CouchDB

7 Databases in 7 Weeks

Highlights

- schema-free, JSON-document, distributed DB
- REST access
- Written in Erlang (highly fault tolerant by design)
- Released 2005

So it’s basically MongoDB?

- MongoDB: if you need dynamic queries. If you prefer to define indexes, not M/R. If you need good performance on a big DB. If your data changes too much.

- CouchDB: For accumulating, occasionally changing data on which pre-defined queries are to be run. Place where versioning is important.

Demo!

Integration

- Disappointing Java Integration
  - Expected Spring-data, but there’s only a 2-year-dormant community extension :-(
  - 2007-era dormant “couchdb4j”
  - 2010-era dormant “opencredo-couchdb”
  - Alpha github project “couch4j”
- Found http://www.lightcouch.org/
  - Recent support
  - Thin wrapper around REST API
- Demo

Querying

- Anything more complicated than simple GETs is done via “views”
- Temporary views - inefficient, used for development
- Design Docs - views, saved in to the CouchDB just like other docs. Optimized.

Mappers

- Primary mechanism to view/query DB
- Ad hoc over REST

  -or-

- Compiled/saved with DB
  - Re-run when the underlying document(s) change

Reducers

- Recursive
- Persisted output
- Re-reduced when underlying docs change

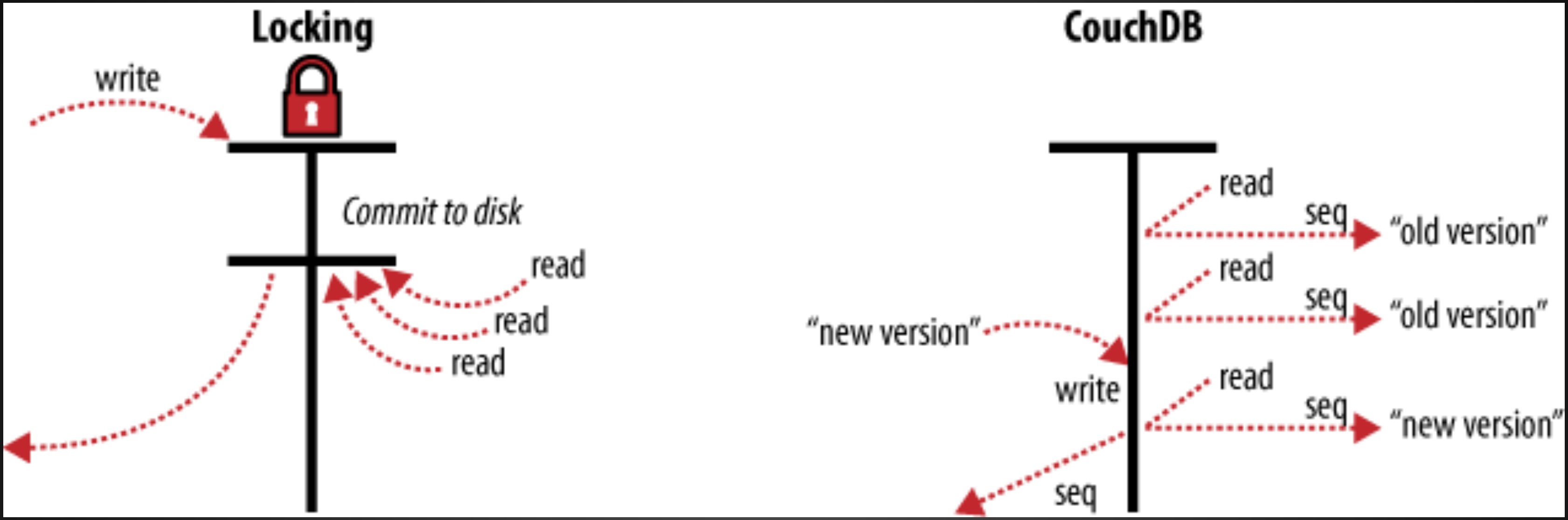
Data Storage

- CouchDB uses a B-Tree storage engine for everything
- Binary “.couch” files on disk
- Documents are versioned, not locked

Revisions

- Documents have revisions
- Pattern: “Nth revision” - “SHA hash”
- Also used as HTTP response E-Tags
- This MVCC system lets Couch stay stateless and REST-y
- Older revisions are not guaranteed to stay

No Locks



- Reads are always serviceable (gets last SNAPSHOT)
- First-write-wins, subsequent losers are told:
  {"error":"conflict","reason":"Document update conflict."}

Attachments

- Like email “docs”, you can add attachments
- demo

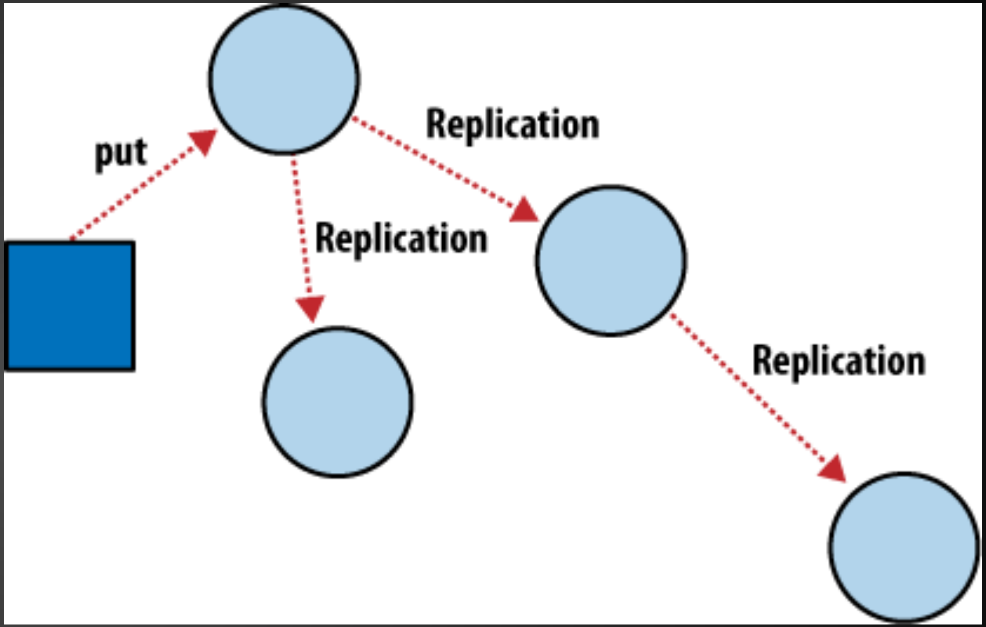
Validation

- CouchDB can do in-database data validation
- Javascript validators can be plugged in

Replication

Distributed Consistency

Eventual consistency achieved by incremental document replication



(aka doc changes periodically copied between servers)

Replication Demo

- Futon has a built-in “Replicator” page
- Just a facade on top of web service endpoint

e.g.

```
curl -X POST http://127.0.0.1:5984/_replicate \
     -d '{"source":"music","target":"music-replica"}' \
     -H "Content-Type: application/json"
```

- local/local, local/remote, remote/local, remote/remote

How do replication conflicts work?

- Couch DB comes with automatic conflict detection <u>and resolution</u> (!)

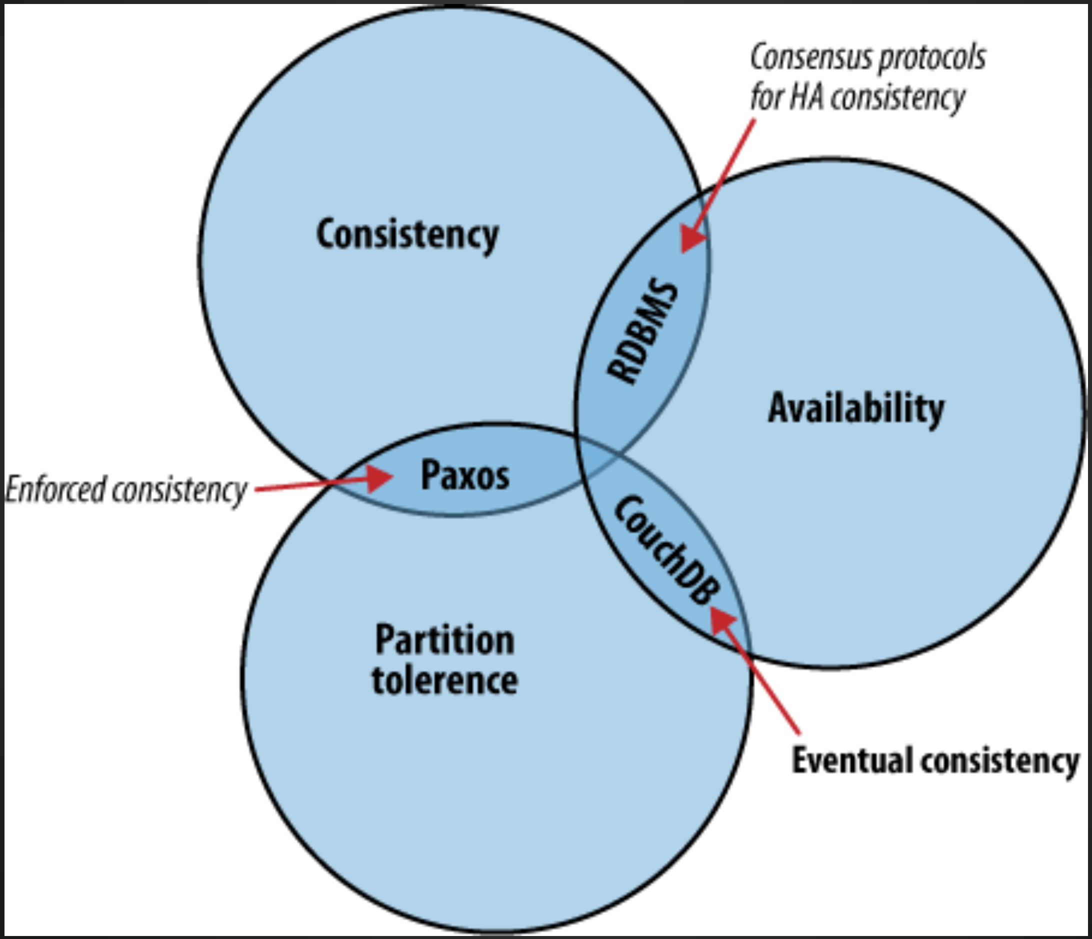
- If a replication conflict is detected, all nodes resolve a single “winner” the same way

- The loser version(s) is not discarded, it is saved as a previous version

THAT’s the solution?

- Philosophy: let the application figure it out
- E.g. EverNote
- E.g. iTunes library
- E.g. Ticketmaster

Eventually Consistent



. Replication

- CouchDB replication is everywhere-to-everywhere
- No sharding
- So really just used to increase r/w throughput

Changes API

Allows clients to watch DB for changes and get updated instantly:

- polling
- long-polling (aka “Pulling a Spradlin”)
  node src/main/js/watch_changes_longpolling.js music
- continuous

Other interesting errata

- Query server (JS and optionally Erlang)
- OS daemon watching
- httpd_global_handlers & couch-as-proxy
- built-in reduce functions: *supah fast*
- CouchApps (http://docs.couchdb.org/en/latest/couchapp/ddocs.html#list-functions)
- Externals API (delegate to procs for doc handling w/ JSON over stdio)
- /db/_local/ (non-replicating)