nommée reuters ou, s'y connecter quand elle existe : `use reuters`

The image shows two windows. On the left is a terminal window titled 'Invite de commandes - mongo' with the following output:

```
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
reuters  0.021GB
>
```

On the right is the MongoDB Compass interface. The 'DATABASES' tab is selected, showing a table of databases:

Database Name	Storage Size	Collections	Indexes
admin	20.0KB	0	1
config	36.0KB	0	2
local	36.0KB	1	1
reuters	481.0MB	4	5

3. Créer une collection nommée articles au sein de la base reuters :
`db.createCollection('articles')`

Cette collection accueillera l'ensemble des documents issus du fichier json.

b. Chargement et préparation des données

1. Ouvrir un nouveau terminal et exécuter l'instruction :

```
mongoimport --db reuters --collection articles --file "C:\Users\Daniel\Google Drive\1-MASTER\NoSQL\data\reuters_26-09-1997.json"
```

Attention : cette instruction retournera une erreur si elle est exécutée depuis le shell mongo.

2. Vérifier que la collection articles a bien été alimentée : `db.articles.find().pretty()`
3. Afficher les statistiques de la collection : `db.articles.stats()`

```
{
  "ns" : "reuters.articles",
  "size" : 21111437,
  "count" : 21495,
  "avgObjSize" : 982,
  "storageSize" : 12775424,
  "capped" : false,
  "wiredTiger" : {
```

4. Créer une collection `articles_5k` (contenant 5 milles lignes) et afficher ses statistiques :

```
db.articles.aggregate([ { $limit: 5000 }, { $out: 'articles_5k' } ]); db.articles_5k.stats();
```

```
> db.articles_5k.stats();
{
  "ns" : "reuters.articles_5k",
  "size" : 4932263,
  "count" : 5000,
  "avgObjSize" : 986,
  "storageSize" : 4096,
  "capped" : false,
  "wiredTiger" : {
```

5. Créer une collection `articles_10k` et afficher ses statistiques :

```
db.articles.aggregate([ { $limit: 10000 }, { $out: 'articles_10k' } ]); db.articles_10k.stats();
```

```
> db.articles_10k.stats();
{
  "ns" : "reuters.articles_10k",
  "size" : 10075467,
  "count" : 10000,
  "avgObjSize" : 1007,
  "storageSize" : 4096,
  "capped" : false,
  "wiredTiger" : {
```

The screenshot shows the MongoDB Compass Community interface. On the left, the 'Local' sidebar shows the 'reuters' database selected. The main panel displays a table of collections:

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
articles	21,495	982.2 B	20.1 MB	1	224.0 KB	
articles_10k	10,000	1007.5 B	9.6 MB	1	112.0 KB	
articles_5k	5,000	986.5 B	4.7 MB	1	64.0 KB	

B. Couchbase

a. Installation, lancement et création de la base

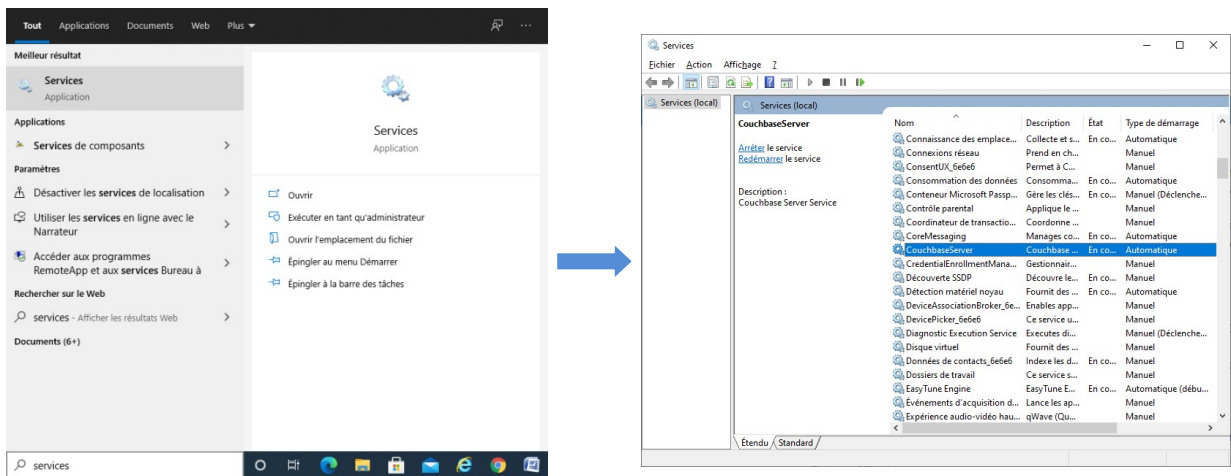
i. Installation

Pour installer la version Couchbase Server 6.6 sous Windows, j'ai suivi la procédure fournie par Couchbase sur le lien ci-dessous.

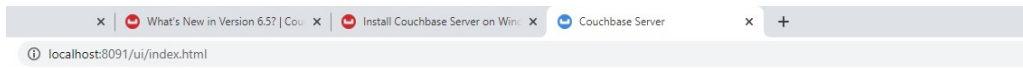
<https://docs.couchbase.com/server/current/install/install-package-windows.html>

ii. Lancement

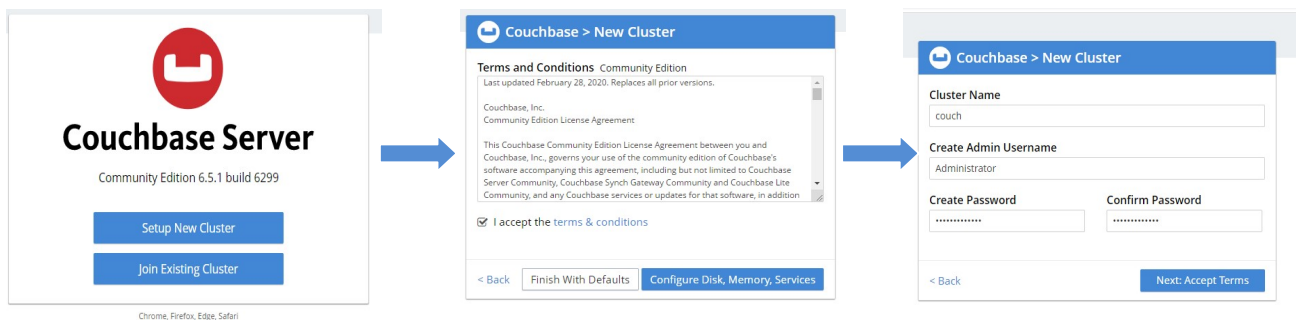
1. Ouvrir les services Windows et activer le service CouchbaseServer.



2. Se connecter à l'adresse <http://localhost:8091/ui/index.html> pour lancer l'interface graphique.

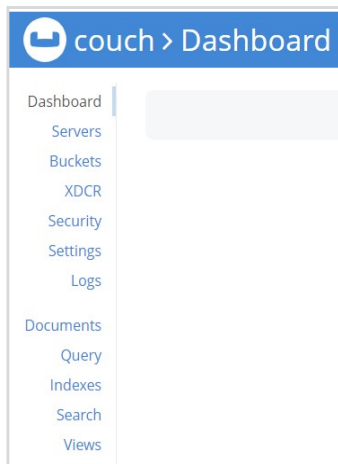


3. Créer un nouveau cluster



iii. Création d'une base

Créer un Bucket reuters depuis le Dashboard

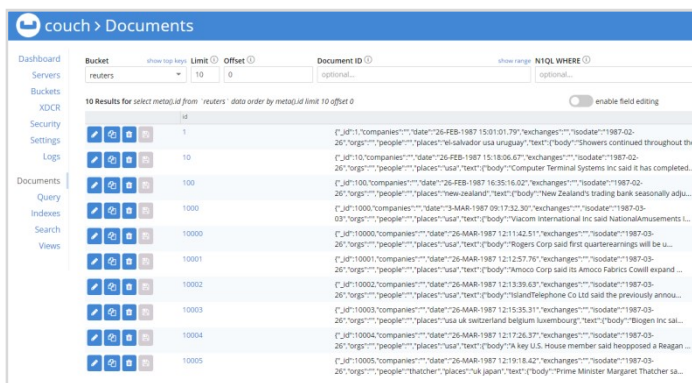


b. Chargement et préparation des données

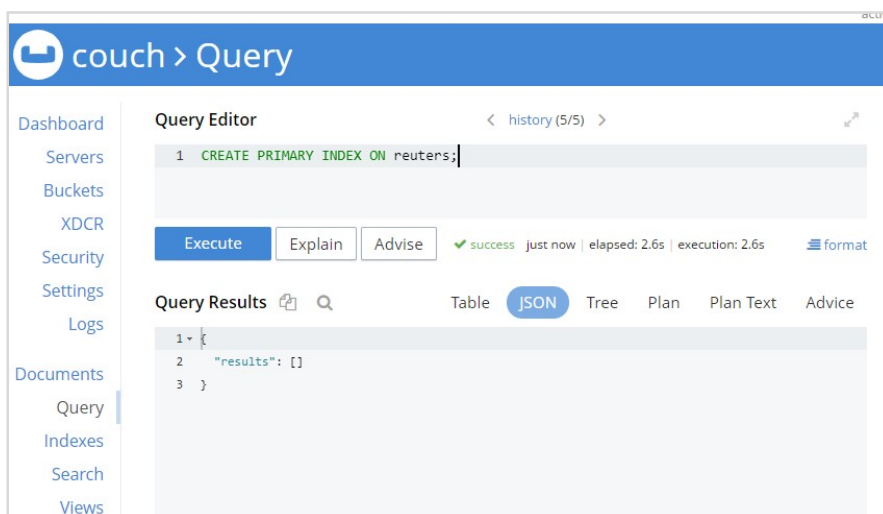
1. Pour charger les données, ouvrir un terminal et exécuter l'instruction :

```
cbimport json -c couchbase://127.0.0.1 -u Administrator -p Reussite1000% -b reuters -d
"file:///C:/Users/Daniel/Google Drive/1-MASTER/NoSQL/data/reuters_26-09-1997.json" --
generate-key %_id% -f lines
```

Les documents chargés sont accessibles sous le menu Document du Dashboard.



2. Créer un index pour le bucket reuters : **CREATE PRIMARY INDEX ON reuters;**



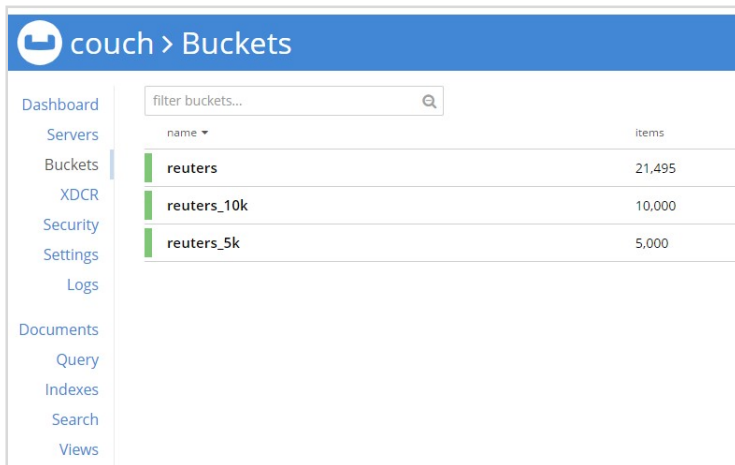
3. Alimenter le bucket reuters_10k et créer la clé primaire d'indexation

```
INSERT INTO reuters_10k(KEY _k, VALUE _v)
SELECT META().id _k, _v
FROM reuters _v LIMIT 10000;
CREATE PRIMARY INDEX ON reuters_10k;
```

4. Alimenter le bucket reuters_5k et créer la clé primaire d'indexation

```
INSERT INTO reuters_5k(KEY _k, VALUE _v)
SELECT META().id _k, _v
FROM reuters _v LIMIT 5000;
CREATE PRIMARY INDEX ON reuters_5k;
```

5. Vérifier la création des 3 buckets dans l'interface utilisateur



The screenshot shows the CouchDB Buckets interface. The left sidebar contains navigation links: Dashboard, Servers, Buckets (selected), XDCR, Security, Settings, Logs, Documents, Query, Indexes, Search, and Views. The main area has a search bar labeled 'filter buckets...' and a table with two columns: 'name' and 'items'. The table lists three buckets: 'reuters' with 21,495 items, 'reuters_10k' with 10,000 items, and 'reuters_5k' with 5,000 items.

name	items
reuters	21,495
reuters_10k	10,000
reuters_5k	5,000

III. Test des bases sur des cas d'usage

A. Requêtes Analyste de données

- a. Créer un nouveau champ isodate contenant la date au format unix

i. MongoDB

MongoDB offre la simplicité et rapidité au travers de fonctions de gestion des dates flexibles, capables de s'adapter à de nombreux formats de chaînes de caractères.

```
db.articles.aggregate( [  
  { $addField: {  
    ...   isodate: {  
    ...     $dateFromString: { dateString: {  
    ...       $substr: ["$date", 0, 11]  
    ...     }  
    ...   }  
    ... }  
    ... }  
    ... }  
    ... },  
  { "$out": "articles" } ] )
```

ii. Couchbase

La gestion des dates est complexe en raison de fonction de gestion de dates peu flexibles.

```
UPDATE reuters SET dt = split(REPLACE (substr(date, 0, REGEXP_POSITION(date,  
" ") ) , substr(date, REGEXP_POSITION(date, " ") - 8, 3) ,  
CASE  
WHEN date LIKE '%JAN%' THEN '01'  
WHEN date LIKE '%FEB%' THEN '02'  
WHEN date LIKE '%MAR%' THEN '03'  
WHEN date LIKE '%APR%' THEN '04'  
WHEN date LIKE '%MAY%' THEN '05'  
WHEN date LIKE '%JUN%' THEN '06'  
WHEN date LIKE '%JUL%' THEN '07'  
WHEN date LIKE '%AUG%' THEN '08'  
WHEN date LIKE '%SEPT%' THEN '09'  
WHEN date LIKE '%OCT%' THEN '10'  
WHEN date LIKE '%NOV%' THEN '11'  
WHEN date LIKE '%DEC%' THEN '12'  
END  
) , '-');  
UPDATE reuters SET dt_string = concat(dt[2], '-', dt[1], '-', dt[0]);  
UPDATE reuters SET isodate = date_format_str(dt_string, "1234-12-12");  
UPDATE reuters SET dt_string = concat(dt[2], '-', dt[1], '-0', dt[0]) WHERE  
isodate IS NULL;
```

```
UPDATE reuters SET isodate = date_format_str(dt_string,"1234-12-12")
WHERE isodate IS NULL;
UPDATE reuters UNSET dt , dt_string;
```

- b. Trouver l'ensemble des articles relatifs aux bourses mondiales, dont le texte contient l'un ou plusieurs des mots clés suivant : fall, crash, crack, decline, panic, fear.

i. MongoDB

Collection articles_5k : 225 millisecondes

```
db.articles_5k.find({
  exchanges : { $exists: true , $ne: ""} ,
  "text.body": { $regex: /fall|crash|crack|decline|panic|fear/}
}).explain("executionStats")
```

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 12,
  "executionTimeMillis" : 225,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 5000,
  "executionStages" : {
```

Collection articles_10k : 464 millisecondes

```
db.articles_10k.find({
  exchanges : { $exists: true , $ne: ""} ,
  "text.body": { $regex: /fall|crash|crack|decline|panic|fear/}
}).explain("executionStats")
```

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 20,
  "executionTimeMillis" : 464,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 10000,
  "executionStages" : {
```

Collection articles : 989 millisecondes

```
db.articles.find({
  exchanges : { $exists: true , $ne: ""} ,
```

```
"text.body": { $regex: /fall|crash|crack|decline|panic|fear/
}).explain("executionStats")
```

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 70,
  "executionTimeMillis" : 989,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 21495,
  "executionStages" : {
```

ii. Couchbase

Bucket articles_5k : 364 milliseconds

```
SELECT * FROM reuters_5k WHERE LENGTH(exchanges)>0 AND
REGEXP_CONTAINS(text.body, "fall|crash|crack|decline|panic|fear");
```

```
✓ success just now | elapsed: 364.2ms
execution: 364.2ms docs: 17 | size: 51320 bytes
```

Bucket articles_10k : 693 milliseconds

```
SELECT * FROM reuters_10k WHERE LENGTH(exchanges)>0 AND
REGEXP_CONTAINS(text.body, "fall|crash|crack|decline|panic|fear");
```

```
✓ success just now | elapsed: 693.2ms
execution: 693.2ms docs: 29 | size: 81552 bytes
```

Bucket articles : 1.3 secondes

```
SELECT * FROM reuters WHERE LENGTH(exchanges)>0 AND
REGEXP_CONTAINS(text.body, "fall|crash|crack|decline|panic|fear");
```

```
✓ success 1 min ago | elapsed: 1.3s | execution: 1.3s
docs: 70 | size: 164795 bytes
```

c. Comptabiliser le nombre de documents par pays (ordre décroissant)

i. MongoDB

Collection articles_5k : 11 milliseconds

```
d = new Date;
```

```
db.articles_5k.aggregate([
  { $group: { _id: "$places", count: { $sum: 1 } } },
  {$sort:{"count":-1}}
]);
print(new Date - d + 'ms')
```

```
> print(new Date - d + 'ms')
11ms
```

Collection articles_10k : 13 milliseconds

```
d = new Date;
db.articles_10k.aggregate([
  { $group: { _id: "$places", count: { $sum: 1 } } },
  {$sort:{"count":-1}}
]);
print(new Date - d + 'ms')
```

```
> print(new Date - d + 'ms')
13ms
```

Collection articles : 989 milliseconds

```
d = new Date;
db.articles.aggregate([
  { $group: { _id: "$places", count: { $sum: 1 } } },
  {$sort:{"count":-1}}
]);
print(new Date - d + 'ms')
```

```
> print(new Date - d + 'ms')
33ms
```

ii. Couchbase

Bucket articles_5k : 223 milliseconds

```
SELECT places,COUNT(*) AS count FROM reuters_5k GROUP BY places ORDER
BY count DESC;
```

✓ success 8 min ago | elapsed: 222.9ms | execution: 222.9ms | docs: 348 | size: 22511 bytes

Bucket articles_10k : 281 milliseconds

```
SELECT places,COUNT(*) AS count FROM reuters_10k GROUP BY places ORDER BY count DESC;
```

✓ success | just now | elapsed: 281.2ms | execution: 281.2ms | docs: 634 | size: 42642 bytes

Bucket articles : 570 millisecondes

```
SELECT places,COUNT(*) AS count FROM reuters GROUP BY places ORDER BY count DESC;
```

✓ success | just now | elapsed: 570.5ms | execution: 570.5ms | docs: 1088 | size: 75544 bytes

B. Requêtes utilisateur standard

a. Lire les 10 derniers articles publiés sur la bourse de Tokyo (ou une autre bourse)

i. MongoDB

Collection articles_5k : 0 millisecondes

```
db.articles_5k.find({ exchanges:'tse'}).sort({'isodate' : -1}).limit(10 ).pretty()
```

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 0,
  "executionTimeMillis" : 0,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 0,
  "executionStages" : {
```

Collection articles_10k : 0 millisecondes

```
db.articles_10k.find({ exchanges:'tse'}).sort({'isodate' : -1}).limit(10 ).pretty()
```

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 0,
  "executionTimeMillis" : 0,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 0,
  "executionStages" : {
```

Collection articles : 10 millisecondes

```
db.articles.find({ exchanges:'tse'}).sort({'isodate' : -1}).limit(10 ).pretty()
```

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 10,
  "executionTimeMillis" : 10,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 21495,
  "executionStages" : {
```

ii. Couchbase

Bucket Reuters_5k : 143 millisecondes

```
SELECT * FROM Reuters_5k WHERE exchanges ='tse' ORDER BY isodate DESC
LIMIT 10;
```

```
✓ success just now | elapsed: 142.6ms | execution: 142.6ms
```

Bucket Reuters_10k : 292 millisecondes

```
SELECT * FROM Reuters_10k WHERE exchanges ='tse' ORDER BY isodate DESC
LIMIT 10;
```

```
✓ success just now | elapsed: 292.2ms | execution: 292.2ms
```

Bucket Reuters : 548 millisecondes

```
SELECT * FROM Reuters WHERE exchanges ='tse' ORDER BY isodate DESC LIMIT
10;
```

```
✓ success 2 min ago | elapsed: 548.2ms | execution: 548.2ms
```

- b. Trouver l'ensemble des articles relatifs aux bourses mondiales publiés à une date donnée (ici le jeudi noir)

i. MongoDB

Collection articles_5k : 0 milliseconde

```
db. Articles_5k.find({
  "isodate":{
    "$gte": ISODate("1987-10-19T00:00:00Z"),
    "$lt": ISODate("1987-10-20T00:00:00Z")},
  exchanges : { $exists: true , $ne: ""} }).pretty();
```

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 0,
  "executionTimeMillis" : 0,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 0,
  "executionStages" : {
```

Collection articles_10k : 0 milliseconde

```
db. articles_10k.find({
  "isodate":{
    "$gte": ISODate("1987-10-19T00:00:00Z"),
    "$lt": ISODate("1987-10-20T00:00:00Z")},
  exchanges : { $exists: true , $ne: ""} }).pretty();
```

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 0,
  "executionTimeMillis" : 0,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 0,
  "executionStages" : {
```

Collection articles : 20 millisecondes

```
db. articles.find({
  "isodate":{
    "$gte": ISODate("1987-10-19T00:00:00Z"),
    "$lt": ISODate("1987-10-20T00:00:00Z")},
  exchanges : { $exists: true , $ne: ""} }).pretty();
```

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 28,
  "executionTimeMillis" : 20,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 21495,
  "executionStages" : {
```

ii. Couchbase

Bucket reuters_5k : 548 millisecondes

```
SELECT * FROM reuters_5k WHERE LENGTH(exchanges)>0 AND isodate = '1987-10-19';
```

✓ success | just now | elapsed: 152.2ms | execution: 152.2ms

Bucket articles_10k : 292 millisecondes

```
SELECT * FROM reuters_10k WHERE LENGTH(exchanges)>0 AND isodate = '1987-10-19';
```

✓ success | just now | elapsed: 292.2ms | execution: 292.2ms

Bucket articles : 548 millisecondes

```
SELECT * FROM reuters WHERE LENGTH(exchanges)>0 AND isodate = '1987-10-19';
```

✓ success | just now | elapsed: 548.2ms | execution: 548.2ms

IV. Analyse théorique

L'analyse théorique va consister à comparer les deux moteurs sous six aspects :

- l'architecture,
- la gestion de la réplication,
- la gestion du partitionnement,
- la reprise sur panne,
- la gestion des transactions et de la reprise sur panne,
- et enfin, la gestion de la cohérence, de la disponibilité et de la tolérance au partitionnement.

Par souci de lisibilité et praticité, la comparaison se fera sous forme de tableau.

A. L'architecture des clusters MongoDB et Couchbase

MongoDB	Couchbase
<p>La topologie de MongoDB adopte une approche hiérarchique de type maître esclave (master/slave).</p> <p>Un cluster MongoDB consiste en trois types de composants (installés sur différents nœuds/serveurs) :</p> <ul style="list-style-type: none">• les serveurs de données (Shards),• les routeurs de requêtes (query routers ou mongos) et,• les serveurs de configuration (configuration servers). <p>Les shards contiennent l'ensemble des données (les chunks) et disposent d'une réplique dans différents nœuds du cluster, de manière à assurer la disponibilité et la cohérence (availability and consistency). Chaque groupe de répliques est une structure hiérarchique (à l'instar du cluster) constituée d'un nœud primaire et de nœuds secondaires.</p> <p>Les routeurs de requêtes, comme leur nom l'indique, assurent le routage des requêtes. Pour ce faire, ils se connectent à l'application client, orientent les requêtes vers le shard approprié, puis retourne le résultat vers le client. Un cluster MongoDB contient à minima deux routeurs pour distribuer les opérations et permettre de gérer la tolérance aux pannes.</p> <p>Les serveurs de configuration (Config Servers), s'occupent de la connaissance du réseau, aussi bien au niveau des routeurs que les shards. Ils vont ainsi stocker les meta data du cluster (les informations sur l'arbre de routage utilisé par les routeurs, les informations de répartition de charge sur les différents shards, et la structure des ReplicaSets). Lors des traitements, les routeurs de requêtes utilisent les meta data pour orienter les opérations vers les shards appropriés. Un Cluster MongoDB contient exactement 3 serveurs de configuration.</p>	<p>La topologie de Couchbase adopte une approche horizontale de type pair à pair (peer-to-peer/P2P) qui s'appuie sur un seul type de nœud.</p> <p>Tous les nœuds jouent le même rôle au sein du cluster et peuvent communiquer les uns avec les autres.</p> <p>Au sein d'un cluster, un seul nœud est configuré avec une série de paramètres. Les autres nœuds (y compris ceux qui rejoignent le cluster), récupèrent la configuration du nœud initialement paramétré.</p> <p>In fine, le cluster est opéré en se connectant à n'importe quel nœud, via une interface Web.</p> <p>Chaque nœud exécute plusieurs processus regroupés de manière logique au sein d'un Data Manager et d'un Cluster Manager. Le Data Manager répond aux requêtes client. Le Cluster Manager gère les services plus généraux liés aux nœuds et au cluster. Il contrôle chaque nœud du cluster, détecte les échecs, redistribue les opérations, etc.</p>

B. La gestion de la réplication

La réplication est le processus qui consiste à synchroniser les données sur plusieurs serveurs de manière à disposer de plusieurs copies des mêmes données (sur différents serveurs). Elle assure, dès lors, la redondance et augmente la disponibilité des données. Ainsi, elle protège la base de la perte d'un seul serveur.

La réplication joue un rôle essentiel dans le cadre de la tolérance aux pannes puisqu'elle permet de récupérer les données après une panne matérielle et des interruptions de service.

Alors que MongoDB a opté pour une gestion de la réplication selon le principe de maître/esclave, Couchbase adopte une approche paire à paire.

MongoDB	Couchbase
<p>Sous MongoDB, la gestion de la réplication s'opère au travers de groupes de répliques (Replica Sets ou grappes de serveurs partageant des copies d'un même ensemble de documents).</p> <p>Un groupe de répliques constitue un groupe d'instance mongod contenant les mêmes données.</p> <p>Au sein du groupe d'instances, un seul mongod (mongod primaire) reçoit l'ensemble des opérations/requêtes (insert / update / delete), tandis que les instances secondaires appliquent les opérations de l'instance primaire.</p> <p>Seule l'instance primaire peut recevoir des opérations d'écritures. Les instances secondaires, répliquent de manière asynchrone les mutations de l'instance primaire (stockées dans l'oplog de l'instance primaire) et appliquent les opérations sur leurs données.</p> <p>Si l'instance primaire est indisponible, une élection sera tenue entre les instances secondaires pour désigner une nouvelle instance primaire.</p> <p>Il est également possible de conserver des répliques supplémentaires à des fins spécifiques, telles que la reprise sur panne, la création de rapports, etc.</p>	<p>Sous Couchbase, la gestion de la réplication s'opère au travers d'un système de réplique pair-à-pair.</p> <p>Pour ce faire, Couchbase crée automatiquement des copies des données actives dans chaque nœud (subset de l'ensemble des données), pour ensuite les répartir entre différents nœuds. Trois répliques, au plus, peuvent être réalisées.</p> <p>Dès lors, un nœud stocke à la fois ses données actives et les répliques de trois autres nœuds (au maximum). Aussi, il exécute les opérations de lecture et d'écriture sur ses propres données actives tout en maintenant une copie des données actives des autres nœuds. Ceci contribue alors à accroître la disponibilité et la durabilité du cluster.</p>

C. La gestion du partitionnement

Dans les systèmes de bases de données distribuées, le partitionnement a pour objectif de permettre le passage à l'échelle des applications. En la matière, les approches de MongoDB et de Couchbase se différencient particulièrement au niveau de la taille et du nombre de partition et sont fortement influencés par l'architecture adoptée.

MongoDB	Couchbase
<p>MongoDB subdivise l'ensemble des données en blocs (chunks), pour ensuite les attribuer aux partitions (shards). Chaque bloc stocke toutes les données sur une plage spécifique. Par défaut, MongoDB crée deux blocs de 64 mégaoctets par partition. Lorsqu'un bloc est plein, MongoDB le divise en deux morceaux plus petits de 32 mégaoctets.</p> <p>Pour assurer une distribution uniforme des données, MongoDB migrera automatiquement les blocs entre les partitions afin que chacune d'entre elles ait le même nombre de blocs.</p> <p>Dans le cadre de cette gestion des partitions, les serveurs de configuration (Config Servers) vont stocker les métadonnées du cluster, y compris le mappage entre les blocs de données et les partitions. Les routeurs, quant à eux, sont responsables de la migration des blocs et de la mise à jour des serveurs de configuration. Toutes les demandes de lecture et d'écriture sont envoyées aux routeurs car eux seuls peuvent déterminer où résident les données.</p>	<p>Couchbase Server partitionne les données en 1024 godets (buckets) virtuels, ou vBuckets, et les attribue à des nœuds. Comme les blocs MongoDB, les vBuckets stockent toutes les données dans une plage spécifique. Cependant, Couchbase Server attribue tous les 1 024 buckets virtuels lors de son démarrage, et il ne réaffectera pas les vBuckets à moins qu'un administrateur ne lance le processus de rééquilibrage.</p> <p>Les clients Couchbase Server maintiennent la carte du cluster qui mappe les vBuckets aux nœuds. En conséquence, les routeurs, ainsi que les serveurs de configuration sont inutiles. Les clients communiquent directement avec les nœuds.</p>

D. La reprise sur panne

La reprise sur panne est une fonctionnalité majeure des SGBD et des systèmes NoSQL. Elle est extrêmement utile dans la mesure où elle garantit la sécurité des données même en cas de panne et assure ainsi la continuité de fonctionnement des applications et l'absence de perte de données.

Le mécanisme de reprise sur panne est indispensable pour garantir la robustesse des systèmes distribués.

Une panne peut être définie comme tout dysfonctionnement du système affectant un disque (mémoire persistante) ou la RAM (mémoire volatile) d'un serveur.

MongoDB	Couchbase
<p>Le système de jeux de répliques (replica set) de MongoDB permet d'assurer une reprise sur panne automatique. Ainsi, lorsqu'une instance primaire est inactive (ne communique plus avec les instances secondaires) pendant plus de 10 secondes par défaut, une élection se tient entre les instances secondaires du cluster pour désigner une nouvelle instance primaire. Le cluster va alors tenter d'achever l'élection et reprendre les opérations normales. Si la précédente instance primaire se reconnecte au</p>	<p>Le système de réplication paire à paire de Couchbase permet d'assurer une reprise sur panne. A la différence de MongoDB, le mécanisme de reprise sur panne peut être automatique (automatic failover) ou initié par l'administrateur (manual failover).</p> <p>Lorsqu'un nœud est déclaré comme inactif une réplique de ses données actives (vBucket) est promu (automatiquement ou</p>

<p>cluster, elle deviendra alors une instance secondaire.</p> <p>Pour qu'une élection puisse être menée à bien le nombre de nœuds secondaires doit être impair pour assurer le vote à la majorité.</p> <p>Lorsque le nombre de nœud secondaires candidats à l'élection est pair, la présence d'un nœud dit Arbitre (arbitre dans ce contexte) va permettre de trancher en ramenant le nombre de nœuds du cluster à une valeur impaire.</p> <p>Un arbitre est une instance MongoDB qui fait partie intégrante d'un replica set (au même titre que l'instance primaire et les instances secondaires). A la différence des deux autres types d'instances, l'arbitre ne peut ni contenir de données, ni être désigné comme nœud primaire à l'issue d'une élection. Aussi, l'arbitre doit être ajouté par l'administrateur de la base.</p>	<p>à l'initiative de l'administrateur) en tant que données active d'un nœud existant.</p> <p>En cas d'échec d'un noeud ou de la mise en maintenance du cluster, le processus de basculement (re-synchronisation des noeuds après une panne) contacte chaque serveur contenant les répliques et met à jour la table interne qui mappe les demandes de documents des clients et les serveurs Couchbase disponibles.</p> <p>Il faut noter que l'Automatic Failover n'est pas activé par défaut dans Couchbase. Ceci dans la mesure où il faut respecter certaines règles dans le dimensionnement du cluster pour que le failover automatique se passe sans risque.</p>
---	---

E. La gestion des transactions

Dans le cadre de la gestion de bases de données, le terme transaction désigne un groupe d'opérations apportant des modifications aux données. Afin de garantir la validité d'une transaction en toutes circonstances (même en cas d'erreur ou de pannes) les transactions doivent présenter les propriétés d'atomicité, de cohérence, d'isolation et de durabilité.

L'atomicité des transactions signifie que tous les changements apportés aux données sont effectués totalement, ou pas du tout. On parle alors du principe du "tout ou rien". Le respect de ce principe permet d'éviter qu'une base soit corrompue par des opérations partiellement complétées.

Sous le principe d'atomicité, une transaction est considérée comme une seule unité d'opérations qui peut réussir complètement ou échouer complètement. Une transaction ne peut pas être exécutée partiellement. Si une condition de consultation d'une transaction échoue, la transaction entière échouera complètement et la base de données restera inchangée.

La cohérence signifie qu'une transaction doit respecter les contraintes d'intégrité des données de la base. Ainsi, toute transaction appliquée à un jeu de données cohérent (respectant les contraintes d'intégrité) doit retourner un ensemble de données cohérent. Dès lors, pour maintenir la cohérence des transactions, un système de base de données va abandonner les transactions qui risquent de générer une incohérence.

L'isolation est la possibilité pour une opération de s'exécuter uniquement après qu'une autre opération (la précédent) sur les mêmes données ait été effectuée. De cette manière, les opérations sont indépendantes les unes des autres. Dès lors, lorsqu'une base gère un accès simultané aux données, une seule opération à la fois doit être prise en charge. Pour ce faire, les données en cours de mise à jour doivent être verrouillées pour ne pas être modifiées (et / ou affichées) jusqu'à ce que l'opération soit terminée.

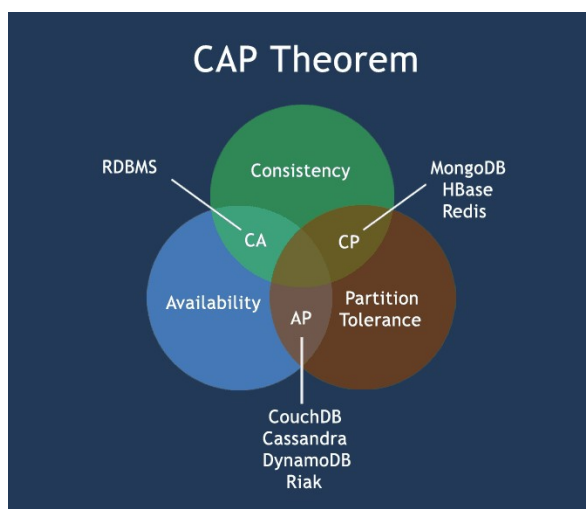
La durabilité renvoie au fait que, lorsqu'une transaction est validée, les modifications doivent être maintenues à tout moment, même en cas de panne du système (pannes de courant ou déconnexion d'Internet).

Dans le contexte des bases de données distribuées, les transactions ACID peuvent être considérées au niveau document et "multi documents" (au niveau partition et/ou au niveau Cluster). Au niveau document, toutes les opérations n'affectent qu'un seul document, tandis qu'au niveau "multi documents", toutes les opérations d'une transaction affectent plusieurs documents (situés dans une unique partition ou sur différents nœuds du cluster).

Sous MongoDB, comme sous Couchbase une transaction (lecture et écriture) affectant un seul document est atomique par nature. En effet, dans la mesure où les deux bases acceptent l'empilement de documents (grâce aux formats Json, XML et Bson), elles capturent les relations entre les données dans une structure de document unique et limite ainsi l'usage de transactions multi-documents.

MongoDB	Couchbase
<p>Depuis la version 4.0, lancée en juin 2018, MongoDB prend en charge les transactions ACID multi documents et multi collections sur les répliques de jeux de données.</p> <p>Avec l'ajout de l'isolement des instantanés (combinée à l'atomicité), la version 4.2 autorise la gestion des transactions ACID au niveau cluster. On parle alors de transactions distribuées dans lesquelles les transactions peuvent être utilisées dans plusieurs opérations, collections, bases de données, documents et fragments.</p>	<p>Depuis la version 6.5.1 Couchbase prend en charge les transactions ACID multi-documents au niveau cluster.</p>

F. La gestion de la cohérence, de la disponibilité et la tolérance au partitionnement



Selon le "CAP Theorem", un système distribué doit toujours être tolérant aux partitions, afin d'éviter qu'une partition réseau nuise à l'ensemble du système. Ainsi, lorsqu'un partitionnement existe, pour que le système puisse continuer de fonctionner, il faut garantir, soit la cohérence, soit la disponibilité car, selon le CAP Theorem, il est impossible de garantir les trois propriétés à la fois.

La cohérence signifie que lorsque l'on écrit des données sur un système distribué, on devrait pouvoir lire les mêmes données, à tout moment, à partir de n'importe quel nœud du système ou, simplement, renvoyer une erreur si les données sont dans un état incohérent. Autrement dit, tous les nœuds du système doivent voir exactement les mêmes données au même moment. Les données incohérentes ne doivent jamais être retournées.

La disponibilité signifie que le système doit toujours pouvoir exécuter, avec succès, les opérations de lecture / écriture sur tout nœud non défaillant du cluster et sans erreur. Ceci garantit que toutes les requêtes recevront une réponse. Cette disponibilité est principalement associée au partitionnement du réseau.

La tolérance au partitionnement signifie qu'en cas de partition entre les nœuds (morcellement en sous-réseaux) ou que les parties du cluster dans un système distribué ne peuvent pas communiquer, le système doit toujours fonctionner.

Il est bon de noter que le CAP Theorem est quelque peu restrictif. Dans la réalité selon les paramètres retenus, une base peut être plus CA, CP ou AP.

<u>MongoDB</u>	<u>Couchbase</u>
<p>MongoDB est fortement cohérent en raison du fait que son architecture suit une approche maître esclaves et que les opérations de lecture s'appliquent par défaut sur le nœud primaire.</p> <p>Lors de la survenue d'une panne sur le nœud primaire, le processus de basculement automatique et l'élection du nœud primaire qui en découle tendent à réduire la disponibilité du système. Toutefois, il est possible de modifier les paramètres de lecture pour que la source par défaut devienne un nœud secondaire et ainsi gagner en disponibilité au prix de la cohérence.</p> <p>Par ailleurs, le système de replica set et le processus de réplication asynchrone garantissent la tolérance au partitionnement.</p>	<p>Couchbase a été conçu pour être CP par défaut grâce à son système de réplication pair à pair et de reprise sur panne.</p> <p>Comme pour MongoDB, le délai du processus de basculement lors d'une reprise sur panne tend à réduire la disponibilité. En effet lorsque lorsqu'un nœud échoue, les données actives de ce nœud ne sont pas disponibles tant que ce nœud n'est pas basculé.</p>

V. Conclusion

A. Cas d'usage et analyse théorique

Dans les cas d'usage explorés, MongoDB s'est clairement démarquée de CouchBase. Que ce soit en termes de temps d'exécution ou de gestion des dates, MongoDB présente une performance supérieure à Couchbase. Il est toutefois bon de noter que grâce au langage database query language N1QL, la prise en main de Couchbase est bien plus aisée, ce qui facilite et accélère la transition d'un SGBDR vers une base Nosql.

Sur le plan de l'interface d'utilisation, on peut reprocher à Couchbase de manquer de souplesse, comparativement à MongoDB, qui en plus de l'interface graphique standard, offre la possibilité de contrôler l'application depuis le terminal de commande au travers du mongo Shell. Ceci présente de nombreux avantages pour l'administration de la base et pour les utilisateurs comme moi qui préfèrent utiliser les lignes de commande par souci d'efficacité de rapidité.

Du point de vue théorique, l'architecture de Couchbase et ses implications en termes de réplication, de reprise sur panne (absence de processus d'élection de nœud primaire), de scalabilité,..., et d'administration de la base, semble particulièrement intéressant pour tout utilisateur cherchant la simplicité, l'élasticité et une plus grande fiabilité. Toutefois il serait intéressant de pouvoir comparer les deux bases dans le cadre d'une configuration multi nœuds pour réellement apprécier la puissance et le potentiel de ces outils sur des données de grand volume et en situation de production.

B. Difficultés rencontrées dans la réalisation du projet

La première difficulté rencontrée sur ce projet découle du déséquilibre en matière de ressources disponibles en ligne pour les deux outils. MongoDB est largement documentée par l'éditeur et une importante communauté d'utilisateurs, alors que Couchbase, moins populaire, dispose de ressources encore limitées (ne serait-ce qu'en termes de diversité des sources ou sur des sujets comme les transactions ACID).

La seconde difficulté tient à l'évolution rapide des versions qui, malheureusement, crée des décalages entre les études publiées et les capacités actuelles des systèmes. Ainsi, j'ai noté à maintes reprises que de nombreuses publications étaient obsolètes.

Enfin la dernière difficulté tient au fait qu'il m'a fallu réaliser ce projet seul compte-tenu du fait que mes contraintes de père de famille, combinées à la période de vacances scolaires rendaient difficile toute coordination avec un camarade.