Université libre de Bruxelles

Advanced Databases Project

Developing of a social network using CouchDB

Aldar Saranov, Najim Essakali

[Aldar.Saranov@ulb.ac.be](mailto:Aldar.Saranov@ulb.ac.be)

Najim.Essakali@ulb.ac.be

INFO-H-415 Advanced Databases (M-INFOS/F277)

Esteban Zimányi

December 2016

# Project introduction 1/20

We decided to develop a simple social network for user interacting via Internet connection. The social networks became one of the most trending areas in World Wide Web since Web 2.0 rise in 2005 [1]. Developing of a social network definitely requires establishing of a database capable of storing enormous amount of data (posts, messages, media etc.) and operatively respond to user requests.

In order to remain competitive over the social network market a social network is forced to have high technical characteristics and withstand benchmarking with the others. We can determine following general technical demand for a modern social network:

1. Fast response time.
2. Security of personal data.
3. Persistence and consistency of data.
4. Alterability (being able to easily refine functionality and/or type of content).

# CouchDB 4 /20

CouchDB is a document-oriented schemaless opensource database developed by Apache Software Foundation. CouchDB is a part of NoSQL conception databases and key-value databases in particular.

The term schemaless means that the database does not oblige the documents/objects to correspond to a certain logical form (besides the document formatting) while ensuring only a minor set of constraints (e.g. unique id). Most of the constraints should be implemented at application level what provides a high model flexibility to the developer.

## Fast response time

To ensure low response time CouchDB uses B+ trees as data structure what assures that all essential operations like accessing/adding/deleting will be handled in logarithmic time. While accessing nodes with height less than 10, CouchDB can still index millions of elements.

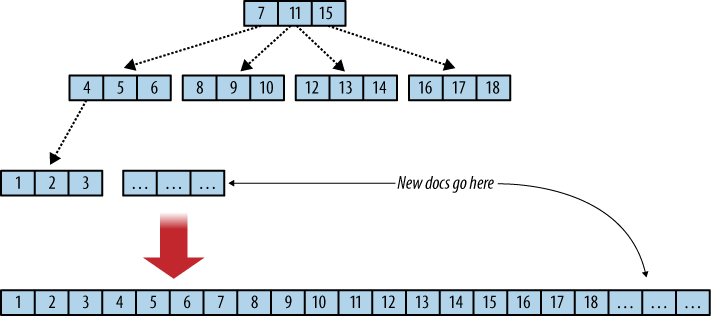


Figure X. B-tree and append only.

## Security of personal data

CouchDB has a quite simple security policy. There is a set of users (Admins) which are authorized to do any modifications to the database. This is called an “Admin party” mechanism. By default CouchDB server is only listening to the requests coming from 127.0.0.1 (localhost). List of operations which an admin can carry out:

1. Creating a database (PUT /database)
2. Deleting a database (DELETE /database)
3. Creating a design document (PUT /database/\_design/app)
4. Updating a design document (PUT /database/\_design/app?rev=1-4E2)
5. Deleting a design document (DELETE /database/\_design/app?rev=1-6A7)
6. Triggering compaction (POST /\_compact)
7. Reading the task status list (GET /\_active\_tasks)
8. Restarting the server (POST /\_restart)
9. Reading the active configuration (GET /\_config)
10. Updating the active configuration (PUT /\_config)

CouchDB also provides built-in tools for password hashing what is especially useful in our case.

## Persistence and consistency of data

CouchDB is fault-tolerant since there are many mechanisms which are designed to ensure the persistence of the stored data.

### Operation consistency

However CouchDB B-tree implementation is slightly different from the canonic one. While it maintains all of the important properties, it adds Multi-Version Concurrency Control (MVCC) and an append-only design. B-trees are used to store the main database file as well as view indexes. One database is one B-tree, and one view index is one B-tree.

MVCC is designed to conduct concurrent read and write operations without using the locking of the system. Writes are serialized, allowing only one write operation at any point in time for any single database of the server. Write operations do not block read operations, thus there can be any number of read operations at any time. Each read operation represents a consistent view of the database. How this is accomplished is at the core of CouchDB model of storage.

The short answer is that because CouchDB uses append-only files, the B-tree root node must be rewritten every time the file is updated. However, old portions of the file will never change, so every old B-tree root, should you happen to have a pointer to it, will also point to a consistent snapshot of the database.

In a B-tree, data is being stored only in leaf nodes. CouchDB appends data only to the database file that keeps the B-tree on disk and grows only at the end. Both adding and deleting the documents are being recorded at the end of the file. The consequence is a robust database file. Computers fail for plenty of reasons, such as power loss or failing hardware. Since CouchDB does not overwrite any existing data, it cannot corrupt anything that has been written and committed to disk already.

If during in this process the power is cut-off occurs and CouchDB is being restarted later - the database file is in a consistent state and does not need a checkup. CouchDB starts to read the database file backwards. When it passes a footer pair, it is checking its state: if the first 2 kb are corrupted (figured out using checksum), CouchDB replaces it with the second footer and all is well. If the second footer is corrupt, CouchDB copies the first 2 kb over and all is well again. Only when both footers are flushed to disk successfully CouchDB will confirm that a write operation was successful. Data is never lost, and disk data is never corrupted.

### Replication

CouchDB has a very powerful replication system. Replication synchronizes two copies of the same database, hence allowing users to have lower latency access to data. These databases can be situated on the same server or on two different server - CouchDB does not make any distinction. If changes take place in one copy of the database, replication will send these changes to the other copy.

Replication is a one-off operation: you send an HTTP request to CouchDB that includes a source and a target database, and CouchDB will send the changes from the source to the target. Granted, calling something world-class and then only needing one sentence to explain it does seem odd. Replication request is called by making a POST request:

Listing X. Replication post request.

|  |
| --- |
| POST /\_replicate HTTP/1.1  {"source":"database","target":"http://example.org/database"} -H "Content-Type: application/json" |

When an admin ask CouchDB to replicate one database to another, it will go and compare the two databases to find out which documents on the source differ from the target and then submit a batch of the changed documents to the target until all changes are transferred. Changes include new documents, changed documents, and deleted documents. Documents that already exist on the target in the same revision are not transferred; only newer revisions are.

Such simple replication system can be useful for creating back-ups or snapshots for ensuring data persistence in our social network.

## Alterability

Since CouchDB is a schema-less database it allows to conduct almost unlimited changing of the entity-relationship model. If the social network developers that it is desired to implement new features for it then it can be done painlessly for the already accumulated date. It is allowed to add any new attributes to the new documents. The only thing developers should care about is to implement correct data processing at application-level.

## General description

## Flexibility of decomposition

### Http protocol

### Json documents

### Map/Reduce

### Futon

# Social network developing 15/20

## Database structure 3/15

### Model

### Common documents

### View documents

## Curl 2/15

Listing X. Curl script snippet.

|  |
| --- |
| curl -X DELETE http://127.0.0.1:5984/social  curl -X PUT http://127.0.0.1:5984/social  curl -X PUT http://localhost:5984/social/\_design/find -d @find.txt  curl -X PUT http://localhost:5984/social/\_design/statistics -d @statistics.txt  curl -X PUT http://127.0.0.1:5984/social/cff457c34484830b569a999a27014134 -d @user\_1.txt  curl -X PUT http://127.0.0.1:5984/social/cff457c34484830b569a999a27016314 -d @user\_2.txt |

## Server description 10/15

### Python DB access interface

### Backend

# References

1. <https://en.wikipedia.org/wiki/Web_2.0>
2. http://guide.couchdb.org/