latex input: mmd-article-header Title: NoSQL Notes Author: Ethan C. Petuchowski Base Header Level: 1 latex mode: memoir Keywords: SQL, NoSQL, Databases CSS: http://fletcherpenney.net/css/document.css xhtml header: copyright: 2014 Ethan C. Petuchowski latex input: mmd-natbib-plain latex input: mmd-article-begin-doc latex footer: mmd-memoir-footer

CouchDB

8/7/15, 1/15/16

- Docs
- **This database is surprisingly interesting**
- Book

From the Docs and the Book

1. Seems a lot like **MongoDB**
2. NoSQL database with no schema
3. Data is stored as JSON "documents", whose structure is not pre-defined
4. Each document has a unique ID field `_id`
5. "Views" are built on-demand for aggregating and reporting on documents
6. Designed to store and report on large amounts of *semi-structured, document oriented* data
7. Greatly simplifies *document oriented applications*, such as **collaborative web applications**
8. **Peer based** -- any number of hosts (servers *or* offline clients) can have independent "replica copies" of the same database, giving applications full database interactivity (CRUD)
    - When back online, or on a schedule, database changes can be replicated bi-directionally, using built-in conflict detection and management
9. It has extensive replication configuration functionality, for creating powerful solutions to many IT problems
10. It was implemented in Erlang, which enhances its reliability and scalability
11. The CouchDB CRUD API is RESTful HTTP (emphasis on _stateless_ness)
12. Documents can have any number of fields and *attachments*
13. Document updates are lockless, optimistic, and all-or-nothing (can't only partially complete)
14. ACID semantics on a document-level
15. "Any number of clients can be reading documents without being locked out or interrupted by concurrent updates, even on the same document."
16. "CouchDB read operations use a Multi-Version Concurrency Control (MVCC) model where each client sees a consistent snapshot of the database from the beginning to the end of the read operation."

17. Documents are indexed in b-trees by `(docID, seqID)`, where the seqID is incremented on updates
18. Commits are append-only, and then there is a compaction process
19. "Views" are defined in "*design documents*". They have a `map` function (in Javascript) that for each document emits zero or more rows to the view table
20. When a view is computed, it is scored, so that when you want to view an updated version, it just updates the previous view
21. You can write validation code in Javascript to limit what is allowed to be written to the database, and by whom
22. **Eventually consistent** replication model
23. **Built for Offline** -- can replicate to devices (like smartphones) that can go offline and handle data sync for you when the device is back online.
24. Offers a built-in administration interface accessible via web called Futon
25. To update a document with a particular `_id`, you must supply your latest known `_rev` (revision) field for that document. If yours is out of date compared to the one on the server (or you don't send any `_rev`), the update will fail.
26. So `_rev` id's are given on creation and updated of documents
    - Surprisingly, note: "CouchDB does *not* guarantee that older versions are kept around."
    - These are also used as `Etags` for caching (great call)
27. Note: always use `PUT` instead of `POST` so that you specify your `_id` instead of letting Couch assign its own `uuid`, because that's just not a useful value. Consider using `Date.now()` instead.
28. I'm pretty sure Couch closes the TCP connection after every HTTP req/res pair, to reduce state overhead on the server. My intuition is that this would be a bad idea because of the increased overhead of initiating and closing all these connections, but I'll trust the designers to be correct on this one.
29. replication replicates based on a snapshot at the point in time when replication was started
30. Replication modes
    1. Push -- pushes changes from this server to a remote one
    2. Pull -- requests changes from the remote one
    3. Remote -- "useful for management operations" -- no idea what that means

Replication

- Docs
    - Replication-specific sections are
        - All of Chapter 4: Replication
        - Chapter 11: JSON Structure reference
            - 11.11 -- list of active tasks
            - 11.12 -- replication settings
            - 11.13 -- replication status

        - Chapter 3: Configuration
            - 3.7 -- replicator
        - Chapter 10: API reference
            - 10.2.6 -- `/_replicate`
            - 10.3.13 -- `/db/_revs_diff`
    - Relevant change-feed sections are
        - Chapter 10: API reference
            - 10.3.4 -- `/db/_changes`
- Synchronizes two copies of the same database
- Controlled through **documents** in the *_/replicator* database
- Only copies latest revisions of each document
- Involves exactly one source and one destination database
- At least one of those must be local
    - Make both local to e.g. use this as a snapshotting mechanism
- If the dest is remote, then it's "*push*", otw it's "*pull*"
- Triggered by [the the *act* of] **storing** a document in the `/_replicator` database
    - This triggers a *task* whose "*replication status*" can be inspected through the "`/_active_tasks` API"
- Cancelled by
    - deleting the relevant replication document,
    - *or* updating it to set it's `cancel` property to `true`
- It seems like replication amounts to just hitting the `/db/_changes` api for batches
    - if any of the retrieved `_ids` are missing,
    - or the `_revs` don't match,
    - a query is sent for those particular documents

Algorithm

- There are three *databases* involved: a Source, a Target, and a Replicator
    - At the end, the Target's contents will match the contents of Source at the time replication began, wrt the contents of Source
        - I.e. I don't think Target will delete documents that Source has never seen (and hence never deleted), although I'm not sure about this, and luckily it doesn't matter to me right now.
    - These are *databases* in the sense of "my RDBMS has 5 databases"
        - *Not* in the sense of "I have a mysql database installed"
        - What a doozy
- First of all, we must get the *checkpoint*, i.e. the last `seq_id` seen by the target, in terms that the `source/db/_changes` feed understands
- `/db/_revs_diff` is something that you pass an `_id` mapped to a list of `_revs`
    - It returns to you the list of revisions for that document that the target has *not* seen