

NoSQL - SGBD distribués et CouchDB

I.Mougenot

Espace-Dev UM

Dev∂LR 2017

Les motivations autour du NoSQL (Not Only SQL)

Recouvre différentes initiatives complémentaires aux modèles relationnels et relationnels-objets

- 1 évolution du Web, sources de données ouvertes (LOD), impulsion Google, Facebook, Amazon, Twitter, ...
- 2 ↗ volume des données ↗ interconnexion des données
- 3 limites des bases de données relationnelles face à de nouveaux besoins :
 - flexibilité : schémas très ouverts : nombreuses entités et associations entre ces entités
 - adaptabilité : évolutions très fréquentes des schémas
 - des milliers voire des millions d'utilisateurs

Quand passer par un système NoSQL ?

Alternatives au relationnel dans des cas de figure ciblés

- recours fréquent à de l'évolution de schémas
 - entités munies de diverses caractéristiques souvent non renseignées
 - nombreuses associations avec des multiplicités 1..* aux extrémités
 - attributs composites
- un flux transactionnel (lecture, écriture) très élevé
- données distribuées dès l'origine (mondialisation)

NoSQL : se démarquer des SGBD relationnels

Réplication et partitionnement : deux techniques qui peuvent se combiner

- schémas normalisés vus comme des sophistications inutiles au détriment de l'efficacité
- modèle transactionnel et propriétés ACID : proposer une alternative moins exigeante
- passage à l'échelle par ajout de serveurs au niveau de l'architecture physique : diminuer le temps de réactivité lors de l'afflux de nouveaux usagers, de nouvelles transactions à servir
- systèmes distribués et mécanismes de tolérance aux pannes : fragmentation des schémas et réplication, médiateur, entrepôt de données ...

Passage à l'échelle ou scalabilité

Capacité de l'architecture à s'adapter à une montée en charge (nouveaux usagers, nouvelles transactions) sans besoin de refonte des applications

- scalabilité horizontale (scaling out) : ajouter des serveurs (noeuds) avec des mécanismes de répartition de charge ← NoSQL
- scalabilité verticale (scaling up) : rendre plus performant un serveur : ajout de processeurs (CPU), barrettes mémoire (RAM), disques secondaires, cartes réseaux ...

Scalabilité horizontale

Etablir une relation linéaire entre les ressources ajoutées et l'accroissement des performances

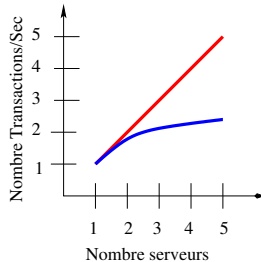


Figure: 1 serveur : 100 transactions/s ; 2 serveurs : 200 transactions/s

SGBDR et besoins applicatifs à large échelle

Limites face aux besoins des applications à large échelle sur le Web
(à partir Web 2.0)

- partitionnement : les schémas fragmentés (fragmentations horizontale, verticale, hybride) distribués sur l'ensemble des partitions doivent être des fragments d'un seul schéma de données initial
- réplication sur différents noeuds : les fondements OLTP (On Line transactional processing) imposent de maintenir une intégrité forte sur les données, dans une application faisant appel à de nombreux noeuds, la disponibilité des données va être pénalisée (surtout si les transactions impliquent de nombreuses écritures).

systemes NoSQL : grands principes

- **Simplicité**
- **Flexibilité**
- **Efficacité**
- **Passage à l'échelle** : gros volumes de données distribués et interconnectés
 - partitionnement dynamique - sharding (partitionnement horizontal + plusieurs co-occurrences de schémas)
 - réplication à large échelle
 - architecture décentralisée

Complémentarité des systèmes NoSQL

One size doesn't fit all

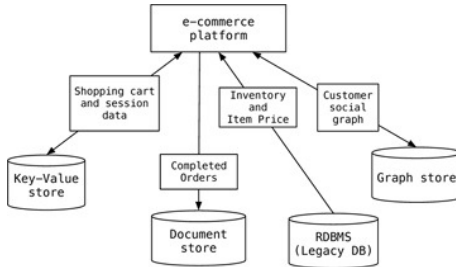


Figure: Persistence dite polyglotte (extrait de NoSQL distilled)

Principe CAP

Constat de Brewer (Towards robust distributed systems, ACM 200) : aucun systèmes distribué n'est à même de satisfaire en même temps les principes C, A et P (au mieux 2 sur les 3)

- 1 **Consistency (cohérence)** : toute modification de donnée est suivie d'effet pour tous les noeuds du système
- 2 **Availability (disponibilité)** : toute requête émise et traitée par un noeud du système, reçoit une réponse (même en situation d'échec à produire une réponse)
- 3 **Partition tolerance (recouvrement des nœuds)** : assurer une continuité du fonctionnement en cas d'ajout/suppression de noeud du système

Principe CAP

Considérations SGBDR / Systèmes NoSQL

- 1 **SGBDR** : Cohérence et haute disponibilité (pas ou peu de P, cad de différents noeuds système)
- 2 **Systèmes NoSQL** : Choix du P (système naturellement distribué) et sélection soit du C, soit du A
 - 1 abandon du A \Leftarrow Accepte d'attendre que les données soient cohérentes
 - 2 abandon du C \Leftarrow Accepte de recevoir des données parfois incohérentes

Positionnement des systèmes / CAP

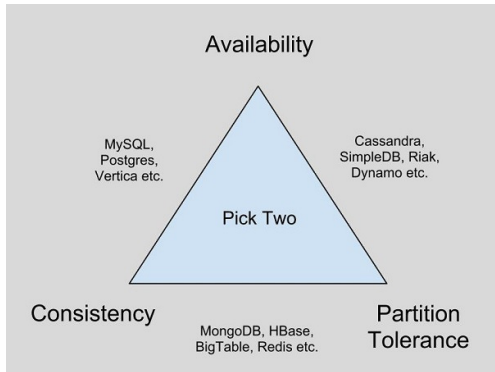


Figure: Synthèse CAP

CAP et architecture distribuée

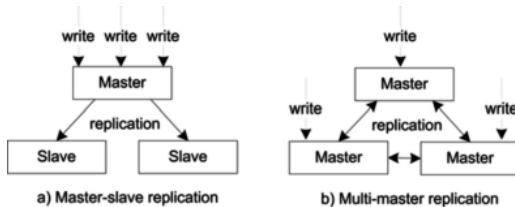


Figure: schémas de distribution

Typologie des systèmes NoSQL

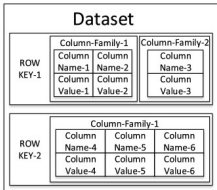
Au regard du mode de représentation choisi

- principe de base : clé/valeur
 - **Systèmes clé/valeur distribués**
 - **Systèmes orientés colonne**
 - **Systèmes orientés document**
- **Systèmes orientés graphe**
- dans un certaine mesure les triples stores et les SGBDOO

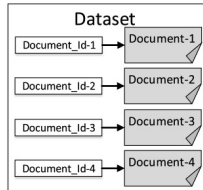
Agrégats clé/valeur : unités naturelles pour le distribué

Key_1	Value_1
Key_2	Value_2
Key_3	Value_1
Key_4	Value_3
Key_5	Value_2
Key_6	Value_1
Key_7	Value_4
Key_8	Value_3

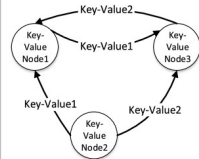
Key-Value Store



Column-Family Store



Document Store



Graph Databases

Figure: Extrait de K. Grolinger et al, 2013

Difficulté : absence de standards

Au regard du mode de représentation comme du système choisis

- 1 APIs spécifiques
 - 2 Terminologies propriétaires
 - 3 Mécanismes de requêtage à géométrie variable
-
- 1 Systèmes ayant fait école ("proofs of concept")
 - 1 BigTable
 - 2 Memcached
 - 3 Amazon's Dynamo

Systèmes existants

Table: Quelques systèmes et leurs modes de représentation

Name	Mode représentation	CAP
CouchDB	Document	AP
Neo4j	Graph	CA
Hbase	Column	CP
Riak	Key-Value	CP
Project Voldemort	Key-Value	AP
Cassandra	Column	AP
Hypertable	Column	unknown

Systèmes existants

Table: Applications communautaires sur le Web

Name	Système NoSQL	Mode
Google	BigTable, LevelDB	Column
LinkedIn	Voldemort	Key-Value
Facebook	Cassandra	Column
Twitter	Hadoop/Hbase, Cassandra	Column
Netflix	SimpleDB, Hadoop/HBase, Cassandra	Column
CERN	CouchDB	Document
Amazon	Dynamo	Key-Value

Définition : semi-structuré

The semi-structured data model is designed as an evolution of the relational data model that allows the representation of data with a flexible structure. Some items may have missing attributes, others may have extra attributes, some items may have two or more occurrences of the same attribute. The type of an attribute is also flexible: it may be an atomic value or it may be another record or collection. Moreover, collections may be heterogeneous, i.e., they may contain items with different structures. The semi-structured data model is a self-describing data model, in which the data values and the schema components co-exist (Dan Suci, Encyclopedia of Database Systems, 2009)

JSON vs XML

JSON : les annotations sur les arêtes et les valeurs sur les nœuds

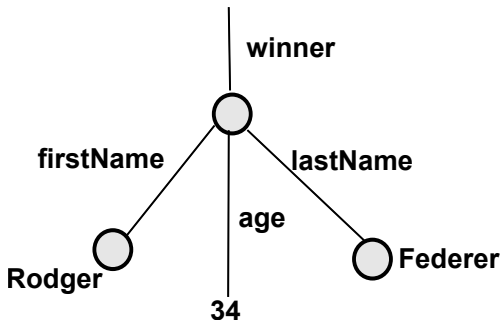


Figure: La structuration JSON

XML vs JSON

XML : les annotations et les valeurs sont sur les nœuds

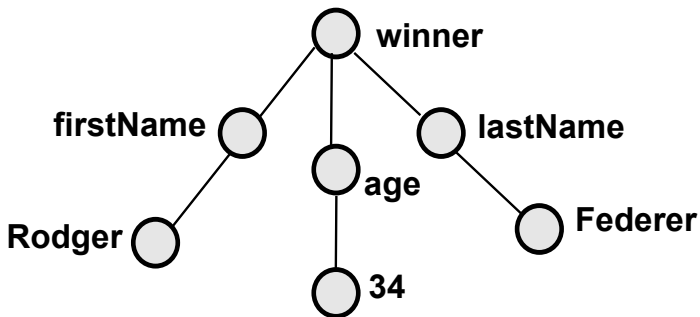


Figure: La structuration XML

JSON, JavaScript Object Notation

JSON (Notation Objet issue de JavaScript) : format d'échange de données notoirement simple

- exploitable par les machines et plutôt lisible par les humains
- sous-ensemble de JavaScript mais indépendant de tout langage
- en train de s'imposer comme le langage du web : format de sérialisation des objets Javascript (mais pas que)

JSON, deux principes structuraux

La paire nom/valeur et le tableau de valeurs

- collection de paires nom/valeur réifiée selon les langages comme un objet, un enregistrement, une structure, un dictionnaire, une table de hachage, une liste typée ou un tableau associatif.
- liste de valeurs ordonnées réifiée selon les langages comme un tableau, un vecteur, une liste ou une suite.

JSON, des exemples d'objets simples à composés

Ensemble de paires nom/valeur

- objet simple : "lastName": "Federer" ou encore "age": 34
- objet composé : ensemble de couples non ordonnés :
{ "lastName": "Federer", "firstName": "Rodger", "age": 34 }
- objet composé : "winner" : { "lastName": "Federer",
"firstName": "Rodger", "age": 34 }

JSON, des exemples d'objets simples à composés

Tableau de valeurs

- valeur composée : un tableau est une collection ordonnée de valeurs qui peuvent ne pas être de même type
- "players" : ["Nadal", "Djokovic", "Murray", "Simon", "Tsonga"]
- "people" : [{"name": "Nadal", "age": 30}, {"name": "Federer", "age": 34, "gender": "male"}, {"name": "Ferrer", "age": 32}]

JSON, des exemples d'objets simples à composés

Tableau de valeurs

```
{  
  "tournament": "The French Open",  
  "year": 2015,  
  "director": "lasName": "Ysern",  
  "firstName": "Gilbert",  
  "players": ["Williams", "Nadal", "Djokovic", "Murray",  
  "Simon", "Tsonga", "Cornet"]  
}
```

GeoJSON : structure JSON pour les entités spatiales (Feature)

```
{  
  "type": "Feature",  
  "geometry": {  
    "type": "Point",  
    "coordinates": [2.25, 48.84]  
  },  
  "properties": {  
    "name": "The French Open", "year": 2015, "  
    "director": {"lastName": "Ysern", "firstName": "Gilbert"},  
    "players": [ "Williams", "Nadal", "Murray", "  
    Simon", "Tsonga", "Cornet" ]  
  }  
}
```

CouchDb : une double orientation

- Système de données orienté documents
 - un document est une unité informationnelle autonome
 - un document est une unité informationnelle composite :
textuel mais aussi image, son, . . .
 - fonctionnalités attendues : gestion de versions, évolution et restructuration, réplication, synchronisation
- Système de données orienté web
 - un document est une ressource web
 - défini au travers d'une URI
 - manipulable au travers d'une architecture REST et du protocole HTTP

Principes CouchDB

Extrait de <http://b3d.bdpedia.fr/bddoc.html>

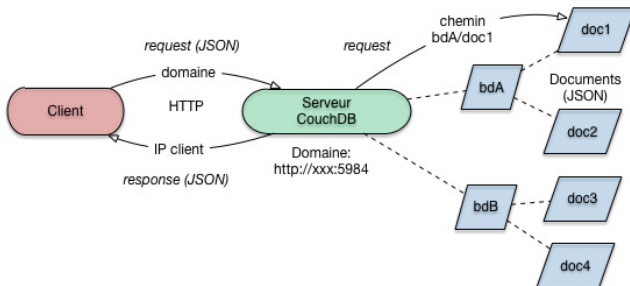


Figure: Principes Généraux CouchDB

La vision document

Modèle de données à base de graphe avec des structures très flexibles

- auto-description des données
- fort pouvoir d'expressivité : le contenu se décrit avec des agrégats de listes, d'ensembles et d'enregistrements
- typage et structure flexibles : données pouvant être typées et/ou structurées (contraintes a priori ou contrôle a posteriori)
- sérialisation : le contenu avec sa structure doit pouvoir être publié sous la forme d'une chaîne de caractères

Efficacité et simplicité des principes REST

- des échanges type web services pour créer, accéder ou gérer des ressources (exploitation des URI à cet effet)
 - GET : retourner une ressource
 - PUT : créer et mettre à jour une ressource
 - POST : faire évoluer une ressource existante par envoi de message (annoter, envoi de données à un processus)
 - DELETE : supprimer la ressource
- Système de données orienté web
 - une interface web est seule nécessaire, disposer de bibliothèques clients est un plus

Exemples d'interaction avec le client HTTP CURL

```
$ curl -X GET localhost:5984  
{ "couchdb": "Welcome", "version": "1.6.0" }
```

- create a db: put a resource

```
$ curl -X PUT localhost:5984/test_db  
{ "ok": true }
```

- call to test_db

```
$ curl -v localhost:5984/test_db
```


Propriétés d'un document : `_id` et `_rev`

```
--create a doc: put a resource in a db
$ curl -X PUT localhost:5984/test_db/o1 -d '{"name": "Nadal"}'
{"ok":true, "id": "o1", "rev": "1-f28a4a5baf607e908cea5863c324d147"}
```

```
--get the document using its URI:
$ curl -X GET localhost:5984/test_db/o1
{"_id": "o1", "_rev": "1-f28a4a5baf607e908cea5863c324d147", "name": "Nadal"}
```

Différentes versions d'un même document

```
--updating a doc : fail
$ curl -X PUT localhost:5984/test_db/o1 -d '{"name": "Nadal", "age": 30}'
{"error": "conflict", "reason": "Document_update_conflict."}
```

--success: updating = adding a new version.
-- multi-version concurrency control protocol which requires the current version number:

```
$ curl -X PUT localhost:5984/test_db/o1 -d '{"_rev": "1-f28a4a5baf607e908cea5863c324d147", "name": "Nadal", "age": 30}'
{"ok": true, "id": "o1", "rev": "2-8e0031775b443f73681b8c2566895f8b"}
```

Illustration Contrôle de concurrence

Extrait de <http://b3d.bdpedia.fr/bddoc.html>

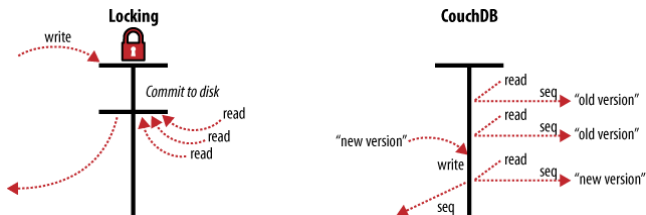


Figure: MVCC et verrouillage pessimiste

Obtenir un UUID, documents de la base, versions d'un document

```
--get an universal identifier
$ curl -X GET http://127.0.0.1:5984/_uuids
{"uuids":["fb506739d9dab3705fe7c58a8c00052c"]}

--display databases
$ curl -X GET localhost:5984/_all_dbs

--display documents' metadata
$ curl -vX GET localhost:5984/test_db/_all_docs

--display information about revision
$ curl -X GET localhost:5984/test_db/ol?revs=true
```

Fichier JSON sans identifiant

```
{  
  "lastname" : "Tsonga",  
  "firstname" : "Jo",  
  "type" : "player",  
  "age" : 31,  
  "tournaments": [ {  
    "city": "Marseille",  
    "year": 2017}],  
  "nationality" : "France"  
}
```

Listing 2: fichier tsonga.json

Fichier JSON avec identifiant

```
{  
  "_id": "laMonf",  
  "lastname" : "Monfils",  
  "firstname" : "Gael",  
  "type" : "player",  
  "age" : 30,  
  "ranking" : 7,  
  "nationality" : "France"  
}
```

Listing 3: fichier monfils.json

Exploiter les fichiers

```
--explicit id : PUT
$ curl -X PUT localhost:5984/test_db/o5 -d @tsonga.
  json -H "Content-Type: application/json"
{"ok":true,"id":"o5","rev":"1-87
  f50d9c01b37db251d7e12a8ad0ce69"}
```

```
-- id within the document : POST
$ curl -X POST localhost:5984/test_db -d @monfils.
  json -H "Content-Type: application/json"
{"ok":true,"id":"laMonf","rev":"1-341422
  aac64aa47ff3a933bb9d62c5b1"}
```

Fichier JSON intégrant plusieurs documents

```
{
  "docs": [
    {
      "_id": "murray",
      "lastname" : "Murray",
      "type" : "player",
      "nationality" : "Great Britain"
    },
    {
      "_id": "djoko",
      "lastname" : "Djokovic",
      "type" : "player",
      "ranking" : 2,
    }
  ]
}
```


Documents de la base, ajout par lots

```
--adding documents : fail = content type is
  required
$ curl -d @murrayDjoko.json -X POST localhost:5984/
  test_db/_bulk_docs

--correct
$ curl -X POST localhost:5984/test_db/_bulk_docs -d
  @murrayDjoko.json -H "Content-Type:_
  application/json"
[{"ok":true, "id": "murray", "rev": "1-7
  d4b633c14d3b66fd2c333947627f7ef"}, {"ok":true, "
  id": "djoko", "rev": "1-0620108
  d9d04bbaa9b55c826664a244d"}]
```

Suppression d'un document, suppression d'une base de données

```
-- document deleting (last version) A new version  
has been created = logical deletion.  
$ curl -X DELETE localhost:5984/test_db/o1?rev=2-8  
e0031775b443f73681b8c2566895f8b  
{"ok":true,"id":"o1","rev":"3-  
c3e4361ab165df874c0ee1d92966d8de"}
```



```
-- drop database  
$ curl -X DELETE localhost:5984/other_db  
{"ok":true}
```

CouchDb : La gestion de documents

- document = objet JSON de taille arbitraire
- Chaque document possède un identifiant (`_id`) et un numéro de version (revision number : `_rev`) amené à changer lors d'une modification
- Des fonctions de validation peuvent venir valider l'insertion ou la modification de documents (type-checking).
- Une vue est un nouvelle collection de paires clé-document (spécification Map/Reduce)
- Un document peut être répliqué sur d'autres instances CouchDb
 - un document est une ressource web défini au travers d'une URI
 - manipulable au travers d'une architecture REST et du protocole HTTP

Le requêtage et la notion de vue

Une vue est le résultat (document JSON) d'une tâche Map/Reduce définie en Javascript

Deux types de vues

- vue permanente (nommée design document) : matérialisée et indexée sur la clé à l'aide d'une structure B+Tree
- vue temporaire : calculée à la volée (possiblement inefficace)

Organisation Générale CouchDB

Extrait de <http://webdam.inria.fr/Jorge/html/wdmch21.html>

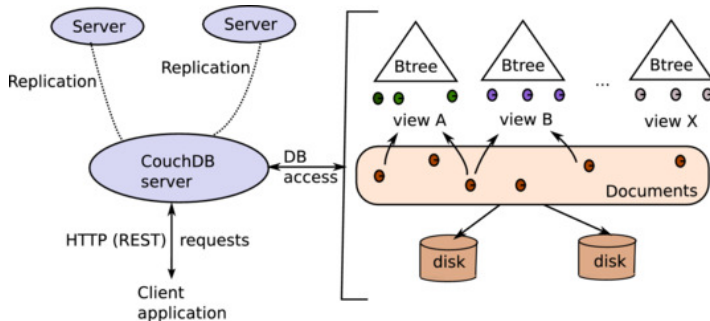


Figure: Vue générale d'un serveur CouchDB

Applicatif Futon (javascript)

Extrait de <http://guide.couchdb.org/editions/1/fr/tour.html>

The screenshot shows the Apache CouchDB Futon web interface. At the top, the browser address bar shows the URL `http://127.0.0.1:5984/_utils/database.html?hello-world/_temp_view`. The interface includes a navigation bar with 'Overview' and 'hello-world'. Below this, there are buttons for 'Create Document...', 'Compact Database', and 'Delete Database'. A 'Select view:' dropdown is set to 'Temporary view...'. The 'View Code' section shows a JavaScript map function:

```
function(doc) {  
  emit(null, doc);  
}
```

 Below the code is a 'Run' button and a language selector set to 'javascript'. A warning message states: 'Warning: Please note that temporary views are not suitable for use in production, as they are really slow for any database with more than a few dozen documents. You can use a temporary view to experiment with view functions, but switch to a permanent view before using them in an application.' The main area displays a table with 3 rows of JSON documents. The table has columns for 'Key' and 'Value'. The first row has a null key and a value object with 'id', 'rev', 'item', and 'prices' fields. The second row has a null key and a value object with 'id', 'rev', 'prices', and 'item' fields. The third row has a null key and a value object with 'id', 'rev', 'item', and 'prices' fields. At the bottom of the table, it says 'Showing 1-3 of 3 rows'. On the right side, there is a sidebar with the CouchDB logo and a 'Tools' menu containing 'Overview', 'Configuration', 'Replicator', 'Status', and 'Test Suite'. Below the tools is a 'Recent Databases' list with entries like 'asdsd', 'couchio_production', 'couchio_test', 'hello-world', and 'pkolight'. At the very bottom right, it says 'Futon on Apache CouchDB 0.9.0dev/13029'.

Figure: Définir une vue avec Futon

Rappel Principes Map-Reduce

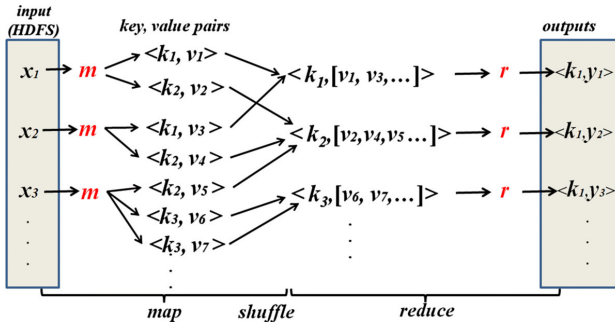


Figure: Map-Reduce : modèle programmation distribuée

Exemple 1 (juste MAP)

Tous les documents de la base

```
{
  "views": {
    "all": {
      "map": "function(doc) { emit(null, doc.name)
            }",
    }
  }
}
```

Listing 5: MAP Function

Exemple 2 (juste MAP)

Identique à `_all_docs`

```
{
  "views": {
    "allDocs": {
      "map": "function(doc) { emit(doc._id, {"rev"
        : doc._rev}); }",
    }
  }
}
```

Listing 6: MAP Function

Exemple 3 (juste MAP)

Poser un index sur la clé (firstname) qui pointe sur des tuples valeurs (nationality)

MAP fonction

```
{
  "views": {
    "players": {
      "map": "function(doc) { emit(doc.firstname,
        doc.nationality); }",
    }
  }
}
```

La vue players une fois enregistrée (document design) peut ensuite être rappelée

Avec REDUCE et cURL

Contenu du fichier tennismen.js

```
{ "_id": "_design/allTennismen", "language": "javascript", "views": { "allT": { "map": "function (doc) {\n{ if (doc.type=='player') {emit(doc._id, {\n\"nationality\" : doc.nationality});}\n}\n}", "reduce": "_count" } } }
```

```
curl -X PUT http://localhost:5984/test_db/_design/allTennismen -d tennismen.js
```

Avec cURL : résultats

```
curl -X GET http://localhost:5984/test_db/_design/  
allTennismen/_view/allT?reduce=false  
  
{ "total_rows":6, "offset":0, "rows": [  
  { "id":"caroGa_93", "key":"caroGa_93", "value":{"  
    nationality":"France"}},  
  { "id":"djoko", "key":"djoko", "value":{"nationality":  
    "Serbia"}},  
  { "id":"JoTs", "key":"JoTs", "value":{"nationality": "  
    France"}},  
  ... ] }
```

La vue nommée allT retourne la nationalité des joueurs

Construire une vue avec cURL

```
curl -X POST -d '{
  "map": "function(doc) {[_emit(doc.nationality, _1)
    ]}",
  "reduce": "function(keys, _values) {_return_sum(
    values)_" }' -H 'Content-Type: application/
  json' 'http://localhost:5984/test_db/
  _temp_view?group=true'
```

retourne :

```
{ "rows": [
  { "key": "France", "value": 4 },
  { "key": "Great_Britain", "value": 1 },
  { "key": "Serbia", "value": 1 }
]}
```

Les joueurs classés au delà de la 6ème place mondiale

Consultation de la vue presqueLesMeilleurs

```
if (doc.nationality && doc.ranking >= 7) {  
    emit(doc.firstname, {"nom": doc.lastname, "age"  
        ": doc.age});  
}  
}
```

Un exemple Map/Reduce Complet

Vue by_country : renvoyer le nombre de joueurs par pays

```
{
  "_id": "_design/tennismen",
  "language": "javascript",
  "views": {
    "by_country": {
      "map": "function(doc) { \n if (doc.
        nationality) { emit(doc.nationality
          , 1);}\n}",
      "reduce": "function(keys, values) {\n
        return sum(values);\n}\n"
    }
  }
}
```

Un exemple Map/Reduce Complet

Vue by_country : exploiter la clé

```
$ curl 'http://.../_design/tennismen/_view/  
by_country?group_level=1'  
{"rows": [  
{"key": "France", "value": 3},  
{"key": "Great_Britain", "value": 1},  
{"key": "Serbia", "value": 1}  
]}
```

```
$ curl 'http://.../test_db/_design/tennismen/_view/  
by_country'  
{"rows": [  
{"key": null, "value": 5}  
]}
```


cURL et sans le reduce

```
curl 'http://127.0.0.1:5984/test_db/_design/  
    tennismen/_view/by_country?key="France"&reduce=  
    false'  
{ "total_rows":5, "offset":0, "rows": [  
  { "id": "laMonf", "key": "France", "value": 1 },  
  { "id": "o5", "key": "France", "value": 1 },  
  { "id": "o6", "key": "France", "value": 1 }  
] }
```

Différents niveaux d'agrégats (1)

```
{
  "_id": "_design/tennis",
  "_rev": "2-4c82905cc71abedd3671d8de02c26ed8",
  "language": "javascript",
  "views": {
    "TestReduce": {
      "map": "function (doc) { if (doc.type=='
        player')\n{ emit([doc.gender, doc.
          nationality, doc._id], 1);}}\n",
      "reduce": "_count"
    }
  }
}
```

Différents niveaux d'agrégats (2)

```
-- all keys
curl 'http://127.0.0.1:5984/urbain/_design/tennis/_view/TestReduce'
-- group on first attribute of the key
curl 'http://127.0.0.1:5984/urbain/_design/tennis/_view/TestReduce?group_level=1'
-- group on first and second attributes of the key
curl 'http://127.0.0.1:5984/urbain/_design/tennis/_view/TestReduce?group_level=2'
...
```

Différents niveaux d'agrégats (3)

```
$ curl 'http://.../_view/TestReduce'  
{ "rows": [  
  { "key": null, "value": 6 }  
]}  
$ curl 'http://.../_view/TestReduce?group_level=1'  
{ "rows": [  
  { "key": ["F"], "value": 2 },  
  { "key": ["M"], "value": 4 }  
]}  
$ curl 'http://.../_view/TestReduce?group_level=2'  
{ "rows": [  
  { "key": ["F", "France"], "value": 2 },  
  ... ] }
```

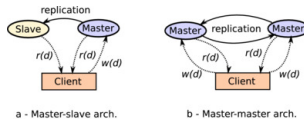
"Collation view" pour contourner l'absence de jointure

```
function(doc) {  
  if (doc.type == "country") {  
    emit([doc._id, 0], [{"pop":doc.population}, {"  
      players":doc.players}]);  
  } else if (doc.type == "player") {  
    emit([doc.nationality, 1], [{"name":doc.  
      lastname}, {"rank":doc.ranking}]);  
  }  
}
```

Rapprocher informations nation et joueurs

Serveurs distribués CouchDB

<http://webdam.inria.fr/Jorge/html/wdmch21.html>



c - Shared-nothing (Consistent hashing)

Figure: schémas de distribution

Les primitives associées à la réplication (unidirectionnelle)

```
curl -X POST http://localhost:5984/_replicate -d '{  
  "source":"test_db", "target":"replika", "  
  continuous":true}' -H 'Content-Type:  
  application/json'
```

Répliquer la BD dans son ensemble, créer au préalable replika, "continuous" pour synchroniser les mises à jour dans la foulée

Conflits liés à la réplication

<http://webdam.inria.fr/Jorge/html/wdmch21.html>

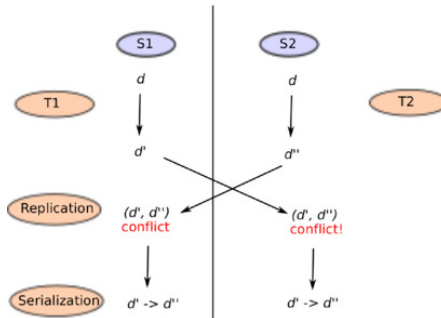


Figure: Mises à jour conflictuelles