Invitation to a New Kind of Database

Sheer El Showk

Cofounder, Lore Ai

www.lore.ai

We're Hiring!

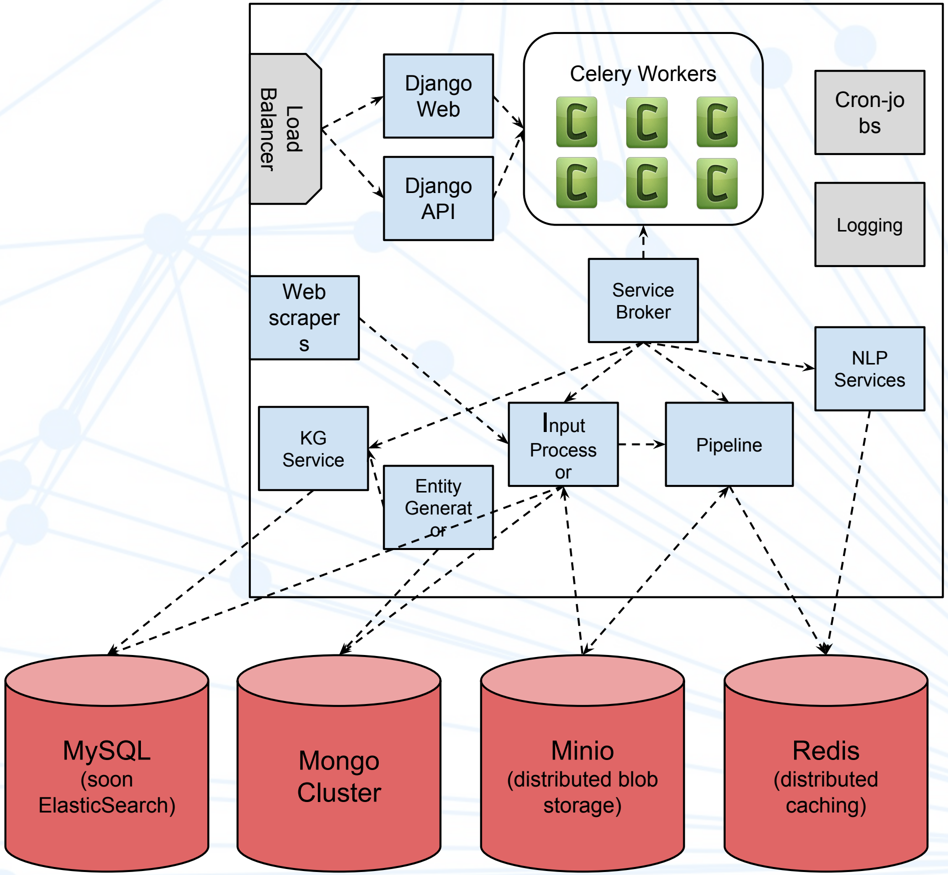
Overview

1. Problem statement (~2 minute)
2. (Proprietary) Solution: Datomics (~10 minutes)
3. Proposed Open Source Solution: (~8 minutes)
    a. Easy Version(s)
    b. Hard Version(s)
4. Discussion & More Brain Storming (after talk & all of tomorrow)

The hard thing about hard things...

LORE AI

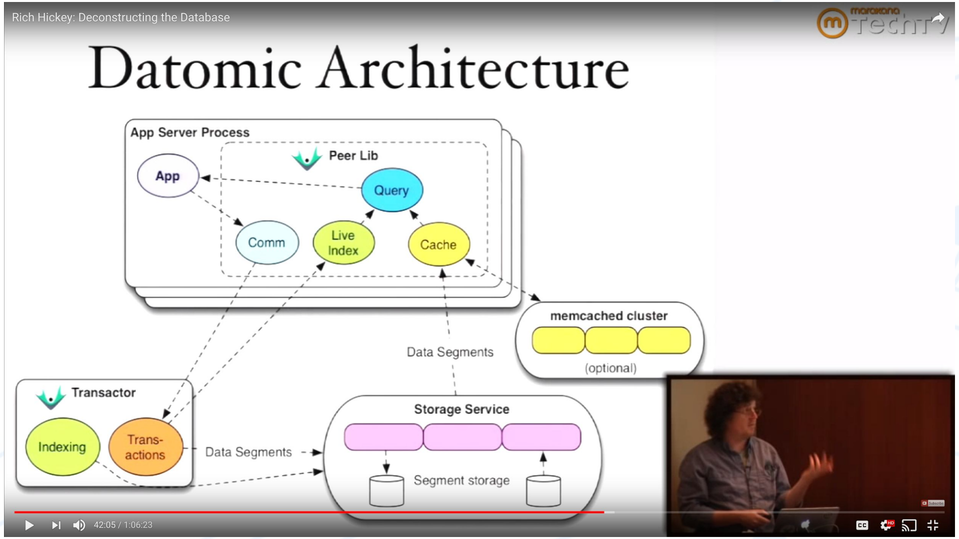
- ➔ NoSQL vs SQL
- ➔ Data-model vs database type
  - ◆ Column vs Row vs Document
- ➔ Backups: consistent snapshots across different (types of) DBs.
- ➔ Versioning: changes over time
- ➔ Trade-offs:
  - ◆ Scaling
  - ◆ Consistency



LORE AI

Deconstructing the Database

Rich Hickey
(inventor of Clojure)



https://youtu.be/Cym4TZwTCNU

Outdated, Bad Ideas:

- Changing data in place (mutability)
- Fixed schema structure
- Disk locality is an outdated idea (networks are faster than disks!)
- Database must be a server (why not db as a library?)
- Reads & Writes done by same system

Even modern "SQL/NoSQL" DBs make out-dated trade-offs (e.g. consistency vs scalability)

⇒ Can we come up with more

Disclaimers

1. I have no affiliation with Datomic
2. I’ve never used Datomic
3. I’m compressing Hickey’s very nice talk into 10 min

   ⇒ Missing lots of important ideas/details

4. I’ve only been thinking about this for a few weeks (on & off)

LORE AI

Hickey's Solution I: Fact-Based Schema

- "Atomic" unit of data (datom) is a "fact":

Transaction Time	Entity (subject)	Property/Relation	Value
12234	334	firstName	sheer
12235	334	cofounderOf	445 (Lore)

- Fact "values" can be literals, lists, types or Ids of other entities (i.e. relations).
- Encode arbitrary schema but additionally includes "time" dimension.
- Don't confound data with encoding.

"Equivalent" JSON:

```
{
    _id:  334,
    first_name:  "sheer",
    ...
    company: {
        name:  "Lore",
        ….
    }
    ….
}
```

Hickey’s Solution II: Immutable Data

LORE AI

- All writes add/retract a “fact” from DB
- Writes are appended to a “log”

  ⇒ append only system
- Deletions are just a series of “retractions”
- Operations (add/del) are “time-stamped”

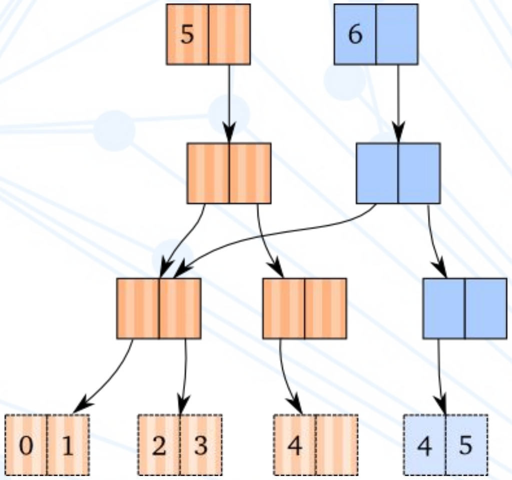
Time	Op	Ent	Rel	Value
123	add	sheer	email	sheer@gmail.com
124	add	sheer	email	sheer@yahoo.com
125	del	sheer	email	sheer@gmail.com
126	add	sheer	livesIn	Paris

Data is never deleted!

- The database is just a log of very granular updates of facts.
- Consistency is trivial: select a timestamp and read log up to timestamp.

Hickey's Solution III: Structural Sharing

- Consistent reads require versioning (timestamps) so what about an index?
- Hickey (+Phil Bagwell) invented/improved "HMAT"
  - Hash-mapped Array Trie
  - Structural sharing for efficiently versioning indices
  - High (32-way) branching ratio for shallow trees
- Querying element at fixed time-stamp only requires accessing simple path on tree
  - Cache/access subset of index as required.
- Universal fact-based schema:
  - only need fixed number (6) of indices
  - composite index on ent-rel-value, rel-ent-value, etc.



https://hypirion.com/musings/understanding-persistent-vector-pt-1

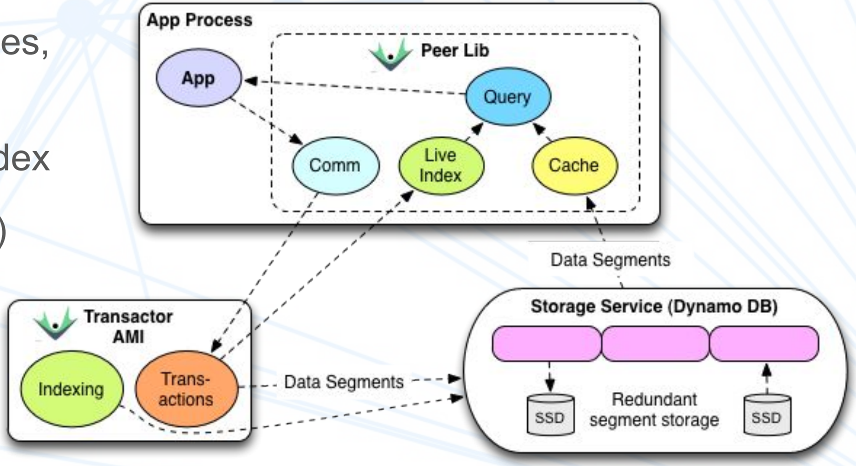
Hickey's Solution IV: Separate Reads & Writes

- Reads, queries and index lookups happen in client process (i.e. here client means app server not user/browser).
- Client-side library provides in-process indexing, queries, aggregations, etc
- Granular data model (fact-based) and incremental index

  ⇒ efficient caching of working set (facts & sub-trees)
- Consistency is trivially assured

  ⇒ each timestamp is consistent "snapshot" of DB.
- Trivial scalability

  ⇒ every app server is a processing peer.
- Only bottleneck is Transactor but it just appends updates to a log (can be made very fast).



How much of this can we replicate?

Improve?

LORE AI

Proposal

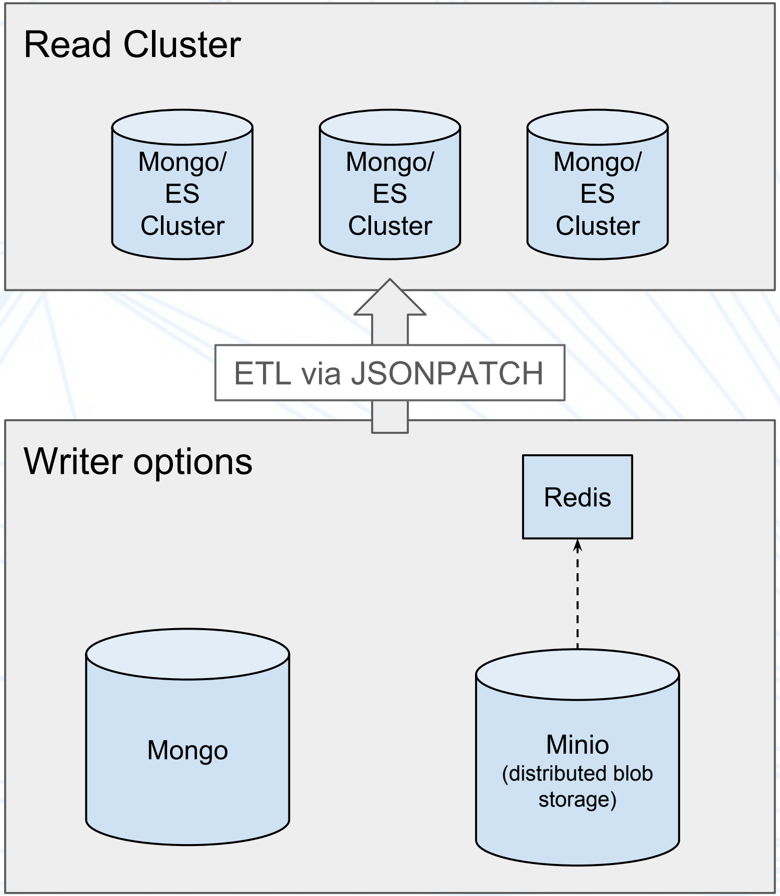
1. Versioned Document Store (“easy”)
   a. Update log via JSON-patch events
   b. Instantiate Documents in JSON-store (mongo) for near real-time indexing
2. Versioned Relational Document Store (“hardish”)
   a. Add “real-time” indexing to the above and restrict to “shallow” json linked by relations.
3. Fully Versioned Indices (“hard”)
   a. Implement HMAT or similar (see https://github.com/tobgu/pyrsistent)
4. Make it all efficient (“very hard”)
   a. Cythonize
   b. Tweak caching, disk layout, etc.

Versioned JSON Store

- Two ideas for write server ("Transactor"):
  - Append-only Mongo Collection
  - Redis-queue persisted to S3/Minio flat file
- Format:
  - Json-patch files ("transaction")
  - Json-schema library to validate
- Read/Index servers:
  - Cluster of Mongo/ES servers "instantiating" docs using jsonpatch library.
  - Indexing "almost" real-time but only "current" timestamp.

Only capturing some benefits (little contention, versioning, some schema independence).

- Only server-side processing.
- No versioned index.
- Essentially: just versioned replication



LORE AI

Versioned Relational JSON Store

- Enforce “shallow” JSON
  - Only strings, ints, lists, etc as values
  - Dicts/subdocs replaced by “ids”
- Index in real-time:
  - Directly index append-log collection
  - Manual index in redis
  - Index individual “facts” not just docs
- Transactor manually validates patched JSON:
  - Enforce existence on IDs (foreign keys)
  - Ensure patch respects schema

Sample patch:

```
{
   _id:  334,
   prev_checksum: “6d96617b37f4f662783c957”,
   patches: [
      {“op”: “add”, “path”: “/first_name”, “value”: “sheer”},
      {“op”: “add”, “path”: “/company”, “value”: Id(443)},
   ]
}
```

Notes

- JSON schemas are JSON docs so can be versioned as special collection in DB.
- Subtle issue: do we enforce FK constraints if doc/entity “retracted”?
- Indexing/reads are still largely centralized (redis/mongo index on append-log).

Fully Versioned Index

Core Ideas:

- Can we move in-process?
  - HMAT or other persistent index?

    (see https://github.com/tobgu/pyrsistent)
- Cache/access individual facts or index subtrees directly in process (with two tiered external cache like Datomics).
- Need to implement in-memory query engine using lazy-access cached persistent index.
  - Pandas (like?)
  - See https://github.com/tobgu/qcache
- Transactor or Indexer has to be able to merge “live” index (recent facts) with persistent index.

Make It All Very Efficient

- Cythonize (what parts?)
  - Jsonpatch
  - Schema validation
- Use dicts/native-types instead of JSON
- Fast serialization: msgpack, pickle, etc..
- Can we borrow existing OSS technologies for indexing, etc..

Some Comments

LORE AI

- An index is a tree:
  - Represent in JSON/dict and version/store like log?
- Schemas are also just documents
  - Can version and manage as a special collection
- Transactions are also “first class” so can carry additional metadata
- More complex ops (+=1) require Transactor to translate op into atomic updates.

Interested in working on this?

Contact me!

Slides will go on blog.lore.ai

Bye!