Data Management in the Cloud

Part 2

Data Models

Document: MongoDB

1

1

Creating Index

- Use of a multi-key index speeds up collection-oriented queries

```
db.collection.createIndex(
 { "a": 1 },
 {
    unique: true,
    sparse: true
 }
)
```

  - The '1' means increasing order

unique: Creates a unique index so that the collection will not accept updates that result in duplicate entries in the indexed field.

Sparse: the index only references documents with the specified field.

2

Soure: https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/#mongodb-method-db.collection.createIndex

2

Document Evolution

```
post = {author: "mike",
        date: new Date(),
        text: "another blog post...",
        tags: ["mongodb"],
        title: "Introduction to MongoDB"}
db.posts.save(post)
```

New Attribute

- Flexible schema
  - documents within the same collection can have different attributes
  - document attributes can simply be added or removed
  - evolution performed when **`save()`** is invoked
- Save with write concern:

Majority: returns after the write propagates to a majority of replicas

```
db.products.save(
   { item: "envelopes", qty : 100, type: "Clasp" },
   { writeConcern: { w: "majority", wtimeout: 5000 } }
)
```

OR the method times out after 5 seconds

3

3

Atomic Update Modifiers

```
var c = {author: "eliot",
         date: new Date(),
         text: "Great post!"}

db.posts.update({_id: post._id},
                {$push: {comments: c}})
```

query selector

update value

- **`$push`** adds (appends) an object to an array
  - For this update, client does not need to worry about locking of documents or contention of updates

4

4

Approaches to Transactions in MongoDB

The approaches available to achieve transactions in MongoDB

Source: MongoDB

5

What is a Transaction?

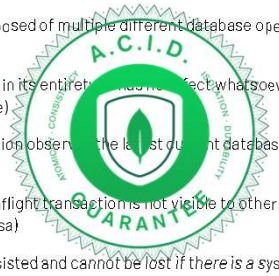
A single unit of logic composed of multiple different database operations, which exhibits the following properties:

Atomic: either completes in its entirety or has no effect whatsoever (rolls back and is not left only partially complete)

Consistent: each transaction observes the last current database state in the correct write ordering

Isolated: the state of an inflight transaction is not visible to other concurrent inflight transactions (and vice versa)

Durable: changes are persisted and cannot be lost if there is a system failure



Source: MongoDB

6

Two Approaches to Transactions

MongoDB Document

>= 1.0 MongoDB

MongoDB Transactions

>= 4.2 MongoDB

Source: MongoDB

7

Transactions with a document

Patient records from a doctor's visit.

date of visit

doctor's notes

drugs prescribed

current weight

Update the document in one operation

```
patients collection
{
  "_id": 2395652,
  "name": "AJ",
  "current_weight": 210,
  "next_physical": "2021-06-13",
  "visits": [
    { "date": "2018-12-24",
      "notes": "Torn right calf" },
    { "date": "2020-02-01",
      "notes": "Strained left hamstring" }
  ],
  "drugs": [
    { "date": "2018-12-24",
      "drug": "Ibuprofen" },
    { "date": "2020-02-01",
      "drug": "Paracetamol" }
  ]
}
```

Source: MongoDB

8

Transactions with a document

Atomic: all writes to one document are done at once

Consistency: no dependency on other documents

Isolation: document being modified not seen by other reads

Durability: guaranteed by doing a write with a "majority" concern

```
patients collection
{
  "_id": 2395652,
  "name": "AJ",
  "current_weight": 210,
  "next_physical": "2021-06-13",
  "visits": [
    { "date": "2018-12-24",
      "notes": "Torn right calf" },
    { "date": "2020-02-01",
      "notes": "Strained left hamstring" }
  ],
  "drugs": [
    { "date": "2018-12-24",
      "drug": "Ibuprofen" },
    { "date": "2020-02-01",
      "drug": "Paracetamol" }
  ]
}
```

- Design your model to have documents embedding the different relational tables updated together

Source: MongoDB

9

Transactions with a document

Patient records and payments from a doctor's visit.

date of visit

doctor's notes

drugs prescribed

current weight

payment information

```
patients collection
{
  "_id": 2395652,
  "name": "AJ",
  "current_weight": 210,
  "next_physical": "2021-06-13",
  "visits": [
             { "date": "2018-12-24",
             "notes": "Torn right calf" },
             { "date": "2020-02-01",
             "notes": "Strained left hamstring" }
],
  "drugs": [
             { "date": "2018-12-24",
             "drug": "Ibuprofen" },
             { "date": "2020-02-01",
             "drug": "Paracetamol" } ]
}
payments collection
{
  "date": "2020-02-01",
  "patient_id": 2395652,
  "amount": 250.00,
}
```

Source: MongoDB

10

Transactions with a MongoDB Transaction

Atomic: all writes to all documents are committed at once

Consistency: all checks are done within the transaction

Isolation: guaranteed through a "snapshot" isolation level

Durability: guaranteed by default, the write has a "majority" concern.

```
patients collection
{
  "_id": 2395652,
  "name": "AJ",
  "current_weight": 210,
  "next_physical": "2021-06-13",
  "visits": [
             { "date": "2018-12-24",
             "notes": "Torn right calf" },
             { "date": "2020-02-01",
             "notes": "Strained left hamstring" }
],
  "drugs": [
             { "date": "2018-12-24",
             "drug": "Ibuprofen" },
             { "date": "2020-02-01",
             "drug": "Paracetamol" } ]
}
payments collection
{
  "date": "2020-02-01",
  "patient_id": 2395652,
  "amount": 250.00,
}
```

Source: MongoDB

11

Write Concern…

- Errors Ignored
  - MongoDB does not acknowledge write operations.
  - Client cannot detect failed write operations – including connection errors, duplicate key exceptions…
  - But fast performance…
- Unacknowledged
  - MongoDB does not acknowledge the receipt of write operation.
  - Similar to *errors ignored*; however, drivers attempt to receive and handle network errors when possible.
- Acknowledged
  - MongoDB confirms the receipt of the write operation.
  - Allows clients to catch network, duplicate key, and other errors.
  - the default write concern.

http://docs.mongodb.org/manual/reference/write-concern/

12

12

Write Concern…

- Journalled
  - MongoDB acknowledges the write operation only after committing the data to the journal. Write is recoverable.
  - Write operations must wait for next journal commit. To reduce latency, MongoDB increases the frequency that it commits to the journal…
- Majority
  - Acknowledge that write has gotten to a majority of replicas
  - Say how many replicas must respond

13

13

Discussion Question

Describe a situation where you might use Errors Ignored or Unacknowledged write concern

Describe a situation where you might want Journalled or Majority write concern

14

14

Read Concern

- Local (default)

  The query returns data from the instance with no guarantee that the data has been written to a majority of the replica set members (i.e. may be rolled back).
- Majority

  The query returns the data that has been acknowledged by a majority of the replica set members. The documents returned by the read operation are durable, even in the event of failure.
- Linearizable

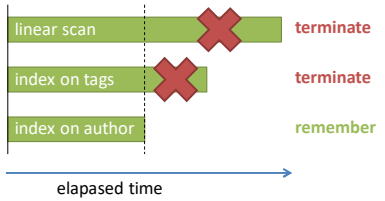
  The query returns data that reflects all successful majority-acknowledged writes that completed prior to the start of the read operation. The query may wait for concurrently executing writes to propagate to a majority of replica set members before returning results.

15

15

Query Optimizer

- No algebraic or cost-based query optimization
  - evaluator processes alternative plans in parallel
  - first plan to finish is remembered as most efficient
  - other plans are terminated
  - query plan is recorded – re-evaluate plan after 1000 writes, add, drop or rebuild index
- Find posts by Joe about database management systems
  - `db.posts.find({author: "joe", tags: "dbms"})`



16

Data File Allocation

- Memory-mapped storage engine
  - entire files are mapped into memory
- MongoDB stores each database as a number of files
  - **namespace** contains the catalog
  - **data files** contain the collections and documents
  -

```
 16MB blog.ns
 64MB blog.0
128MB blog.1
 16MB test.ns
  ...
```

double in size (up to 2 GB)

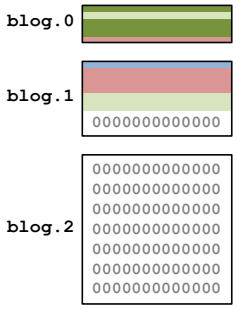
allocated for each database

http://docs.mongodb.org/manual/faq/storage/

17

17

Extent Allocation

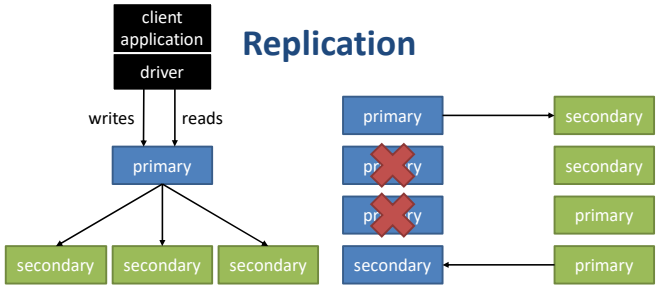


- Extents allocated into namespaces per collection
  - `blog.posts`
  - `blog.comments`
  - `blog.users`
  - `blog.$freelist`
  - 000 pre-allocated space
- Namespace metadata details stored in `blog.ns`
- Empty file is kept in order to avoid blocking on allocation

18

18

Replication



Replication provides redundancy and increases data availability

- "primary/secondary"-style replication, but roles can change
- primary receives all writes
- if primary fails, secondaries elect a new primary

http://docs.mongodb.org/manual/replication/

19

19

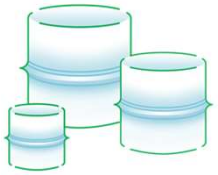
Read Preferences

Read Preference Mode	Description
primary	Default mode. All operations read from the current replica-set primary.
primaryPreferred	In most situations, operations read from the primary but if it is unavailable, operations read from secondary members.
secondary	All operations read from the secondary members of the replica set.
secondaryPreferred	In most situations, operations read from secondary members but if no secondary members are available, operations read from the primary.
nearest	Operations read from the nearest member of the replica set, irrespective of the member's type.

http://docs.mongodb.org/manual/core/read-preference/

20

20



Sharding in MongoDB: Why you should shard?

Increase volume of persisted data

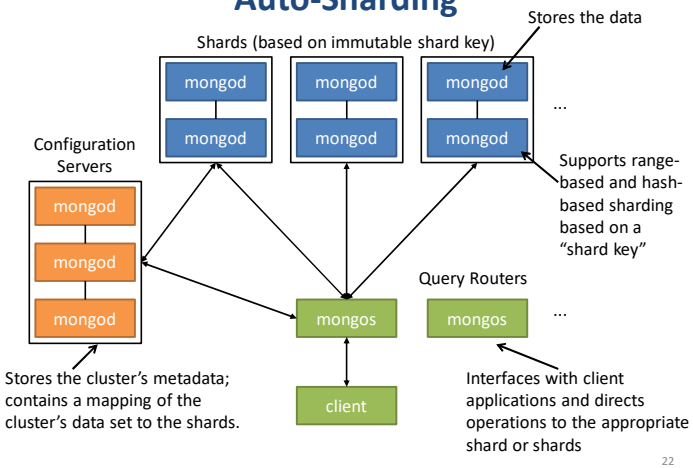
Increase/distribute throughput to scale both reads & writes

Decrease latency of both reads and writes

Source: MongoDB

21

Auto-Sharding



http://docs.mongodb.org/manual/sharding/

22

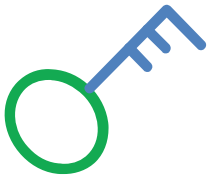
How is a shard key used?

The **shard key is used to route operations to the appropriate shard**. The shard key is used for reads and also for writes.

Source: MongoDB

23

What is the perfect shard key?

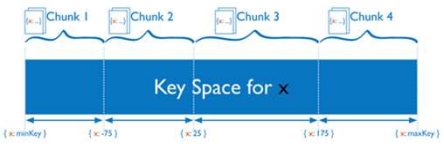


All **queries, inserts, updates, and deletes would each be distributed uniformly across all of the shards**. Operations would only ever go to the relevant shard with the data.

Source: MongoDB

24

Sharding strategies: Range sharding



Chunks, **contiguous range of shard key values**

Each **shard key range is associated to a chunk.**

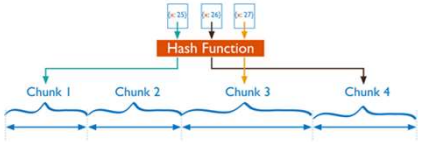
Each doc is assigned to a chunk per it's shard key

The database strives to balances these chunks across the shards.

Source: MongoDB

25

Sharding strategies: Hashed sharding



Single field hashed index or **compound field hashed index.**

Computes the hash value as the shard key value.

Better **data distribution** at the cost of **potentially more broadcast queries.**

Source: MongoDB

See: https://docs.mongodb.com/manual/core/ranged-sharding/

26

Other Features

- Capped collections
  - fixed-sized collection with wrap around (overwrite)
  - e.g., for website logging
- Unique indexes
- GridFS
  - file system API built on top of MongoDB
  - stores large binary data by chunking it up into smaller documents
- Two-dimensional geo-spatial indexing
  - e.g., Foursquare
- Views

27

27

Use Cases

- World Wide Web
  - scaling out
  - high volume
  - caching
- Less good at…
  - highly transactional
  - ad-hoc business intelligence
  - problems that require full SQL

28

28

References

- M. Dirolf: **Introduction to MongoDB**, *O'Reilly Webcast, 2010, available at:* http://www.youtube.com/watch?v=w5qr4sx5Vt0.
- http://docs.mongodb.org/manual/
- https://www.mongodb.com/presentations/mongodb-world18--eliot-horowitz-keynote (multi-document ACID Xacts)
- K. Seguin: The Little MongoDB Book, https://www.openmymind.net/mongodb.pdf

29

29

Resources

- **Resources:** The MongoDB Student Pack includes various free benefits for students, including MongoDB University On Demand, free MongoDB certification & $200 MongoDB Atlas credit. Students can access the offer here.
- **MongoDB University**: Free courses on a wide range of topics. The courses may also be of interest to the outreach program (as well as by UIUC students and faculty).
- **Books:**
  - MongoDB: The Definitive Guide, 3rd Edition (2019)
  - Practical MongoDB Aggregations (2021)
  - Designing Data-Intensive Applications (2017)

30

30