



The ultimate guide to relaxation

by Rob de Boer

Goals

- Look beyond traditional database techniques
- Consider an easy storage alternative
- ...of course in a relaxed state of mind ;)

Agenda

- Brief overview
- Key concepts
- See it in action (demo)
- Technical overview
- Conclusion
- Resources / Q&A

What is it all about?

BRIEF OVERVIEW

What is it?

- Document-oriented database
- Written in Erlang
- Schema-free
- Accessible via RESTful HTTP/JSON
- Distributed and fault-tolerant
- Reliable and efficient
- Query-able via Javascript

What is it *not*?

- A relational database
- A *replacement* for relational databases
- A seamless persistence layer for OO programming languages

How does it work?

KEY CONCEPTS

Document

- Object with named fields
- Identified by a unique ID

Example:

"Subject": "I like Plankton"

"Author": "Rusty"

"PostedDate": "5/23/2006"

"Tags": ["plankton", "baseball", "decisions"]

"Body": "I decided that I don't like baseball. I like plankton."

Views

- Add structure to semi-structured documents
- Uses Javascript
- Can be built ad-hoc
- Don't affect the underlying document
- Multiple views on the same document

Schema-free

- Most web applications are document oriented
- When requirements change:
 - SQL database schemas need to be updated (possibly on multiple hosts)
 - Document oriented databases can have “old” and “new” document types coexist

Distributed

- Peer-based
- Bi-directional replication
- Conflict detection
- Replication is incremental and fast
 - Copies only changed documents and even fields
- Works without special planning

See it in action

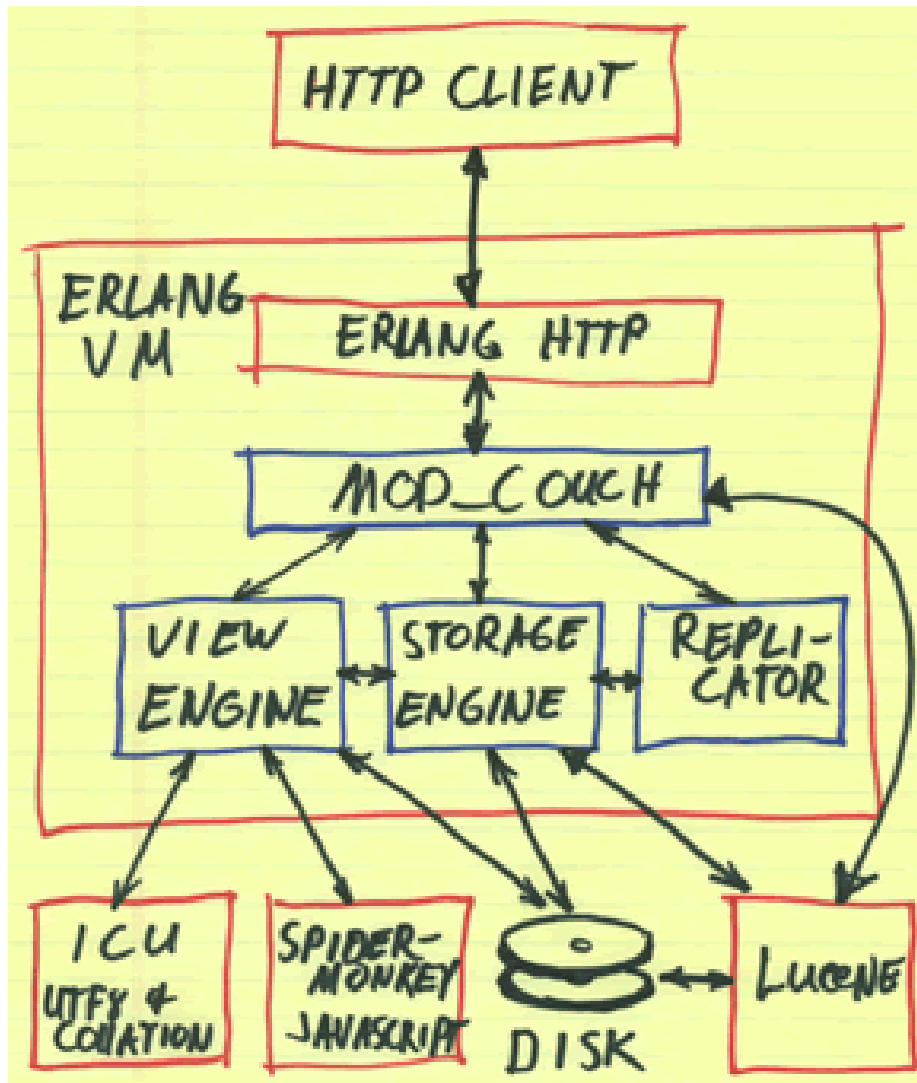
DEMO



A more detailed look

TECHNICAL OVERVIEW

Architecture



Document storage

- Primary unit of data
- Identified by ID and revision
- Named fields and attachments
- Metadata
- Varying datatypes:
 - Text, number, boolean, date, list, maps
- Lockless and optimistic update model
 - Conflicts can be merged

ACID

- All ACID properties are supported
- Document updates are serialized
 - Except for binary blocks
- Multi-Version Concurrency Control (MVCC)
 - Document ID, Sequence ID
 - Snapshot reads
- Transactional commits

Compaction

- Wasted space recovery
- Scheduled or on threshold
- Active data cloned to new file
- Database remains completely online

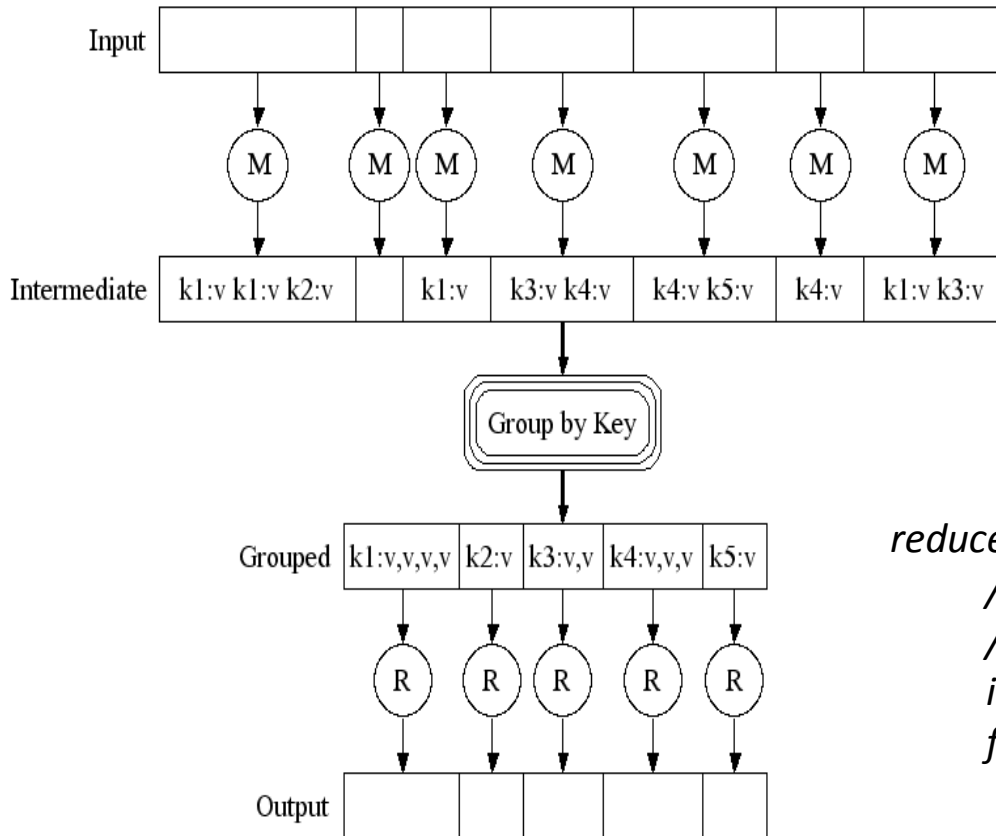
View model

- Aggregates and reports on documents
- Can be built dynamically on-demand
- Document remains unaffected
- Different views on the same document
- Views = **design document**
 - So they replicate too!

Defining views

- Javascript functions
- “Map” part of a map-reduce system

MapReduce



```
map (String input_key, String input_value):  
    // input_key: document name  
    // input_value: document contents  
    for each word w in input_value:  
        EmitIntermediate(w, "1");
```

```
reduce (String output_key, Iterator intermediate_values):  
    // output_key: a word  
    // output_values: a list of counts  
    int result = 0;  
    for each v in intermediate_values:  
        result += ParseInt(v);  
    Emit(AsString(result));
```

View indices

- Dynamic view generation can be expensive
- Solution: view indices
- Incremental updates
 - View-document comparison on Sequence ID's
- Concurrent reads and updates

Security and Validation

- Administrator access
- Reader access
 - Reader list protects documents
 - User credentials
- Update validation

Distributed updates and replication

- Peer-based distribution
- Efficient and reliable
- Offline usage
- Incremental replication
 - Only fields and binary data that have changed
- Partial replication supported

Conflict detection

- Update conflicts treated as a common state, not exceptional
- Conflicting documents can coexist
- Database itself determines the “winning” document for views
- Conflicted documents can still be accessed
 - Usually removed during compaction

Erlang

- Functional, concurrent programming
- Extreme emphasis on reliability and availability
- Lightweight “processes” and messaging
- No shared state threading
- All data is immutable

So how useful is it?

CONCLUSION

Conclusion

- Interesting change from traditional RDBMS
- Still incubated, but looks mature
- Quite useful if:
 - You “only” need a database for storage
 - You don’t really care too much about highly structuring data
 - You want easy clustering and fail-over

Resources

- Site: <http://couchdb.apache.org/>
- Book: <http://books.couchdb.org/relax/>
- Wiki: <http://wiki.apache.org/couchdb/>

Q&A

Any questions / remarks?