ECE464 – Final Project Proposal

Jonathan Lam, Derek Lee, Victor Zhang

2021/11/08

1 Overview

We want to build a simple NoSQL database structured after MongoDB. This database will be in the form of a Rust library that supports basic CRUD operations similar to MQL operations and a document-based data model. Indices will have to be explicitly requested to be built by the user, and will be implemented using a B-tree data structure.

2 Functional requirements

The database should support the basic CRUD operations and MQL-like queries. The data must be persistent (i.e., the database should not be solely in-memory).

For sake of time, many non-essential features of MongoDB may not be implemented: a custom query language (like MQL), transactions, concurrency, ACID/BASE principles, replica sets, sharding, and cross-table operations such as $lookup.

3 Tentative architecture

3.1 Data structures

- Document (in-memory): hashtable or association list mapping field names to (type, value) pairs (since Rust is strictly-typed)
- Document (physical): JSON or BSON
- Collection: vector of documents
- Index: B-tree
- Query: information about collection(s), list of constraints (each constraint contains information about the field, operator, type, value), ordering
- Schema: (none?)

3.2 Architecture implementation details

Physical storage of records will be similar to MongoDB's MMAPv1 storage engine, which uses power-of-two fixed-size blocks and a residency bitmap (i.e., a free-blocks list); each document (after creation or update) will occupy the next free block that has size of the smallest power of two greater than or equal to the size of the document. This allows for some tradeoff between time and space efficiency. There is no provision against fragmentation due to an accumulation of free blocks. This scheme also prevents against an efficient clustered B-tree index, which would probably be made much more complex than if fixed-size records were necessary (MongoDB doesn't have clustered indices anyway, presumably for this reason).

Indices will be created on demand. Indices may comprise one or more fields (multi-field comparisons will be are lexicographic). If no indices match the query constraints (i.e., no index contains a subset of the fields on which a constraint is placed), then the collection is scanned sequentially. Fields present in a query but missing in a document will return a special NOEXIST value (this is subject to change – not sure how MongoDB handles this case).

Queries contain information about the collection to query, a set of constraints on fields, ordering/limit operations, and aggregation operations. (These are analogous to the FROM, WHERE, ORDER BY, LIMIT, GROUP BY, and aggregate operations in SQL.) As is traditionally done in MongoDB, the the structure of the query constraints models the structure of the data, using a similar recursive structure with constraints in the place of values. (Alternatively, nested constraints may be modeled using a "field path," e.g., "field1.field2.field3".)

4 Timeline

- 11/09: Finalized proposal
- 11/16: Implement data structures for documents, queries, collections, set up API, write unit tests
- 11/23: Implement MMAPv1, B-tree (indices), BSON
- 11/30: Implement CRUD operations
- 12/07: Finish implementation of queries, work on presentation
- 12/14: Presentation

5 Evaluation

A test suite will be used to check the correctness of various queries specified in the minimal query language. A series of benchmarks may also be applied to test the empirical scalability of the system, and the scalability may be compared to existing RDBMSs and NoSQL DBMSs.