On the storage, Organization and Index of MongoDB

By Enbo Zhou
Institute of Remote Sensing and Geographic Information System,
Peking University, Beijing, China

Content

The Introduction to MongoDB

The Introduction to MongoDB

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.
- A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

Document Example

```
{
  name: "sue",                          <-- field: value
  age: 26,                              <-- field: value
  status: "A",                          <-- field: value
  groups: [ "news", "sports" ]          <-- field: value
}
```

Key Features

- High Performance
  - MongoDB provides high performance data persistence. In particular,
    - Support for embedded data models reduces I/O activity on database system.
    - Indexes support faster queries and can include keys from embedded documents and arrays.
- High Availability
  - To provide high availability, MongoDB's replication facility, called replica sets, provide:
    - automatic failover.
    - data redundancy.
  - A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.
- Automatic Scaling
  - MongoDB provides horizontal scalability as part of its core functionality.
    - Automatic sharding distributes data across a cluster of machines.
    - Replica sets can provide eventually-consistent reads for low-latency high throughput deployments.

Some Concepts

- Document
  - Just like the record in relation database.
- Collection
  - Just like the table in relation database.
- Database
  - Is same as the database in relation database.

The Storage and Organization of MongoDB

The Storage of MongoDB

- MMAPv1 Storage Engine
- WiredTiger Storage Engine

MMAPv1 Storage Engine

- MMAPv1 is MongoDB's original storage engine based on memory mapped files. It excels at workloads with high volume inserts, reads, and in-place updates. MMAPv1 is the default storage engine in MongoDB 3.0 and all previous versions.

Record Storage Characteristics

- All records are contiguously located on disk, and when a document becomes larger than the allocated record, MongoDB must allocate a new record. New allocations require MongoDB to move a document and update all indexes that refer to the document, which takes more time than in-place updates and leads to storage fragmentation.
- Changed in version 3.0.0.
- By default, MongoDB uses Power of 2 Sized Allocations so that every document in MongoDB is stored in a record which contains the document itself and extra space, or padding. Padding allows the document to grow as the result of updates while minimizing the likelihood of reallocations.

Record Allocation Strategies

- MongoDB supports multiple record allocation strategies that determine how mongod adds padding to a document when creating a record. Because documents in MongoDB may grow after insertion and all records are contiguous on disk, the padding can reduce the need to relocate documents on disk following updates. Relocations are less efficient than in-place updates and can lead to storage fragmentation. As a result, all padding strategies trade additional space for increased efficiency and decreased fragmentation.
- Different allocation strategies support different kinds of workloads: the power of 2 allocations are more efficient for insert/update/delete workloads; while exact fit allocations is ideal for collections without update and delete workloads.

The power of 2 allocations

- With the power of 2 sizes allocation strategy, each record has a size in bytes that is a power of 2 (e.g. 32, 64, 128, 256, 512 ...2MB). For documents larger than 2MB, the allocation is rounded up to the nearest multiple of 2MB.
- The power of 2 sizes allocation strategy has the following key properties:
  - **Can efficiently reuse freed records to reduce fragmentation**. Quantizing record allocation sizes into a fixed set of sizes increases the probability that an insert will fit into the free space created by an earlier document deletion or relocation.
  - **Can reduce moves.** The added padding space gives a document room to grow without requiring a move. In addition to saving the cost of moving, this results in less updates to indexes. Although the power of 2 sizes strategy can minimize moves, it does not eliminate them entirely.

No Padding Allocation Strategy

- For collections whose workloads do not change the document sizes, such as workloads that consist of insert-only operations or update operations that do not increase document size (such as incrementing a counter), you can disable the power of 2 allocation using the collMod command with the noPadding flag or the db.createCollection() method with the noPadding option.

WiredTiger Storage Engine

- WiredTiger uses document-level concurrency control for write operations. As a result, multiple clients can modify different documents of a collection at the same time.
- For most read and write operations,WiredTiger uses optimistic concurrency control. WiredTiger uses only intent locks at the global, database and collection levels. When the storage engine detects conflicts between two operations, one will incur a write conflict causing MongoDB to transparently retry that operation.

Snapshots and Checkpoints

- WiredTiger uses MultiVersion Concurrency Control (MVCC). At the start of an operation, WiredTiger provides a point-in-time snapshot of the data to the transaction. A snapshot presents a consistent view of the in-memory data.
- When writing to disk, WiredTiger writes all the data in a snapshot to disk in a consistent way across all data files. The now-durable data act as a checkpoint in the data files. The checkpoint ensures that the data files are consistent up to and including the last checkpoint; i.e. checkpoints can act as recovery points.

Compression

- With WiredTiger, MongoDB supports compression for all collections and indexes. Compression minimizes storage use at the expense of additional CPU.
- By default, WiredTiger uses block compression with the snappy compression library for all collections and prefix compression for all indexes.
- For collections, block compression with zlib is also available. To specify an alternate compression algorithm or no compression, use the storage.wiredTiger.collectionConfig.blockCompressor setting.
- For indexes, to disable prefix compression, use the storage.wiredTiger.indexConfig.prefixCompression setting.

WiredTiger Storage Engine

•Compression settings are also configurable on a per-collection and per-index basis during collection and index creation.See create-collection-storage-engine-options and db.collection.createIndex() storageEngine option.

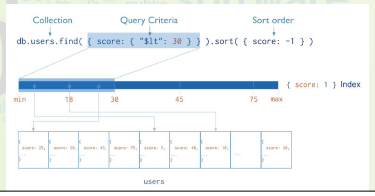
•For most workloads, the default compression settings balance storage efficiency and processing requirements.

•The WiredTiger journal is also compressed by default. For information on journal compression.

The Index of MongoDB

The Index of MongoDB

- Fundamentally, indexes in MongoDB are similar to indexes in other database systems. MongoDB defines indexes at the collection level and supports indexes on any field or sub-field of the documents in a MongoDB collection.



Index Types

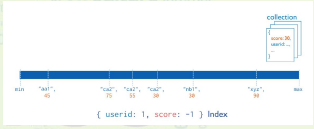
- Default _id
  - All MongoDB collections have an index on the _id field that exists by default. If applications do not specify a value for _id the driver or the mongod will create an _id field with an ObjectId value.
  - The _id index is unique and prevents clients from inserting two documents with the same value for the _id field.

Index Types

- Single Field
  - In addition to the MongoDB-defined _id index, MongoDB supports the creation of user-defined ascending/descending indexes on a single field of a document.
  - For a single-field index and sort operations, the sort order (i.e. ascending or descending) of the index key does not matter because MongoDB can traverse the index in either direction.

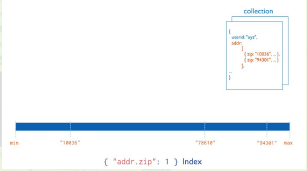
Index Types

- Compound Index
  - MongoDB also supports user-defined indexes on multiple fields, i.e. compound indexes.
  - The order of fields listed in a compound index has significance. For instance, if a compound index consists of {userid: 1, score: -1 }, the index sorts first by userid and then, within each userid value, sorts by score.
  - For compound indexes and sort operations, the sort order (i.e. ascending or descending) of the index keys can determine whether the index can support a sort operation. See Sort Order (page 491) for more information on the impact of index order on results in compound indexes.



Index Types

- Multikey Index
  - MongoDB uses multikey indexes to index the content stored in arrays. If you index a field that holds an array value, MongoDB creates separate index entries for every element of the array. These multikey indexes allow queries to select documents that contain arrays by matching on element or elements of the arrays. MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value; you do not need to explicitly specify the multikey type.



Index Types

- Geospatial Index
  - To support efficient queries of geospatial coordinate data, MongoDB provides two special indexes: 2d indexes that uses planar geometry when returning results and 2sphere indexes that use spherical ge-ometry to return results.
- Geospatial Index Types
  - 2dsphere Indexes
  - 2d Indexes
  - geoHaystack Indexes
  - 2d Index Internals

Index Types

- Text Indexes
  - MongoDB provides a text index type that supports searching for string content in a collection. These text indexes do not store language-specific stop words (e.g. “the”, “a”, “or”) and stem the words in a collection to only store root words.
- Hashed Indexes
  - To support hash based sharding (page 686), MongoDB provides a hashed index (page 504) type, which indexes the hash of the value of a field. These indexes have a more random distribution of values along their range, but only support equality matches and cannot support range-based queries.

Index Properties

- In addition to the numerous index types (page 488) MongoDB supports, indexes can also have various properties. The following documents detail the index properties that you can select when building an index.
  - ➢TTL Indexes The TTL index is used for TTL collections, which expire data after a period of time.
  - ➢Unique Indexes A unique index causes MongoDB to reject all documents that contain a duplicate value for the indexed field.
  - ➢Sparse Indexes A sparse index does not index documents that do not have the indexed field.


mongoDB
The End! Thank you!
open
source
free
culture
internet
digital
users
code
development
available
cultural
content
information
research
also