BUILD A P2P DOCUMENT ORIENTED DATABASE

BARREL

https://barrel-db.org
REQUIREMENTS
- write record
- read
- create
- update
- delete

DB
- more
- more
- more
- more
- more

DOC
- more
- more
- more
- more
- more

View DB
- more
- more
- more
- more
- more

DOC
- more
- more
- more
- more
- more

View DB
- more
- more
- more
- more
- more

NUCC
- more
- more
- more
- more
- more
VISION
AND CONCEPT
DATA IS MOBILE
PEER TO PEER (P2P)

- discover
- replicate
- share
- Local first
- Put/Match the data next to you
- Query Locally
- Replicate a view of the data you need
WHAT IS BARREL
- a document database
- documents are JSON with attachments and links
- changes feed for document and indexes
- replication between any nodes in both ways
- views (~ map)
- HTTP 1.1/2 API

**WHAT IS BARREL**
DATA: not just blobs

Replicated APPs

Couchapps but extended and revisited

REPLICATED APPS
DECONSTRUCT
invalid data

Document revision

Block

btree node

Doc1

Doc2

Doc3

Doc4

Doc5

Doc6

Doc7

Btree Node

Btree Node

version

APPEND ONLY & MVCC
Create a new file to remove the fragmentation

A race between copy and the addition of new data

Require at least twice of the storage
### ID-Index

<table>
<thead>
<tr>
<th>ID</th>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID 1</td>
<td>Metadata 1</td>
</tr>
<tr>
<td>ID 2</td>
<td>Metadata 2</td>
</tr>
<tr>
<td>ID 3</td>
<td>Metadata 3</td>
</tr>
</tbody>
</table>

### Seq-Index

<table>
<thead>
<tr>
<th>Seq</th>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq 1</td>
<td>Metadata 1</td>
</tr>
<tr>
<td>Seq 2</td>
<td>Metadata 2</td>
</tr>
<tr>
<td>Seq 3</td>
<td>Metadata 3</td>
</tr>
</tbody>
</table>

### DB File

- **Doc**
- **Btree Node**
- **Btree Node**

### Indexed Document

**DOCUMENT STORAGE**

*enki*
- 2 indexes (btree): by sequence, by id
- 1 index for local documents without conflict handling
- A revision tree is stored in indexes pointed to the revision offset
- The revision is stored in the file separately

**HOW ARE STORED DOCUMENTS**
- Reverse index (map)
- Index using a function
- Function in javascripts, erlang, ..
- Incremental index
- Retrieves changes (aka view changes)
- View are regrouped by groups (1 db file/group)

**VIEWS**
### Log-Index

<table>
<thead>
<tr>
<th>DOCID</th>
<th>View 1</th>
<th>KEY 1</th>
<th>SEQ 1</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>KEY 2</td>
<td>SEQ 2</td>
<td>DEL</td>
</tr>
<tr>
<td></td>
<td>View 2</td>
<td>KEY 1</td>
<td>SEQ 1</td>
<td>ADD</td>
</tr>
</tbody>
</table>

### Key-Index

<table>
<thead>
<tr>
<th>[KEY 1, DOCID]</th>
<th>[VALUE, DOCREV, SEQ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[KEY 2, DOCID]</td>
<td>[del, DOCREV, SEQ]</td>
</tr>
<tr>
<td>[KEY 3, DOCID 2]</td>
<td>[VALUE, DOCREV, SEQ]</td>
</tr>
</tbody>
</table>

### SEQ-Index

<table>
<thead>
<tr>
<th>[SEQ 1, KEY]</th>
<th>[VALUE, DOCREV, SEQ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SEQ 2, KEY 2]</td>
<td>[del, DOCREV, SEQ]</td>
</tr>
<tr>
<td>[SEQ 3, KEY]</td>
<td>[VALUE, DOCREV, SEQ]</td>
</tr>
</tbody>
</table>

**view**

**VIEW STORAGE**
REVISION TREE

Body of leaf node in winning rev tree must be kept
Body of leaf node in losing rev trees must be kept
BUILT IN ERLANG
Write is slow

Read should not being blocked by writes

No shared memory

No atomic integer trick

Only actors and message passing

Operations on a doc are atomic

CHALLENGES
READ/WRITE OPERATIONS
- LRU to cache blocks
  https://github.com/barrel-db/erlang-lru
- 1 File process, Operations are limited
- DB users are linked to the database process
- Optional Write buffer to reduce the latency
- Optional wal

READ/WRITE OPERATIONS
- STORE SEGMENTS of data for compaction
- IO is "relatively" slow in erlang
- USE a “native KV store” as a nif.
INDEX OPERATIONS

change reader

send /collect changes

indexer

get changes

DB

update

share state

View Group

READER

READER

enki
Credit Flow Based

- The View group keep the state
- View group is created on demande
- kept open until it has readers
- Indexer ask for updates
- Read functions (Map functions) are processed in //

INDEX OPERATIONS
Added 2 features:

- MOVE: move doc(s) to another node or database (like copy but with delete)
- User hooks functions (run in background) using hooks: https://github.com/barrel-db/hooks

- Partition on demand

- Decision depends on the application needs

**NEW FUNCTIONS**
CHANGES HANDLER

change dispatcher

broadcast changes

DB

subscriber
- Use the sequence index
- Changes load balancing
- Consumer subscribe on patterns (delete, update, ...)
- Create changes Load Balancer on demand
- Allows remote nodes to subscribe to a queue
- Based on primer (release on March 2016)
Use the sequence index
changes load balancing
consumer subscribe on patterns (delete, update, ...)
Create changes Load Balancer on demand
Allows remote nodes to subscribe to a queue
Based on primer (release on March 2016)
- inherited the HTTP api in mochiweb
- small changes to makes the server more resilient
- chatterbox
- wip in cowboy.
- yaws?
P2P
Over HTTP

Replication is the core

Each node can replicate each other

PUSH/PULL

Chained replication

P2P
Based on the change feed

- fetch the revisions and their attachments not present on the node

- continuous or not

- try to collect multiple docs at once

- use hackney: http://github.com/benoitc/hackney

- Use a Flow-based pattern instead of a classic pool
REPLICATION OPERATIONS

DB SOURCE

- get changes

- fetch docs

DB TARGET

- push docs

- notify changes

replication worker

replication proxy
Replication state is stored at least on one node.

- Checkpoints
- Get the revisions not actually stored on the nodes ("_rev_diffs")
- The replication proxy maintains routes.
- Build replication chains by replicating status.
Barrel

HTTPS://BARREL-DB.ORG

Enki Multimedia

HTTP://ENKIM.EU