Technologies du Web

Laure Soulier — laure.soulier@lip6.fr
Slides de Ludovic DENOYER

UPMC

31 janvier 2017

NoSQL

Historique

Historique

- NoSQL : Not Only SQL
- 2004 : Big Table Google
- 2008 : Cassandra Facebook
- 2009 : lancement d'une communauté de développement logiciels open source NoSQL

Spécificités

- Manipulation de données peu (pas) structurées, schéma flexible, sans modèle pré-défini
- Manipulation de données volumineuses
- Haute disponibilité
- Capacités élevées de lectures/écritures

SQL vs. NoSQL

	Système SQL	Système NoSQL
Finalité	Système de gestion de bases de données	Système de gestion de données
Indépendance données/ programmes	Garantie (séparation du niveau logique/physique)	Non garantie
Schéma des données	Prédéfini, relativement fixe, peu flexible	Pas besoin de fixer le schéma, schéma dynamique, schema-less, schema-later
Eclatement/répartition des données	Eclatement en respect du principe de normalisation, choix de la répartition des données difficile	Principe de proximité physique des données reliées logiquement
Transaction	Respect des propriétés ACID	Relâchement des propriétés ACID
Valeurs d'attributs	Trois états : vrai, faux, NULL	Variable : pas de valeurs, pas d'état NULL, ...

NoSQL

NoSQL

NoSQL (Not Only SQL), apparu en 2009, et comme son nom l'indique une alternative au langage SQL et au modèle relationnel de base de données que vous avez l'habitude d'utiliser. L'idée est de proposer une architecture souple et puissante avec une forte disponibilité et de faibles contraintes.

Particulièrement à la mode, la majorité des grandes entreprises du web abandonnent leurs bases de données traditionnelles et portent leur propre projet NoSQL : Facebook et Twitter utilisent Cassandra (de la Fondation Apache), Amazon SimpleDB, LinkedIn Voldemort, etc.

Théorème CAP

Propriétés fondamentales pour les systèmes distribués

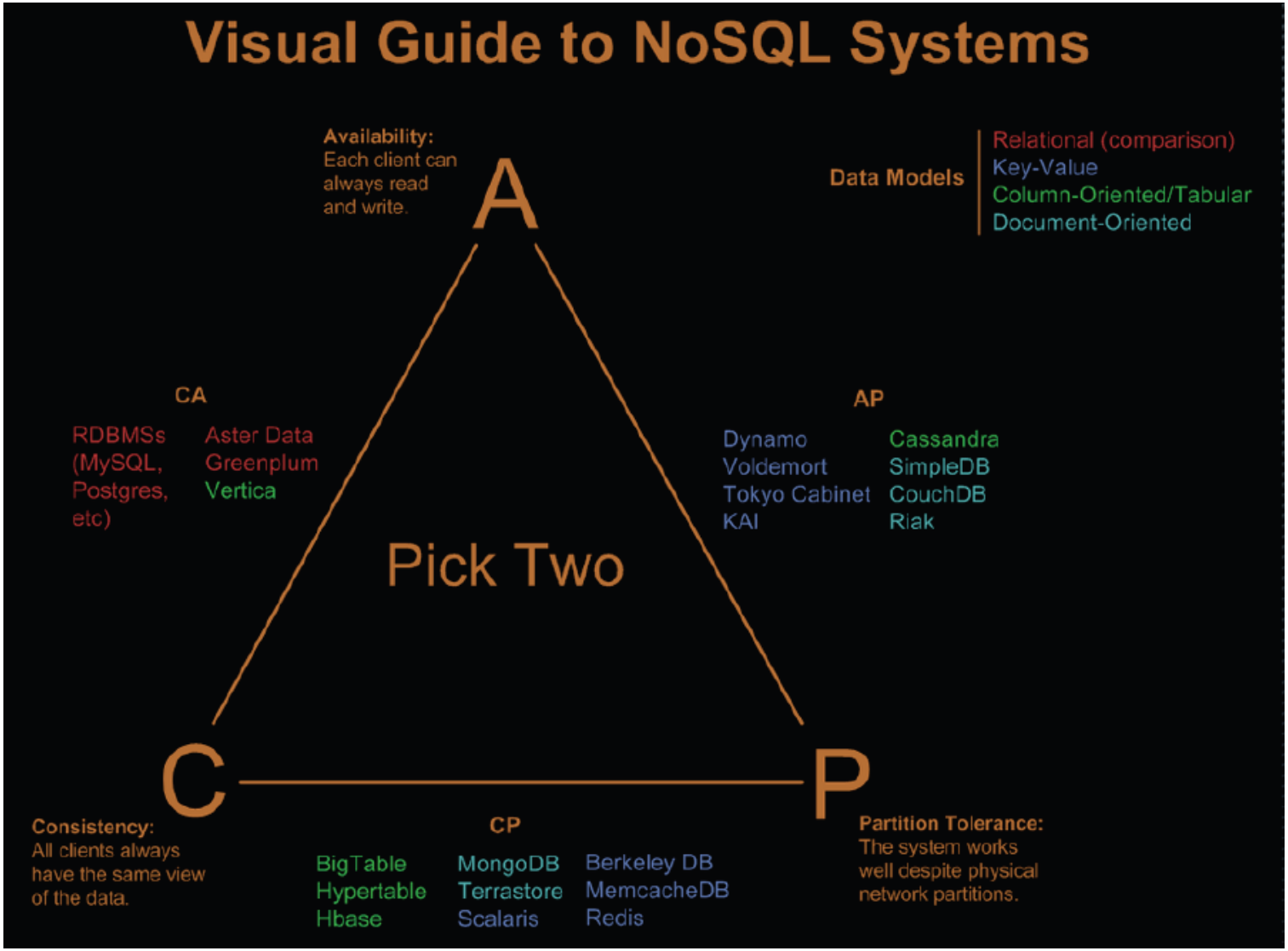
- Coherence : tous les n ?uds du système voient exactement les mêmes données au même moment
- Availability (Disponibilité) : garantie que toutes les requêtes reçoivent une réponse ;
- Partition tolerance (Résistance au partitionnement) : sauf coupure totale du système, aucune panne ne peut empêcher le système de fonctionner normalement

Dans un système distribué, seules deux propriétés sur trois peuvent être assurées à un instant t.

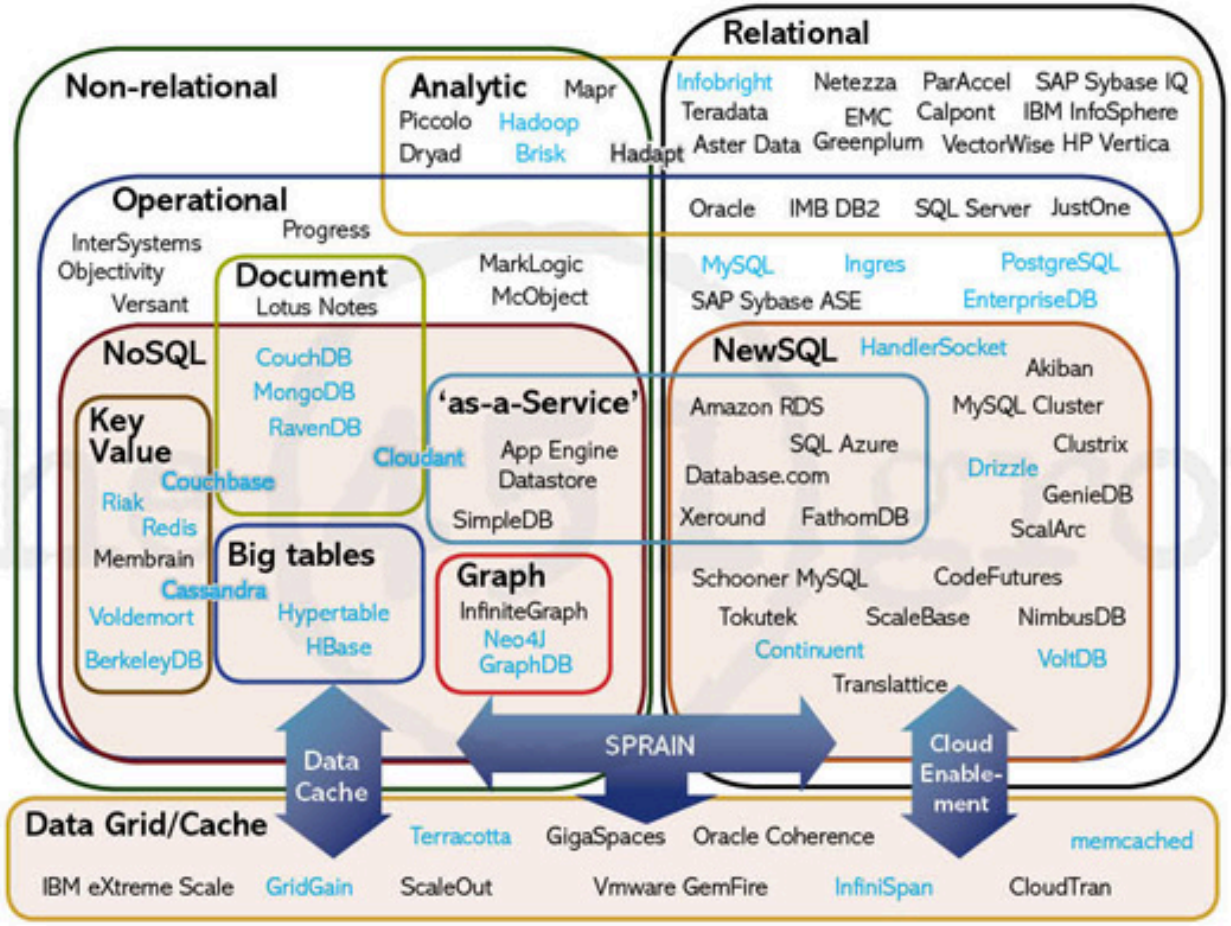
Exemples (wikipedia)

Soit A et B deux utilisateurs, soit N1 et N2 deux noeuds.
• Si A modifie une valeur sur N1, alors pour que B voie cette valeur sur N2 il faut attendre que N1 et N2 soient synchronisés (cohérence).
Exemple d'arbitrage : deux utilisateurs qui font une même recherche peuvent obtenir des résultats différents, mais cet inconvénient est moins grave que de ne pas avoir de résultats du tout.

NoSQL


Visual Guide to NoSQL Systems
Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

Typologie des modèles de données NoSQL



http://blogs.the451group.com/information_management/2011/04/15/nosql-newsql-and-beyond/

Typologie des modèles de données NoSQL

- Bases de données clé-valeur (1/4)
  - Similaire à un tableau associatif / table de hachage
  - Clé : permet d'accéder à la valeur
  - Valeur :
    - Simple chaîne de caractères ou objet sérialisé
    - Absence de structure ou de typage

Clé	Valeur
k1	v1
k2	v2
…	…

Exemples :
- Réseaux sociaux :
  id utilisateur → liste de ses amis
- Livres :
  Id livre → informations (auteur, éditeur, …)
- Journaux
  Date → liste des évènements

Typologie des modèles de données NoSQL

- Bases de données clé-valeur (2/4)
  - Opérations (CRUD)
    - Create : Création d'un nouvel objet
    - Read : Lecture d'un objet à partir de la clé
    - Update : Mise à jour d'une valeur à partir de la clé
    - Delete : Suppression d'un objet à partir de la clé
  - Indexation
    - Index primaire sur la clé
    - Pas d'indexation secondaire (autres attributs)

Typologie des modèles de données NoSQL

❑Bases de données clé-valeur (3/4)

- Avantages
  - Performance élevée en lecture et écriture
  - Modèle de données simple
  - Passage à l'échelle facile

- Inconvénients
  - Interrogation limitée : obligation de connaître la clé
  - Structure trop simple pour les données complexes
  - Complexité déportée sur l'application : les traitements faits de la requête SQL doivent maintenant être faits par l'application

- Solutions : Amazon Dynamo, FundationDB, Redis, Riak, …


Typologie des modèles de données NoSQL

❑ Bases de données clé-valeur (4/4)

- Exemple :

Gestion du serveur ‘azteka’ de la fac avec redis

– Créer un objet : stocker le nom du serveur
  SET server:name ‘’azteka’’
– Lire l’objet : récupérer le nom du serveur
  GET server:name → ‘’azteka ’’
– Supprimer un objet
  DEL server:name
– Modification d’un objet :
  SET server:name ‘’azteka2’’ → écrase la valeur précédente

Typologie des modèles de données NoSQL

❑Bases de données orientées colonne (1/5)

- Modèle le plus d'un SGDB relationnel
- Nombre de colonnes dynamiques (pas de valeurs nulles)
- Une « clé cachée » *(rowkey)* pour chaque objet, triée par ordre lexicographique
- Les colonnes sont traitées indépendamment
- Les lignes peuvent être reconstituées plus tard

Typologie des modèles de données NoSQL

❑ Bases de données orientées colonne (2/5)

Nom (2Go)	**Note (1Go)**
Durand	10
Dupond	15
Dupoint	14

BD orientée ligne :
1:Durand,10 ; 2:Dupond,15 ; 3:Dupoint,14

BD orientée colonne :
Durand:1 ; Dupond:2 ; Dupoint:3
10:1 ; 15:2 ; 14:3

→ Traitement par tableau des valeurs de colonnes
→ Améliore les performances lorsque traitement sur un nombre réduit de colonnes

Exemple : Moyenne des notes
- BD orientée ligne : chargement de toute la table (3Go)
- BD orientée colonne : chargement de l'attribut Note (1Go)

Typologie des modèles de données NoSQL

- Bases de données orientées colonne (3/5)
  - Terminologie :
    - Clés
      - » Point d'entrée de la donnée.
    - Familles de colonnes
      - » Structure définie
      - » Regroupement par lignes de plusieurs colonnes
      - » Utilisées pour l'indexation, par exemple
    - Colonnes
      - » Entités de base (attribut)
      - » Associées à une paire clé-valeur
      - » Pas de structure du schéma de la valeur
    - Timestamp
      - » Associé aux colonnes
      - » Gestion de la cohérence de la BD par rapport aux versions
  - Pour accéder à une donnée

    Clé → Famille → Colonne → Timestamp (par défaut : le plus récent)

Typologie des modèles de données NoSQL

❑Bases de données orientées colonne (4/5)

- Avantages
  - Bon passage à l'échelle
  - Indexation naturelle (colonnes)
  - Gestion des versions → résultats en temps réels

- Inconvénients
  - Lecture difficile pour les données complexes
  - Maintenance : ajout/suppression/regroupement des colonnes
  - Les requêtes dépendent des données

- Solutions : Hbase (Open Source de BigTable utilisé par Google), Cassandra (Facebook), Hyperbase (Amazon)

Typologie des modèles de données NoSQL

❑Bases de données orientées colonne (5/5)

- Exemple :
  - Définir la structure de la table
    CREATE ‘serveurs’, {NAME => ‘nom’, VERSIONS => 3}, {NAME => ‘config’, VERSIONS => 2} , {SPLITS => [‘1000’,’2000’,’3000’,’4000’]}
  - Créer un objet : stocker le serveur azteka
    PUT ‘serveurs’, ‘1’, ‘nom’, ‘azteka’, ‘admin:nom’, ‘soulier’, ‘config’, ‘unix’
  - Lire l’objet : récupérer le nom du serveur tel qu’il a été inséré lors de la version 1
    GET ‘serveurs’, ‘1’ ,{COLUMN => ‘nom’ , VERSIONS =1}
  - Supprimer un objet
    DELETEALL ‘serveurs’ ,’1’
  - Suppression de la table (désactivation, puis suppression)
    DISABLE ‘serveurs’
    DROP ‘serveurs’

Typologie des modèles de données NoSQL

- Bases de données orientées document (1/3)
  - Stockage d'une collection de documents
  - Basé sur les paires clés-valeurs
    - Clé : identifie le document
    - Valeur : document en format semi-structuré hiérarchique (XML, JSON)

- Exemple de fichier JSON

```
{
"card":"2",
"numbers":{
        "Conway":[1,11,21],
        "Fibonacci":[0,1]
        }
}
```

- Exemple de fichier XML

```
<xml>
  <card>2</card>
  <numbers>
        <Conways>
          <Conway>1</Conway>
          <Conway>11</Conway>
          <Conway>21</Conway>
        </Conways>
        <Fibonaccis>
          <Fibionacci>0</Fibionacci>
          <Fibionacci>1</Fibionacci>
        </Fibionaccis>
  </numbers>
</xml>
```

Typologie des modèles de données NoSQL

❑Bases de données orientées document (2/3)

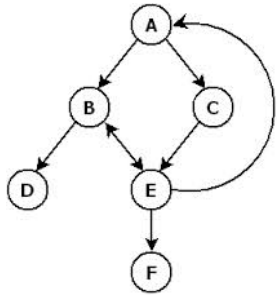
- Avantages
  - Modèle de données simple mais puissant (documents hiérarchiques)
  - Pas de maintenance pour l'ajout/suppression des documents
  - Requêtes assez complexes (simplifie l'application)
- Inconvénients
  - Lenteur pour les grandes requêtes
  - Interconnexion des données difficile
- Solutions : MongoDB, CouchDB, CouchBase

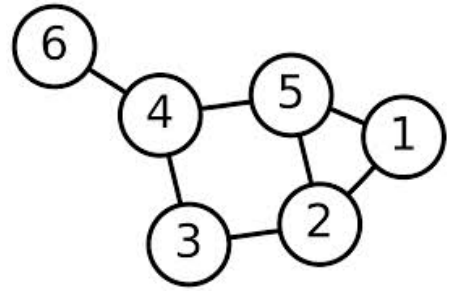
Typologie des modèles de données NoSQL

- Bases de données orientées document (3/3)
  - Exemple avec mongoDB
    - Création d'une collection (table en SGBDR)
      db.createCollection(''macollection'')
    - Insertion d 'un document
      db.macollection.insert({id : '1', titre : 'cours M2', année : '2014-2015'})
    - Lecture d'un document
      db.macollection.find({},{'titre':'cours M2')
    - Mise à jour d'un document
      db.macollection.update({'titre': 'cours M2'},{$set:{'année : '2015-2016'}})
    - Suppression d'un document
      db.macollection.remove({'titre': 'cours M2'})
    - Suppression d'une collection
      db.macollection.drop()

Typologie des modèles de données NoSQL

- Bases de données orientées graphe (1/3)
  - Stockage de données complexes avec des relations
  - Basé sur la théorie des graphes
    - Nœuds : moteur de stockage
    - Relations : description des arcs
  - Application : traitement des données des réseaux sociaux





Typologie des modèles de données NoSQL

❑Bases de données orientées graphe (2/3)

- Avantages
  - Modèle de données puissant
  - Stockage et requêtage efficace pour les données complexes et interconnectées
- Inconvénients
  - Fragmentation difficile
- Solutions : Neo4j, OrientDB, InfiniteGraph, …

Typologie des modèles de données NoSQL

❑*Bases de données orientées graphe (3/3)*

- Exemple avec Neo4j
  - Création d'un nœud

```
CREATE n ={nom : 'Milou', ville : 'Paris'};
```

  - Lecture d'un nœud

```
START n=node(1) return n;
```

  - Création d'un nœud avec une relation

```
START Milou = node(1)
CREATE
    Tintin = {nom : 'Tintin', ville : 'Paris'},
    Castafiore = {nom : 'Castafiore', ville : 'Paris'},
    Milou-[r:AMI]->Tintin
RETURN
    Tintin;
```

MongoDB

MongoDB (Wikipédia)

MongoDB est un système de gestion de base de données :

- orientée documents,
- libre,
- montant bien en puissance (scalable),
- à performance raisonnable,
- ne nécessitant pas de schéma prédéfini des données,
- écrit avec le langage de programmation C++.

Il fait partie de la mouvance NoSQL et vise à fournir des fonctionnalités avancées. Utilisée par Foursquare, bit.ly, Doodle, Disqus

MongoDB


foursquare
twitter
bit.ly
sourceforge
reddit
github
SOCIAL CODING
The New York Times
rackspace
HOSTING
Hot Potato
cloudkick
Eventbrite
Events Made Easy
facebook
digg
fotopedia.com
CollegeHumor
MongoDB
Cassandra

MongoDB

Autres propriétés

- Document-oriented storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Rich, document-based queries.
- Map/Reduce
- GridFS
- Commercial Support

MongoDB

Drivers

- C
- C++
- Erlang
- Haskell
- Java
- Javascript
- .NET (C# F#, PowerShell, etc)
- Perl
- PHP
- Python
- Ruby
- Scala

MongoDB : Principes

- Une instance de Mongodb peut contenir une ou plusieurs base de données
- Une base de données peut avoir une ou plusieurs **collections**
  - Equivalent des tables dans les bases classiques
- Une collection peut avoir de zéro à plusieurs **documents**
  - Les documents d'une même collection n'ont pas obligatoirement le même "schéma" (champs)
  - Les documents correspondent aux enregistrement (tuples)
  - Des documents peuvent contenir des documents (structure hiérarchique)
  - Chaque document possède une clef unique
- MongoDB stockent l'information au format BSON (Binary JSON)

MongoDB : Propriétés

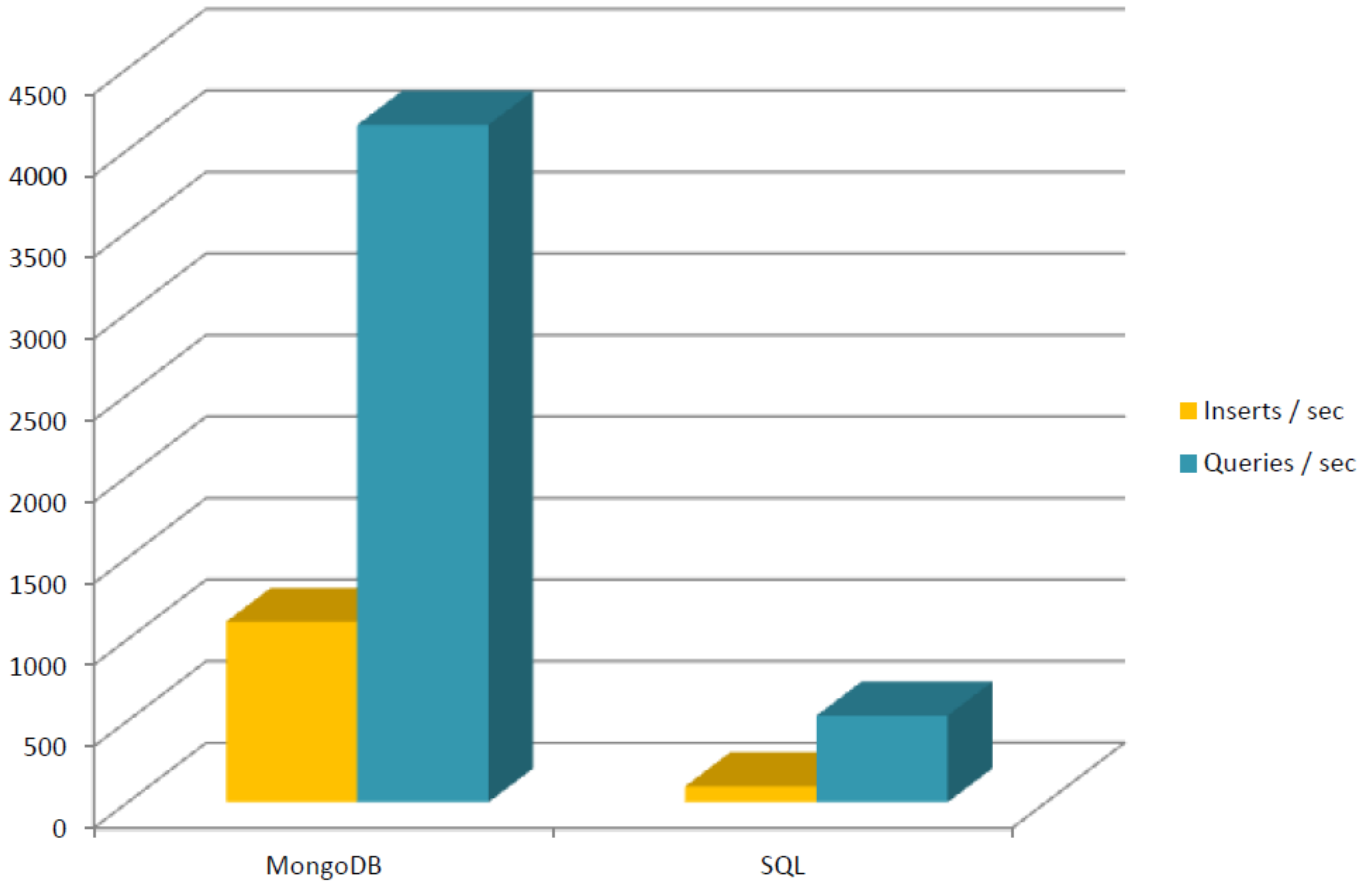
Avantages

- Requêtes "faciles"
- Fait sens avec la plupart des applications Web
- Facilité d'intégration des données

Inconvénients

Pas bien adapté pour les systèmes transactionnels complexes

MongoDB


4500
4000
3500
3000
2500
2000
1500
1000
500
0
MongoDB
SQL
Inserts / sec
Queries / sec

MongoDB

Un document :

```
user = {
          name: "Z",
          occupation: "A scientist",
          location: "New York"
        }
```

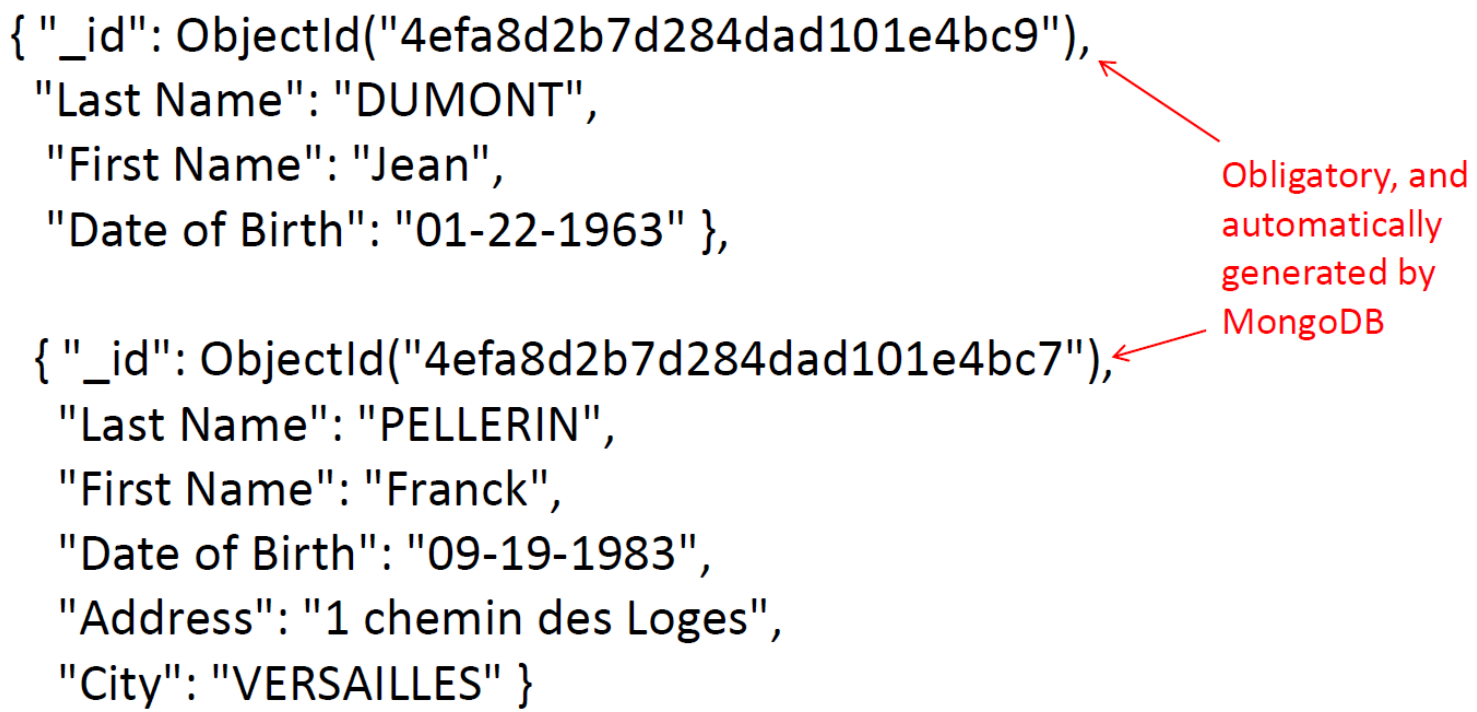
MongoDB

Un document :

```
{
" name ": " John Smith ",
" address ": {
" street ": " Lily Road ",
" number ": 32,
" city ": " Owatonna ",
"zip ": 55060
},
" hobbies ": [ " yodeling ", "ice skating " ]
}
```

MongoDB

Une collection :



MongoDB

Imaginons que l'on veuille modéliser un blog :

Mauvais design

```
post = {
  id: 150,
  author: 100,
  text: 'This is a pretty awesome post.',
  comments: [100, 105, 112]
}
author = {
  id: 100,
  name: 'Michael Arrington'
  posts: [150]
}
comment = {
  id: 105,
  text: 'Whatever this sux.'
}
```

MongoDB

Autre design

```
post = {
  author: 'Michael Arrington',
  text: 'This is a pretty awesome post.',
  comments: [
              'Whatever this post sux.',
              'I agree, lame!'
            ]
}
```

Pourquoi celui-ci est-il meilleur ?

MongoDB

```
location1 = {
        name: "10gen HQ”,
        address: "17 West 18th Street 8th Floor”,
        city: "New York”,
        zip: "10011”,
        latlong: [40.0,72.0],
        tags: [“business”, “cool place”],
        tips: [
                {user:"nosh", time:6/26/2010, tip:"stop by for office hours on
                Wednesdays from 4-6pm"},
                 {.....},
                ]
}
```

Example queries:

- db.locations.find({latlong:{$near:[40,70]}})
- db.locations.find({zip:”10011”, tags:”business”})
- db.locations.find({zip:”10011”}).limit(10)

MongoDB

```
user1 = {
        name: “nosh”
        email: “nosh@10gen.com”,
        .
        .
        .
        checkins: [{ location: “10gen HQ”,
            ts: 9/20/2010 10:12:00,
            …},
            …
         ]
}
```

MongoDB

- SQL
  - `INSERT INTO t (fn, ln) VALUES ('John', 'Doe')`
- MongoDB
  - `db.t.insert({fn:'John', ln: 'Doe'})`

MongoDB

- SQL
  - `UPDATE t SET ln='Smith' WHERE ln='Doe'`
- MongoDB
  - `db.t.update({ln:'Doe'},{$set:{ln:'Smith'}})`
- Only 1 doc. is updated by default
  - Specify `{multi:true}` for multi-doc updates
  - `db.t.update({ln:'Doe'},{$set:{ln:'Smith'}}, {multi:true})`

MongoDB

- SQL
  - `DELETE FROM t WHERE fn='John'`
- MongoDB
  - `db.t.remove({fn:'John'})`

MongoDB

- SQL
  - `DROP TABLE t`
- MongoDB
  - `db.t.drop()`

MongoDB

- SQL
  - `SELECT * FROM t`
- MongoDB
  - `db.t.find()`
- SQL
  - `SELECT id, name FROM t`
- MongoDB
  - `db.t.find({},{name:1})`

MongoDB

- SQL
  - `SELECT * FROM t WHERE fn='John'`
- MongoDB
  - `db.t.find({ln:'John'})`
- SQL
  - `SELECT id, age FROM t WHERE fn='John' AND ln='Smith'`
- MongoDB
  - `db.t.find({fn:'John',ln:'Smith'},{age:1})`

MongoDB

- SQL
  - ```
    SELECT country FROM t
    WHERE fn='John'
    ORDER BY age DESC LIMIT 10
    ```
- MongoDB
  - ```
    db.t.find(
      {fn:'John'},
      {country:1, _id:0}
    ).sort({age:1}).limit(10)
    ```

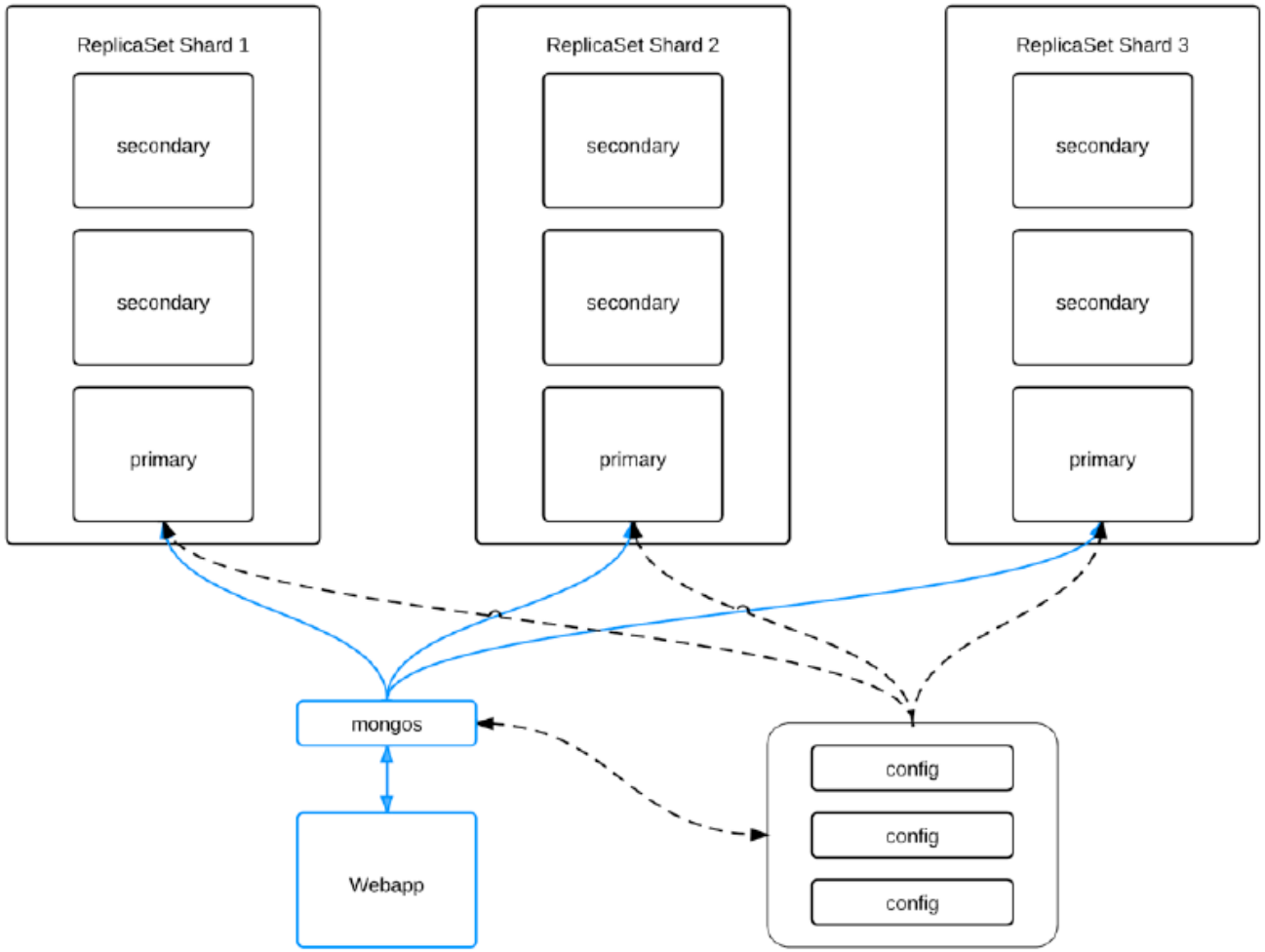
MongoDB

- SQL
  - `SELECT fn,ln FROM t`
    `WHERE age > 30`
- MongoDB
  - `db.t.find(`
    `{age:{$gt:30}},`
    `{fn:1, ln:1, _id:0}`
    `)`

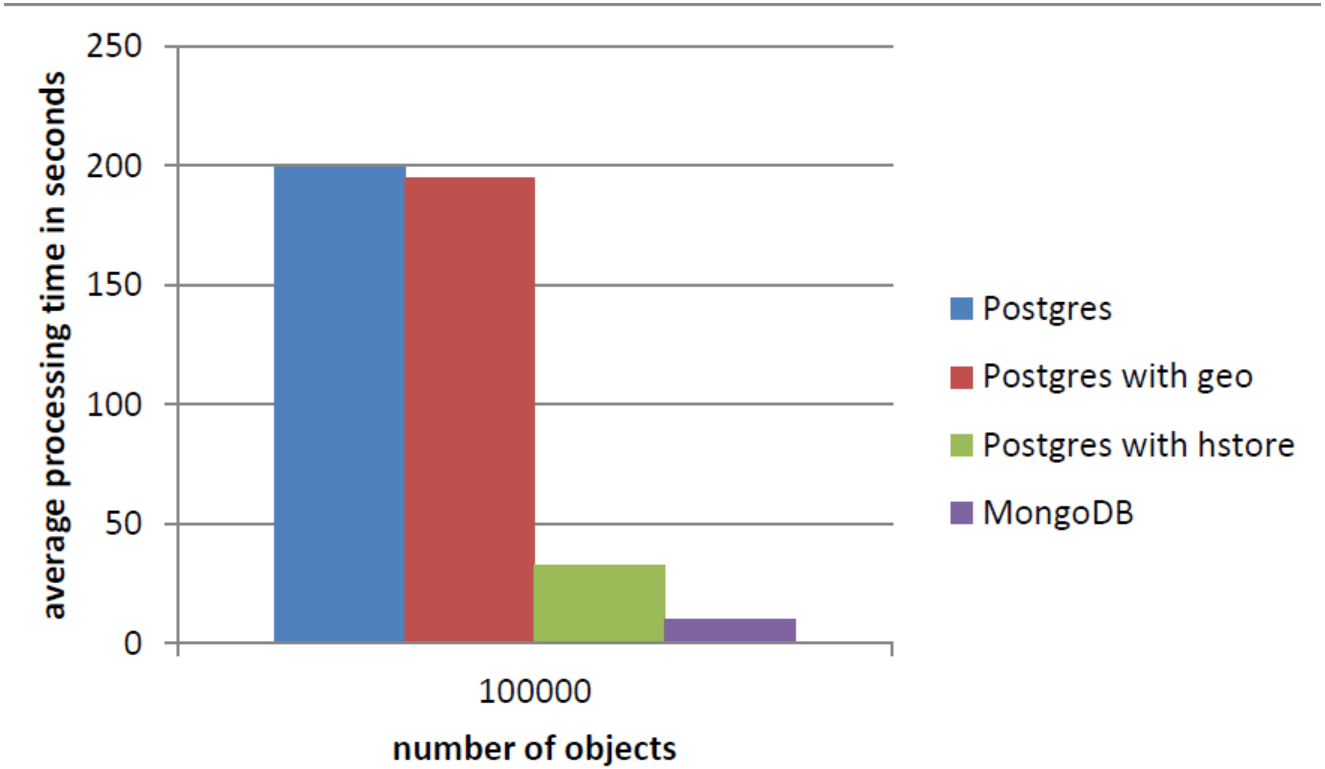
Sharding

- Le sharding consiste en la répartition des données en plusieurs instances de bases de données, selon un critère donné.
- La montée en charge horizontale est plus simple à mettre en oeuvre : il est plus facile et moins coûteux d'acheter de nouveaux serveurs plutôt que d'augmenter la capacité d'un seul serveur.
- Un shard est un serveur mongo normal : il peut contenir des collections shardées ou non, et peut faire partie d'un replica set.
- Il est important de bien choisir ses critères de sharding pour éviter une trop grande disparité.

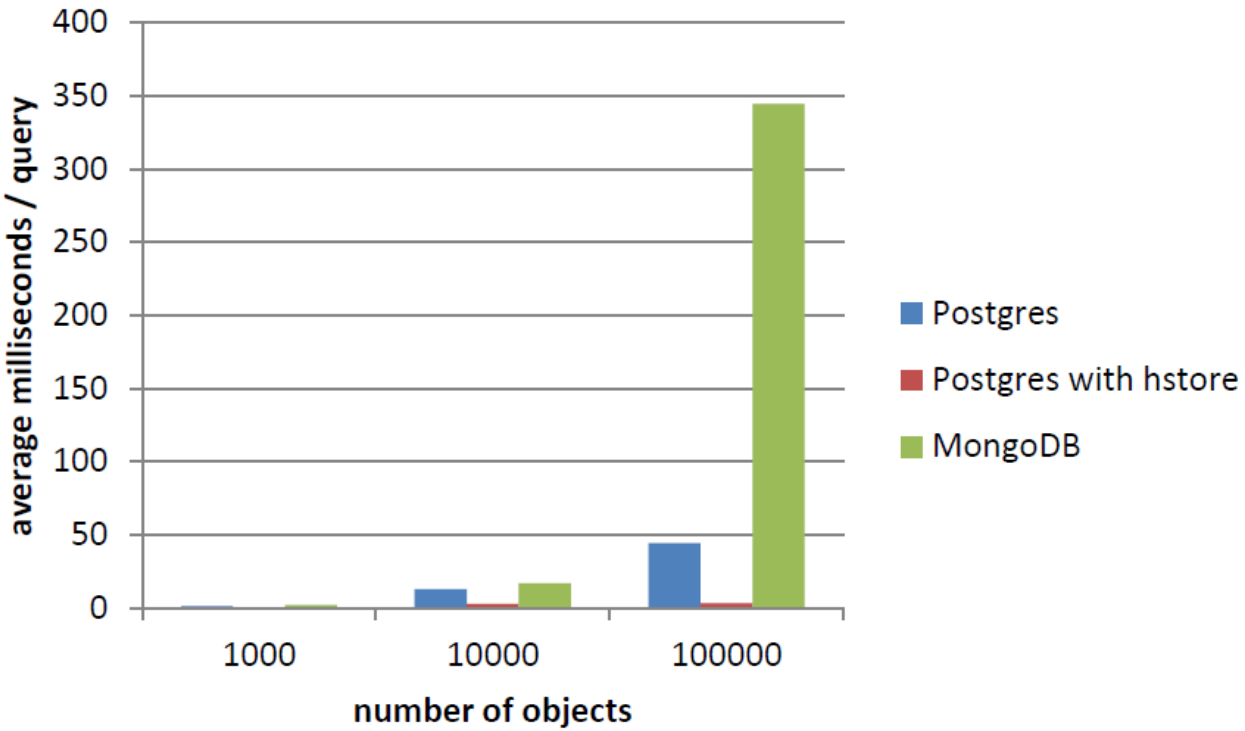
MongoDB


ReplicaSet Shard 1
secondary
secondary
primary
ReplicaSet Shard 2
secondary
secondary
primary
ReplicaSet Shard 3
secondary
secondary
primary
mongos
Webapp
config
config
config

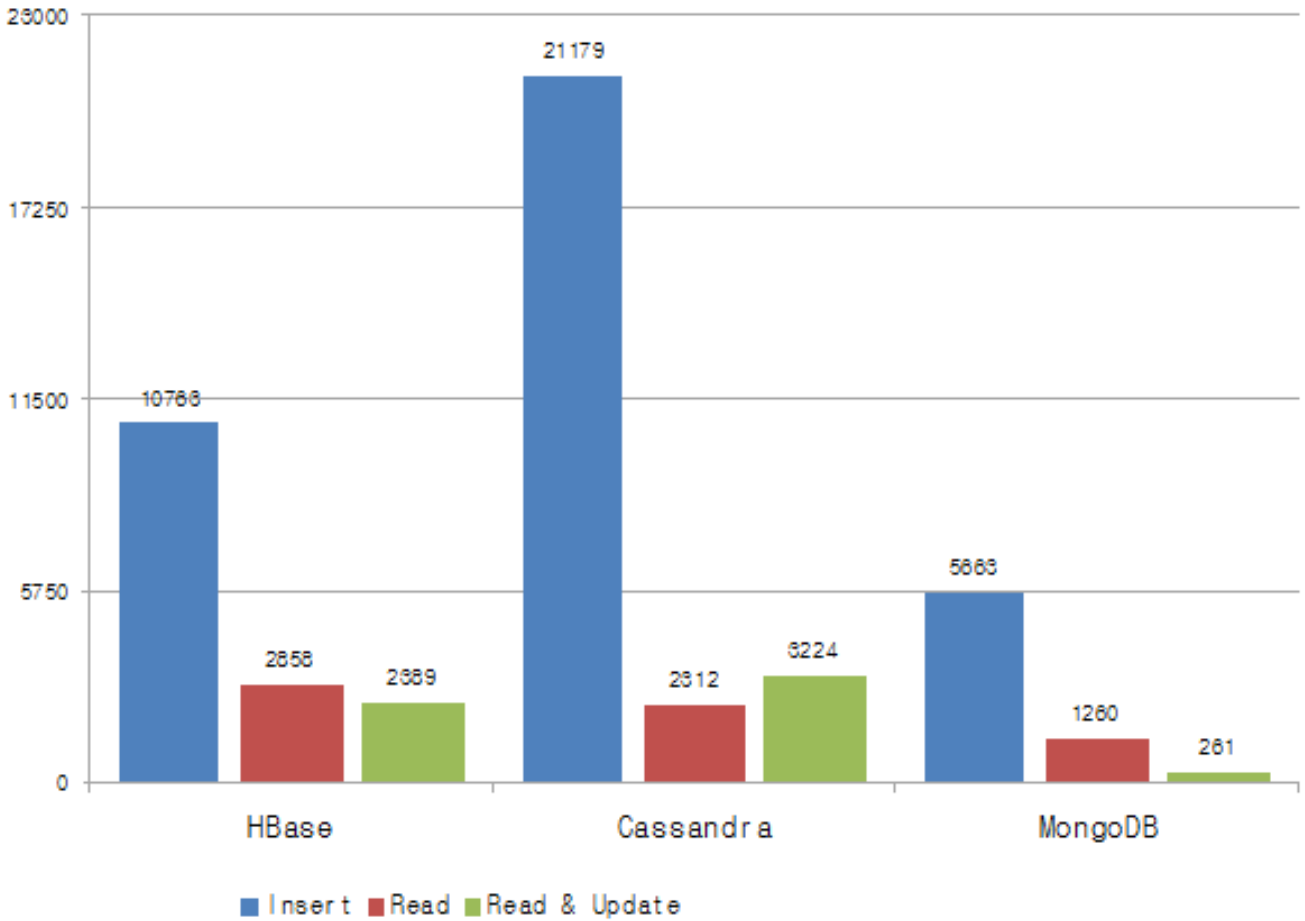
MongoDB


250
200
150
100
50
0
average processing time in seconds
100000
number of objects
Postgres
Postgres with geo
Postgres with hstore
MongoDB

MongoDB


400
350
300
250
200
150
100
50
0
average milliseconds / query
1000
10000
100000
number of objects
Postgres
Postgres with hstore
MongoDB

MongoDB


28000
17250
11500
5750
0
10766
2858
2389
21179
2312
3224
5563
1260
261
HBase
Cassandra
MongoDB
Insert
Read
Read & Update

MongoDB en JAVA

Nouvelle Connnection

```
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" );
```

Le Pooling de connection est automatiquement géré

MongoDB en JAVA

Authentification (optionnel)

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

MongoDB en JAVA

Liste des collections (tables)

```
1  Set<String> colls = db.getCollectionNames();
2
3  for (String s : colls) {
4      System.out.println(s);
5  }
```

MongoDB en JAVA

Utilisation des collections (tables)

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

MongoDB en JAVA

Insertion

```
{
    "name" : "MongoDB",
    "type" : "database",
    "count" : 1,
    "info" : {
                x : 203,
                y : 102
              }
}
```

MongoDB en JAVA

Insertion

```
BasicDBObject doc = new BasicDBObject();

        doc.put("name", "MongoDB");
        doc.put("type", "database");
        doc.put("count", 1);

        BasicDBObject info = new BasicDBObject();

        info.put("x", 203);
        info.put("y", 102);

        doc.put("info", info);

        coll.insert(doc);
```

MongoDB en JAVA

Requête

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

MongoDB en JAVA

Requête

```
1   DBCursor cursor = collection.find();
2       while(cursor.hasNext()) {
3           System.out.println(cursor.next());
4       }
```

MongoDB en JAVA

Requête

```
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());
6       }
```

MongoDB en JAVA

Requête

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

MongoDB en JAVA

Requête

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

MongoDB en JAVA

Requête

```
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  }
```

MongoDB en JAVA

Requête

```
1  BasicDBObject query5 = new BasicDBObject();
2  query5.put("number", new BasicDBObject("$ne", 8));
3  DBCursor cursor6 = collection.find(query5);
4  while(cursor6.hasNext()) {
5      System.out.println(cursor6.next());
6  }
```

Création d'un index

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

MongoDB en JAVA

Requête

```java
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.List;
import com.mongodb.*;

/**
 * Java MongoDB : Query document
 */
public class QueryDocumentApp {
    public static void main(String[] args) {
        try {
            Mongo mongo = new Mongo("localhost", 27017);
            DB db = mongo.getDB("yourdb");

            // get a single collection
            DBCollection collection = db.getCollection("dummyColl");

            // insert number 1 to 10 for testing
            for (int i = 1; i <= 10; i++) {
                collection.insert(new BasicDBObject().append("number", i));
            }
        //....
            System.out.println("Done");
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (MongoException e) {
            e.printStackTrace();
        }
    }
}
```

Conclusion

Vous savez...

- ...utiliser MongoDB
- ...en JAVA

Il faut...

- ...finir le MySQL (servlet User + Friends)
- ...intégrer MongoDB au site actuel (servlet MEssages)
  - ...Définir un "format" de sotckage des commentaires
  - ...Implémenter les servlets de "recherche" des commentaires récents

Vous ne savez pas...

- ...faire un moteur par recherche de mots-clefs
- à base de Map/Reduce
- ... plus tard