CouchDB

An introduction to CouchDB, a “NoSQL” document database

Serge Abiteboul Ioana Manolescu Philippe Rigaux
Marie-Christine Rousset Pierre Senellart

Web Data Management and Distribution
http://webdam.inria.fr/textbook

September 23, 2011

What is CouchDB?

A system representative of the “NoSQL” trend.

1. a semi-structured data model, based on JSON;
2. no schema;
3. *structured materialized views* produced from document collections;
4. views defined with the MAPREDUCE paradigm, allowing both a parallel computation and incremental maintenance of their content;
5. distributed data management techniques: consistent hashing, support for data replication and reconciliation, horizontal scalability, parallel computing, etc.

This presentation

Practice-oriented: the goal is to let you play with CouchDB and discover its features.

The bluk of the course = a general presentation of the main features of CouchDB, with focus on the data model and Map/Reduce programming.

Get the datasets from the book web site, and play with the system on-line.

Data model

JSON: a text format initially designed for serializing Javascript objects $\Rightarrow$ now

Primary use: data exchange in a Web environment (typ. AJAX applications) – Extended use: data serialization and storage.

A lightweight XML – prety easy to integrate to any programming language, with minimal parsing effort.

Downside: no schema (at the moment) – no query language.

At the core: key-value construct

Basic example:

```
"title": "The Social network"
```

Atomic data types: character strings, integers, floating-point number and Booleans (`true` or `false`). Non-string values need not be surrounded by "".

```
"year": 2010
```

Complex values: objects

An *object* is an unordered set of name/value pairs.

The types can be distinct, and a key can only appear once.

```
{"last_name": "Fincher", "first_name": "David"}
```

A object can be used as the (complex) value component of a key-value construct:

```
"director": {
      "last_name": "Fincher",
      "first_name": "David",
      "birth_date": 1962
  }
```

Complex values: arrays

An array is an ordered collection of values that need not be of the same type.

```
 "actors": ["Eisenberg", "Mara", "Garfield", "Timberlake"]
```

A *document* is an object. It can be represented with an unbounded nesting of array and object constructs

```
{
  "title": "The Social network",
  "year": "2010",
  "director": {"last_name": "Fincher",
               "first_name": "David"},
  "actors": [
       {"first_name": "Jesse", "last_name": "Eisenberg"},
       {"first_name": "Rooney", "last_name": "Mara"}
   ]
}
```

CouchDB in a nutshell

A document, web-oriented data system.

Document oriented. Document are complex and autonomous pieces of information. Can store files, functions, any type of media. But no references.

Typical functionalities of document application: versioning, replication, synchronization, restructuring.

Web-oriented. A document is a resource in the Web sense – it has a URI, and can be manipulated via HTTP (REST architecture).

ApartÃ©: REST principles

A Web-service dialect that enables exchanges of HTTP messages to access, create, and manage resources.
protocol as follows:

`GET`	retrieves the resource referenced by the URI.
`PUT`	creates the resource at the given URI.
`POST`	sends a message (along with some data) to an existing resource.
`DELETE`	deletes the resource.

Very convenient in a Web environment: no need to use a client library – Documents can easily be incorporated in a Web interface.

A short interactive session

Talk to the server: send an HTTP request, get a response.

```
$ curl -X GET http://mycouch.org
{"couchdb":"Welcome","version":"1.0.1"}
```

Create a db = put a resource (the name suffices).

```
$ curl -X PUT http://mycouch.org/myDB
{"ok":true}
```

Create a document = put a resource in a db (give the JSON document in the HTTP request).

```
$ curl -X PUT http://mycouch.org/myDB/myDoc \
           -d '{"key": "value"}'
{"ok":true,"id":"myDoc","rev":"1-25eca"}
```

Get the document after its URI:

```
$ curl -X GET http://mycouch.org/myDB/myDoc
{"_id":"myDoc","_rev":"1-25eca","key":"value"}
```

Document management in CouchDB

Each document has an id and a revision number.

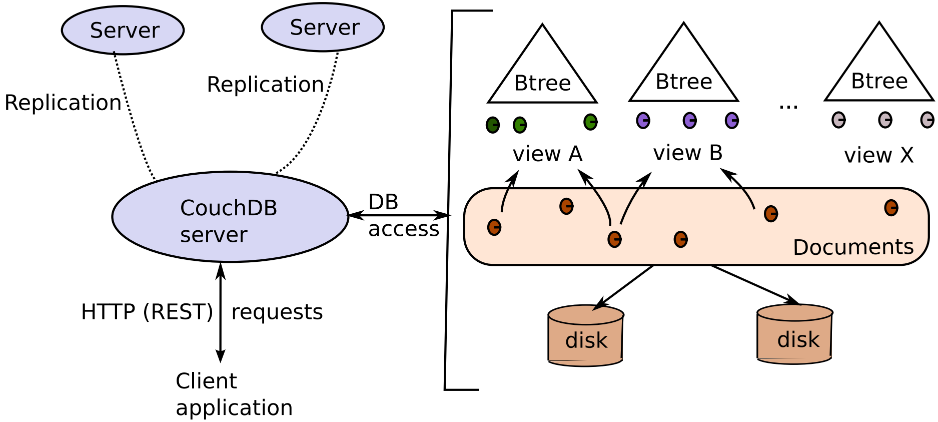
Each update to a document creates an new version, with the same `_id` but a new revision number.

Validation functions can be assigned to a collection: any document inserted or updated must be validated by these functions (ad-hoc type-checking).

A view is a new key-document collection, specified via MAPREDUCE.

Documents can be replicated in other CouchDB instances.

Architecture of CouchDB



Adding data

Get the JSON examples from our site. Documents can always be inserted via the `curl` interface.

If you specify the id in the URI: use PUT.

```
$ curl -X PUT $IP/movies -d @The_Social_Network.json
```

If the id is part of the document use POST.

```
$ curl -X POST $IP/movies -d @The_Social_Network.json\
      -H "Content-Type: application/json"
```

Remember: PUT creates a resource; POST sends a message to an existing resource.

You can also import files from the CouchDB admin web site.

Updating data

Updating in COUCHDB = adding a new version.

COUCHDB applies a Multi-version concurrency control protocol which requires that you send the version that must be updated:

```
$ curl -X PUT $IP/movies/tsn?rev=1-db1261 -d @newDoc.json \
  -H "Content-Type: image/jpg"
{"ok":true,"id":"tsn","rev":"2-26863"}
```

Deletion is obtained with DELETE.

```
$ curl -X DELETE $COUCHADDRESS/movies/tsn?rev=2-26863
{"ok":true,"id":"tsn","rev":"3-48e92b"}
```

A new version has been created! (logical deletion).

Views in CouchDB

A view is the result of a MapReduce job = a list of $(key, value)$ pairs.

Views are materialized and indexed on the key by a B+tree.

A Map function

```
function(doc)
{
   emit(doc.title, doc.director)
}
```

A Reduce function

```
function (key, values) {
    return values.length;
}
```

Accessing views

Here is a view (without reduce function).

```
function(doc)
{
   for (i in doc.actors) {
      actor = doc.actors[i];
      emit({"fn": actor.first_name, "ln": actor.last_name},
   }
}
```

Save it in the design document named `examples`, and name the view `actors`. The view can be queried with:

```
$ curl IP/movies/_design/examples/_view/actors
{"total_rows":16,"offset":0,
"rows":[
 {"id":"bed7",
   "key":{"fn":"Andrew","ln":"Garfield"},"value":"The Socia
 {"id":"91631b",
   "key":{"fn":"Clint","ln":"Eastwood"},"value":"Unforgiven
```

Querying views

A view is a B+tree index. So:

```
function(doc)
{
  emit(doc.genre, doc.title) ;
}
```

is equivalent to

```
create index on movies (genre);
```

Recall the B+trees support key and range queries:

```
$ curl $IP/movies/_design/examples/_view/genre?key=\"Drama\
{"total_rows":5,"offset":2,"rows":[
{"id":"9163", "key":"Drama","value":"Marie Antoinette"},
{"id":"bed7", "key":"Drama","value":"The Social network"}
]}
```

For range queries, send the two parameters `startkey` and `endkey`.

The replication primitive

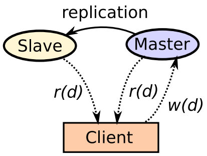
CouchDB supports natively one-way replication from one instance to another.

```
curl -X POST $COUCHADDRESS/_replicate \
   -d '{"source": "movies",  "target": "backup",
        "continuous": true}' \
   -H "Content-Type: application/json"
```

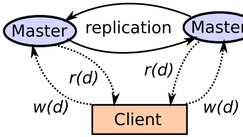
That's all: any change in `movies` is automatically reported in `backup`.

Distribution strategies

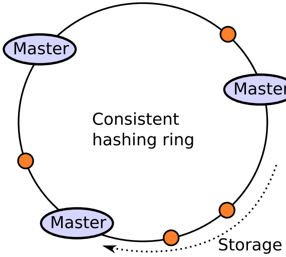
Combine a proxy that distributes requests, with the replication feature of CouchDB.



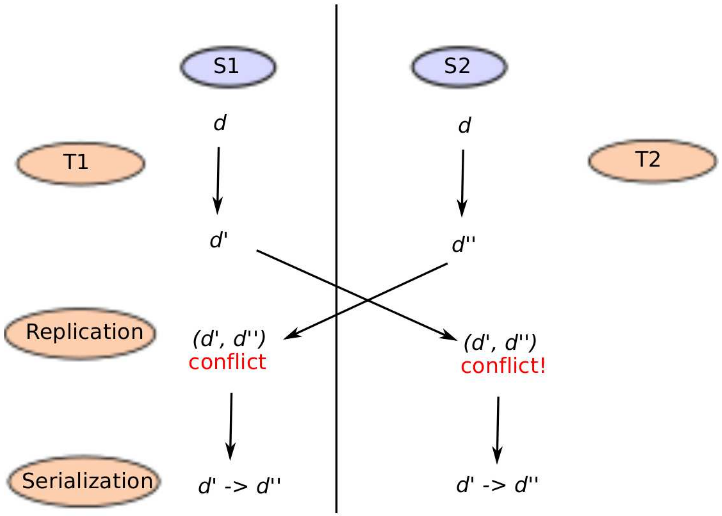
a - Master-slave arch.



b - Master-master arch.



How does MVCC works with replication?



Set up your environment

- Create an account using the admin. interface.
- Create at least one database, and set this db the default one.
- Load the `zipmovies.zip` and `zipartists.zip` files in your database, and look at the imported documents.
- Create the views as suggested in the book Chapter.