Bases de Données NoSQL

LI328 – Technologies Web

Mohamed-Amine Baazizi

Transparents de Bernd Amann
UPMC - LIP6

SGBD → Universalité

Systèmes « SQL » :

- Facilité d'utilisation
- Cohérence des données
- Persistance des données
- Fiabilité (pannes)
- Efficacité
- **Universalité**

Vues **SQL**

Données structurées

Transactions

Optimisation de requêtes

Indexation des données

« One size fits all »

Caractéristiques du Big Data

Variety: Manage and benefit from diverse data types and data structures

Velocity: Analyze streaming data and large volumes of persistent data

Volume: Scale from terabytes to zettabytes

Les évolutions...

Nouvelles **Données** :

- Web 2.0 : Facebook, Twitter, news, blogs, ...
- LOD : graphes, ontologies, ...
- Flux : capteurs, GPS, …

Nouveaux **Traitements** :

- Moteurs de recherche
- Extraction, analyse, ...
- Recommandation, filtrage collaboratif, …

Nouvelles **Infrastructures** :

- Cluster, réseaux mobiles, microprocesseurs multi-coeurs, …

➔ très gros volumes, données pas ou faiblement structurées

➔ transformation, agrégation, indexation

➔ distribution, parallélisation, redondance

Évolution → Spécialisation

Systèmes « noSQL » (***n**ot **o**nly* SQL) :

- **Facilité d'utilisation**
- Cohérence des données
- **Persistence des données**
- Fiabilité (pannes)
- **Efficacité**
- ~~Universalité~~

Langages spécialisées

Données hétérogènes

Réplication

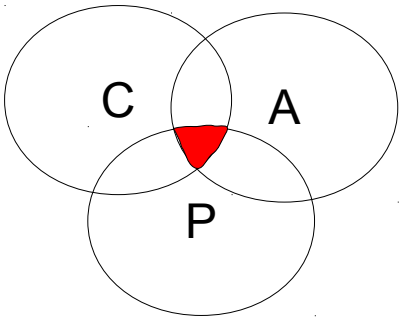
Parallélisation

Indexation de contenus

« Systèmes sur mesure »

Cohérence, Disponibilité, Pannes

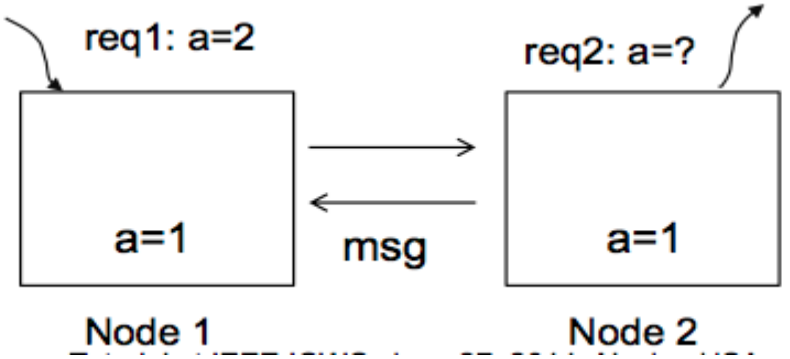
Théorème **CAP** : dans un *système distribué* il est impossible de garantir à chaque instant T plus que *deux parmi les trois* propriétés suivantes :



- **C**oherency (cohérence) :
  - tous les noeuds voient la même version
- **A**vailability (disponibilité) :
  - chaque requête obtient une réponse
- **P**artition tolerance (résistance à une panne partielle) :
  - la perte de messages n'empêche pas le système de continuer à fonctionner

Cohérence, Disponibilité, Pannes

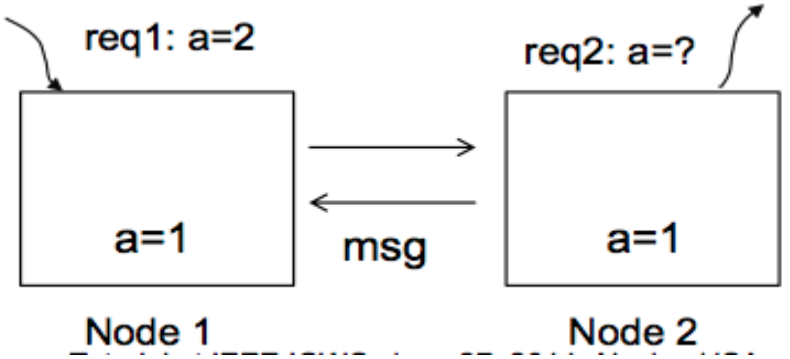
Impossibilité d'assurer C (cohérence), A (Disponibilité) et P (tolérance aux pannes) en même temps



A et C : si A alors Node 2 réponds à req2
si C alors la variable **a** de Node 2 contient 2

Cohérence, Disponibilité, Pannes

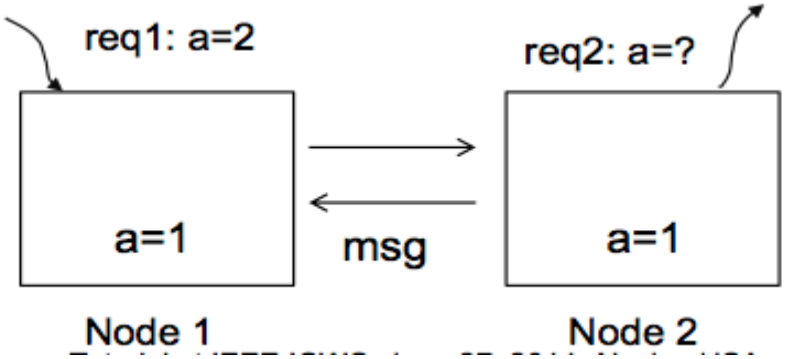
Impossibilité d'assurer C (cohérence), A (Disponibilité) et P (tolérance aux pannes) en même temps



A et P : si A alors Node 2 réponds à req2
si P alors le msg de Node1 vers Node 2 perdu et donc a=1 sur ce dernier

Cohérence, Disponibilité, Pannes

Impossibilité d'assurer C (cohérence), A (Disponibilité) et P (tolérance aux pannes) en même temps



C et P : Exercice

SQL ↔ NoSQL

Cohérence forte :

- Logique : Schémas, contraintes
- Physique : Transactions ACID

Distribution des **données**

- Transactions distribuées

↔

Ressources limitées

- Optimisation de requêtes

Langage **standard** : SQL

Cohérence faible :

- ~~Schémas, contraintes~~
- Cohérence « à terme »

Distribution des **traitements** :

- Traitements « batch »
- MapReduce

Ressources « **illimitées** »

- Passage à l'échelle horizontal

Langages **spécialisés**, API

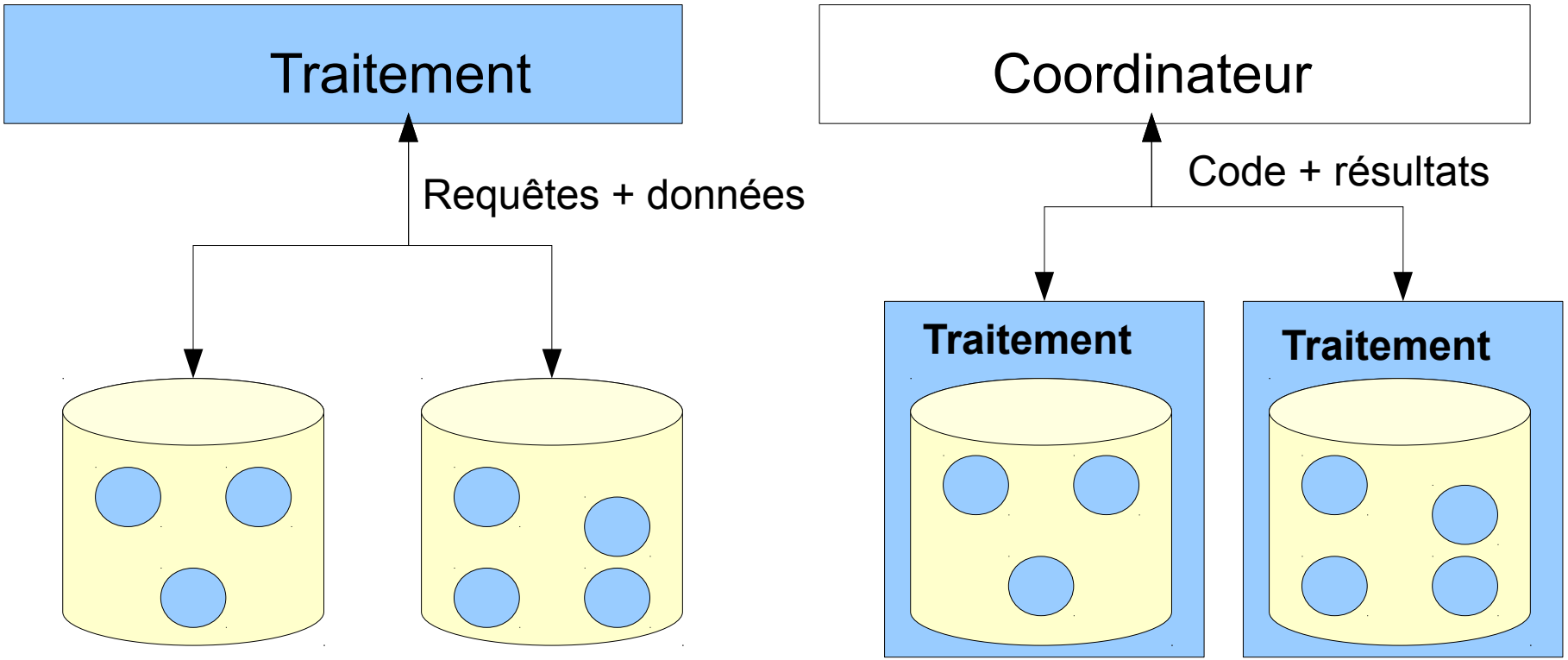
Infrastructures RAIN : le Cloud

- **R**edundant **A**rray of **I**ndependent **N**odes (cloud)
- Infrastructure à faible coût :
  - PC, open-source, LAN générique
- Tolérance aux fautes :
  - redondance du matériel, des données et des traitements
- Administration facile :
  - architecture « shared-nothing »
  - virtualisation
- Utilisation facile :
  - Modèles de programmation restreint : MapReduce

SQL ↔ NoSQL

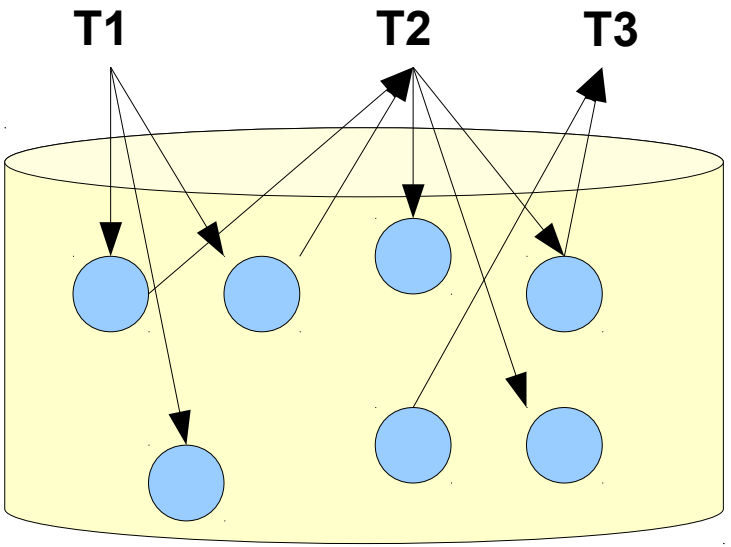
- Traitements centralisés
- Accès distribué

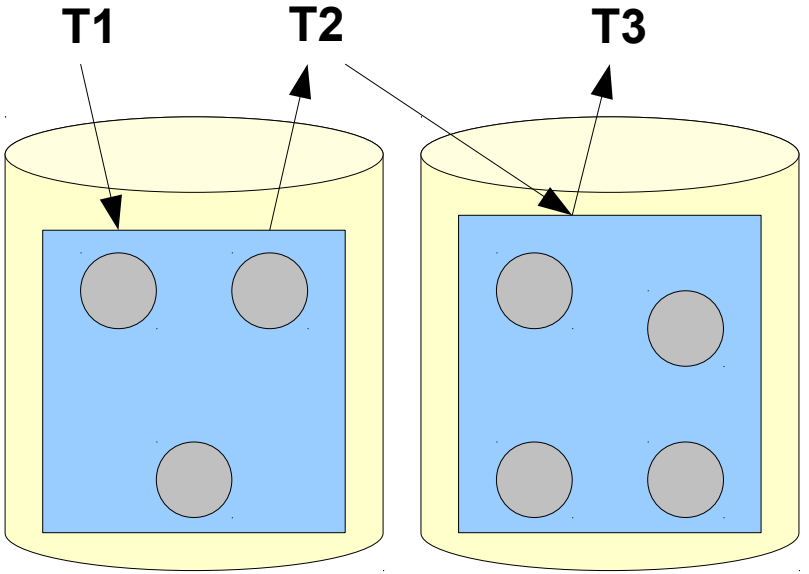
- Traitements distribués
- Accès local



SQL ↔ NoSQL

- Accès à grain fin :
  - beaucoup de lectures / écritures de de petits objets
- Accès « batch » :
  - peu de lectures / écritures de grands objets





Vers noSQL

- Étendre / adapter un SGBD traditionnel :
  - niveaux de concurrence, indexes, stockage
- Définir des systèmes spécialisés pour
  - *une* infrastructure (distribué) : cloud, clusters
  - *un* type de données : profils, documents XML, RDF, ...
  - *un* type de traitements : partage, analyse/agrégation, visualisation

Classification de systèmes

- Types de données : tables, clés/valeurs, arbres, graphes, documents
- Paradigme (langages) : map/reduce (PIG, Hive)
- API / Protocole : JSON/REST
- Persistence : mémoire, disque, cloud...
- Gestion de concurrence / cohérence
- Réplication, protocoles
- Langage d'implémentation, ...

Liste (incomplète) de systèmes noSQL

Document store (DS) :

- Collections de documents
- Natifs : CouchDB, **MongoDB**, TerraStore, ...
- Soft : eXist, Virtuoso

Tabular store (TS) :

- Tables
- Cassandra, **Hadoop** / Hbase, Hypertable (Bigtable)

Key-value store (KVS) :

- *Absence de schéma*
- DynamoDB, Voldemort, Azure Table Storage, MongoDB, Bigtable ...

Graph store (GS) :

- Graphes
- AllegroGraph, InfiniteGraph

Autres types de systèmes „noSQL“ : XML, Triplestore (RDF), oriéntés objets

Acteurs noSQL

- Amazon : DynamoDB, SimpleDB
- Microsoft : Azure Table Storage
- Google : BigTable, Datastore, GFS
- Apache : CouchDB, Cassandra, Hadoop / HBase
- Beaucoup de start-ups...

MongoDB (DS)

- JSON document store
- Protocole : BSON (binary JSON)
- MapReduce, langage orienté-objet (JSON)
- Écrit en C++

MongoDB

- MongoDB est un SGBD :
  - orienté documents
  - libre
  - scalable : réplication, auto-sharding
  - flexible : pas de schéma de données, full-text index
  - écrit en C++.
- Il fait partie de la mouvance NoSQL et vise a fournir des fonctionnalités avancées. Utilisée par Foursquare, bit.ly, Doodle, Disqus

Mongo DB

- Modèle logique
  - BD = ensemble de collections
  - Collection = ensemble de documents
    - Taille variable
    - Taille fixe (capped) : LRU
  - Document = objet BSON
- Modèle physique
  - Index : attributs, geo-spatial
  - MapReduce
  - Sharding (partitionnement horizontal)

Langage MongoDB

- Mise-à-jour
  - db.collection.insert()
  - db.collection.update()
  - db.collection.save()
  - db.collection.findAndModify()
  - db.collection.remove()
- Interrogation
  - db.collection.find()
  - db.collection.findOne()

Mongo DB : insert

db.collection.insert(<doc> [, <doc>]*)

```
> collection = db.contactinfo
> doc = { "nom" : "toto",
        "info" : {
                "twitter" : "@toto5646",
                "email" : "toto@upmc.fr" },
        "amis" : 87687,
        "photo" : BinData(...)
        "date_login" : new Date()}
> db.contactinfo.insert(doc)
```

MongoDB : update

db.collection.update(query, update, <upsert>, <multi>)

- query : requête
- update : modifications
- upsert (booléen) :
  - false (défaut) : update
  - true : update or insert if not exists
- multi (booléen) :
  - false (défaut) : maj de la 1ere occurrence trouvée
  - true : maj toutes les occurrence

MongoDB: update

- db.products.update({ item: "book", qty: { $gt: 5 } }, { $set: { x: 6 }, $inc: { y: 5} })

<>

- db.products.update({ item: "book", qty: { $gt: 5 } }, { x: 6, y: 15 })

- db.products.update({ item: "book", qty: { $gt: 5 } }, { $set: { x: 6, y: 15 } }, { multi: true })

=

- db.products.update({ item: "book", qty: { $gt: 5 } }, { $set: { x: 6, y: 15 } }, false, true)

- db.products.update({ item: "magazine", qty: { $gt: 5 } }, { $set: { x: 25, y: 50 } }, { upsert: true })

Mongo DB : requêtes

db.collection.find(<query>, <projection>)

db.collection.findOne(<query>, <projection>)

```
> db.inventory.find( {} )
> db.contactinfo.findOne({"nom" : "toto"});
> db.contactinfo.findOne({"contact.twitter" : "@toto5646"});
> db.contactinfo.find({"date_login" : {
        "$gt" : ISODate("2015-09-17T23:25:56.314Z"),
        "$lt" : ISODate("2014-09-17")}}).sort({nom :
1}).limit(10).skip(100)
```

MongoDB : requêtes

```
> db.employee.insert(
      {
            " name ": " John Smith ",
            " address ": {
            " street ": " Lily Road ",
            " number ": 32,
            " city ": " Owatonna ",
            "zip ": 55060
      },
      " hobbies ": [ " yodeling ", "ice skating " ]
})
>db. employee . FindOne (
      {" name ": " John Smith "})

> db. employee . find (
      {" address . city ": " Owatonna "},
      {" name ": 1})

> db. employee . find (
      {" hobbies ": {" $ne ": " yodeling "}})
```

```
> db. employee . find (
      {" address . city ": " Owatonna "},
      {" address.city ": 0, age : 1, name : 1})
```

MongoDB en Java

- Authentification (optionnel):

```
boolean auth = db. authenticate ( myUserName , myPassword );
```

- Liste des collections :

```
Set < String > colls = db. GetCollectionNames ();
for ( String s : colls ) {
        System . out . println (s); }
```

- Accès à une collection :

```
DBCollection coll = db. getCollection (" testCollection ")
```

MongoDB en Java

- Insertion d'un objet (document) :

```
 1 BasicDBObject doc = new BasicDBObject ();
 2
 3 doc . put (" name ", " MongoDB ");
 4 doc . put (" type ", " database ");
 5 doc . put (" count ", 1);
 6
 7 BasicDBObject info = new BasicDBObject ();
 8
 9 info . put ("x", 203);
10 info . put ("y", 102);
11
12 doc . put (" info ", info );
13
14 coll . insert ( doc );
```

MongoDB en Java

- Requêtes :

```
1 DBObject doc = collection . findOne (); // get first document
2 System . out . println ( dbObject );
```

```
1 DBCursor cursor = collection . find ();
2 while ( cursor . hasNext ()) {
3     System . out . println ( cursor . next ()); }
```

```
1 BasicDBObject query = new BasicDBObject ();
2 query .put(" number ", 5);
3 DBCursor cursor = collection . find ( query );
4 while ( cursor . hasNext ()) {
5     System . out . println ( cursor . next ()); }
```

MongoDB en Java

- Requêtes :

```
1 BasicDBObject query = new BasicDBObject ();
2 List < Integer > list = new ArrayList < Integer >();
3 list . add (9);
4 list . add (10);
5 query .put(" number ", new BasicDBObject (" $in ", list ));
6 DBCursor cursor = collection . find ( query );
7 while ( cursor . hasNext ()) {
8     System . out . println ( cursor . next ());
9 }
```

MongoDB en Java

- Requêtes :

```
1 BasicDBObject query = new BasicDBObject ();
2 query .put(" number ", new BasicDBObject (" $gt ", 5). append (" $lt ", 8));
3 DBCursor cursor = collection . find ( query );
4 while ( cursor . hasNext ()) {
5     System . out . println ( cursor . next ());
6 }
```

- Création d'un index :

```
coll . createIndex (new BasicDBObject ("i", 1)); // create index on "i", ascending
```

Parallélisation : MapReduce

Entrée :

- une (grande) *collection* d'objets (documents, données)
- un traitement / calcul ensembliste : compter, agréger, filtrer, indexer, *jointure*
- un grand nombre de machines connectées

Problème :

- paralléliser le traitement en utilisant efficacement les machines disponibles

Approche MapReduce

- Programme :
  - Séquence de deux appels de fonctions :
  - map1 → reduce1 → map2 → reduce2 → ...
- Coordinateur
  - Déploiement des données et des traitements dans le cluster
  - Coordination de l'exécution
- Avantage :
  - Parallélisation «automatique» de l'évaluation
  - Virtualisation de l'infrastructure

Parallélisation : MapReduce

Collection d'objets = collection de couples (Clés, Valeurs)

Deux fonctions *utilisateur* f et g :

- *map* (f, [(A, V)]) → (B,V')* :

  map (count_words, (url, doc)) → (mot, int)*

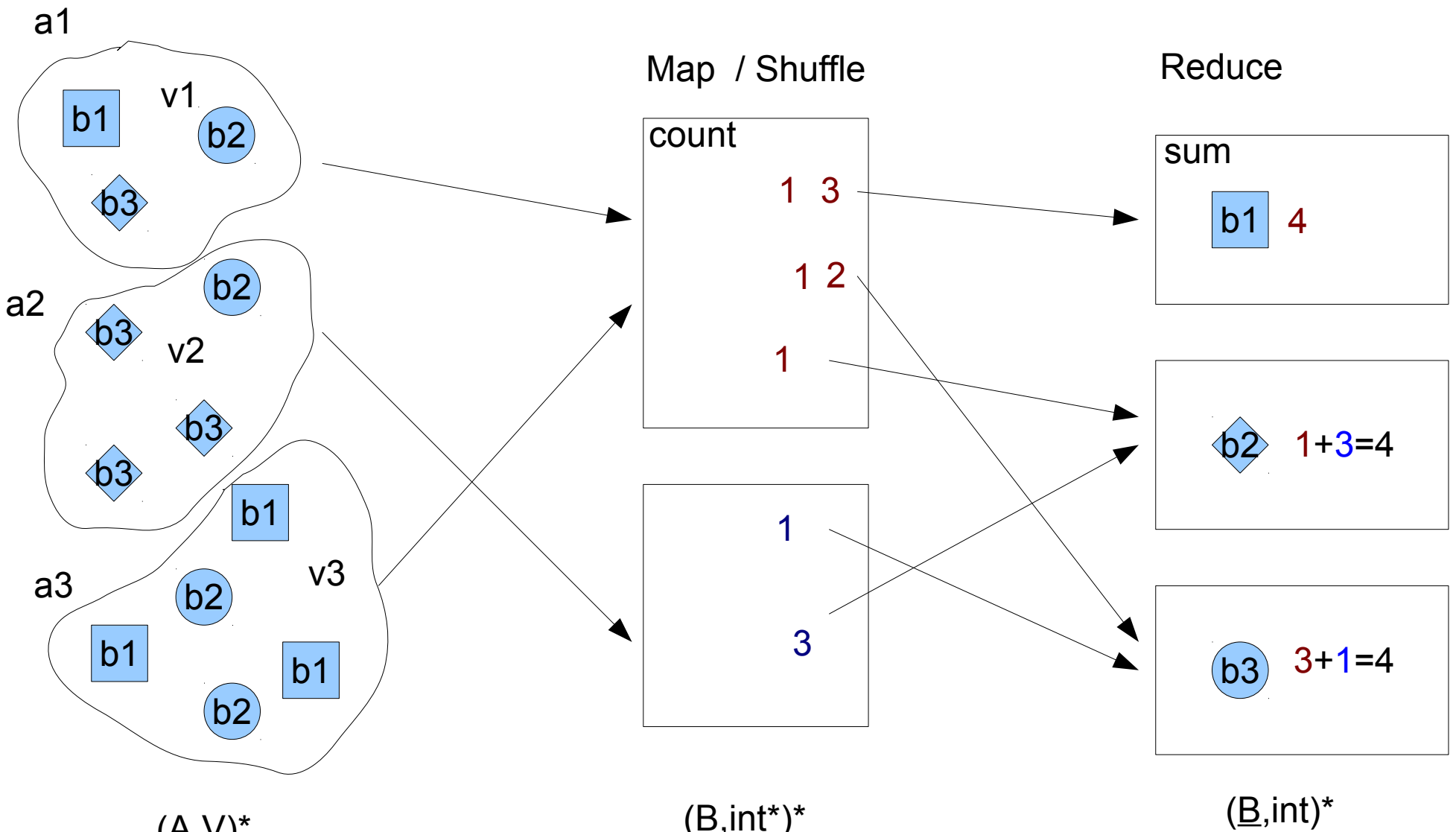
- *shuffle* (B,V')* → (B, V'*)* :

  shuffle (mot, int)* → (mot, int*)* *group by mot*

- *reduce* (g, (C',V'*)*) → (C,V'')* :

  reduce (sum, (mot, int*)*) → (mot, int)* *agrégation*

MapReduce (exemple)



MongoDB : MapReduce

```
db.runCommand(
 { mapreduce : <collection>,
   map : <mapfunction>,
   reduce : <reducefunction>
   [, query : <query filter object>]
   [, sort : <sorts the input objects using this key. Useful for optimization, like
              sorting by the emit key for fewer reduces>]
   [, limit : <number of objects to return from collection>]
   [, out : <see output options below>]
   [, keeptemp: <true|false>]
   [, finalize : <finalizefunction>]
   [, scope : <object where fields go into javascript global scope >]
   [, jsMode : true]
   [, verbose : true]
 }
);
```

MapReduce : exemple

Collection de commentaires :

```
{ username: "jones", likes: 20,
    text: "J'aime cette photo!" }
```

Fonction Map :

```
function() {
    emit( this.username,
          {count: 1, likes: this.likes} );
 }
```

Fonction Reduce :

```
function(key, values) {
    var result = {count: 0, likes: 0};
    values.forEach(function(value) {
      result.count += value.count;
      result.likes += value.likes;
    });
    return result;
}
```

- Map / shuffle : [(C,V)] → [(C',[V'])]
- Reduce : [(C',[V'])] → [(C',V')]

MapReduce incrémental

Sharding

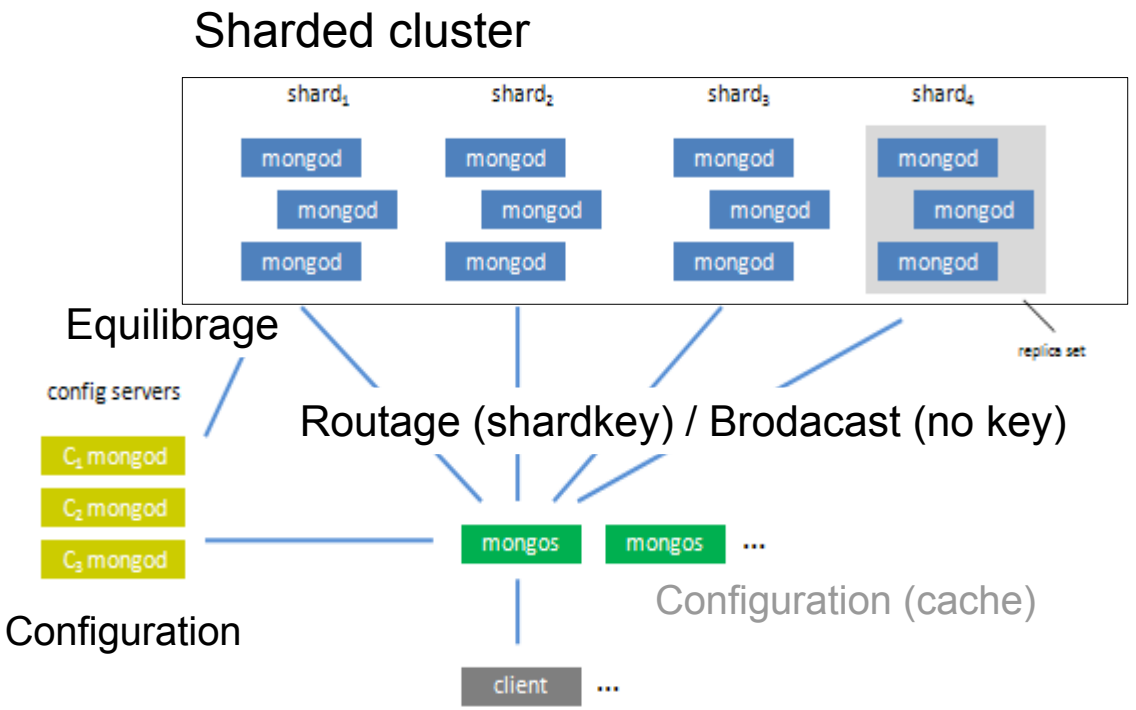
Partitionnement horizontal d'une BD (collection) :

- Réplication maître / esclave
- plusieurs milliers de nœuds
- ajout / suppression de nœuds (dynamique)
- équilibrage de charge automatique

MonoDB : Sharding

Notions :

- shard : traitement de requêtes (query, maj) sur un fragment de collection (identifié par shardkey)
- mongos : routage de requêtes
- config server : surveillance des shards (métadonnées)



Conclusion

- On a besoin de SQL et de NoSQL
  - NoSQL = not only SQL
  - Principe CAP
- Importance de noSQL
  - Analyse de données
  - Passage à l'échelle
  - Parallélisation / partionnement verticale

MongoDB en Java

```
1 import com. mongodb . Mongo ;
2 import com. mongodb .DB;
3 import com. mongodb . DBCollection ;
4 import com. mongodb . BasicDBObject ;
5 import com. mongodb . DBObject ;
6 import com. mongodb . DBCursor ;
7
8 Mongo m = new Mongo ();
9 // or
10 Mongo m = new Mongo ( " localhost " );
11 // or
12 Mongo m = new Mongo ( " localhost " , 27017 );
13
14 DB db = m. getDB ( " mydb " );
```