MongoDB

Richie Li, Infra team

Outline

1. MongoDB basic
2. MongoDB advanced (distributed computing)
    1. MongoDB replication
    2. MongoDB sharding
3. Orphan problem
4. MongoDB best practice

The mindset

Limit the data scope.

Use partition key.

Use simple features.

The cluster

```
(u'sweeper-notifications', u'notification', 8713.42),

(u'sweeper', u'productfeedtile', 1536.03),

(u'sweeper-user', u'userwishedcontest', 1315.07),

(u'sweeper', u'commercetransaction', 1305.56),

(u'sweeper', u'ordertrackinginfo', 1263.36),

(u'sweeper-merchant', u'merchanttransaction',
942.58),

(u'sweeper-merchant', u'productstatsbyday',
887.28),

(u'sweeper', u'parammapping', 880.73),

(u'sweeper', u'commercevariation', 859.29),

(u'sweeper-merchant', u'importjoberrorbucket',
749.28),

(u'sweeper', u'usertruetagaffinity', 742.46),

(u'sweeper', u'commerceticketreplybucket',
627.89),

(u'sweeper', u'contest', 478.07),

(u'sweeper', u'recentproductimpressions', 423.10),

(u'sweeper', u'contestphoto', 396.12),

(u'sweeper', u'contestinfo', 368.37),

(u'sweeper-user', u'user', 352.40),

(u'sweeper-merchant', u'orderlogisiticinfo',
309.70),

(u'sweeper', u'commerceticket', 291.99),

(u'sweeper', u'recentaddtocarts', 262.62)]
```

MongoDB basic

MongoDB is a NoSQL [1] document store.

Stores BSON (binary JSON, schema-less)

Cached in memory (like redis/memcached)

*Text search (like solr/elasticsearch)

Persistent storage

[1]https://en.wikipedia.org/wiki/NoSQL

MongoDB basic

MongoDB uses BSON, a binary format of JSON, with extended data types such as “ObjectId”, “Date”, “Timestamp”, etc. It is similar to ProtoBuf but much more loosely defined.

MongoDB basic

Index

{user_id: 1} Query by user_id. Index is ordered ascending.

{user_id: -1} Query by user_id. Index is ordered descending.

{user_id: 1, order_id: -1} Query by user_id; or user_id and order_id together. Index is ordered ascending using user_id first, then descending using order_id. **Order matters**!

{user_id: “hashed”} Query by user_id and single doc retrieval.

MongoDB basic

Index cont'd

"_id" is created automatically for new collection if you don't specify one. It is an index sorted in ascending and comes with unique constraint.

Should we use the auto generated "_id"? Try NOT to!!!

```
user_id = f.StringField(primary_key=True)
```

MongoDB basic

Index cont'd

Unique constraint enforce no dupes for a specific field.

```
db.coll.createIndex({email: 1}, {unique: true})
```

```
db.coll.createIndex({email: "hashed"}, {unique: true})
```
Won't work!

MongoDB basic

Index cont'd

DO NOT create too many indexes, although there is no hard limit. 2 or 3 for collection w/o many fields. A few more is ok if the collection have a lot more fields.

10+ indexes are bad indication of the collection design.

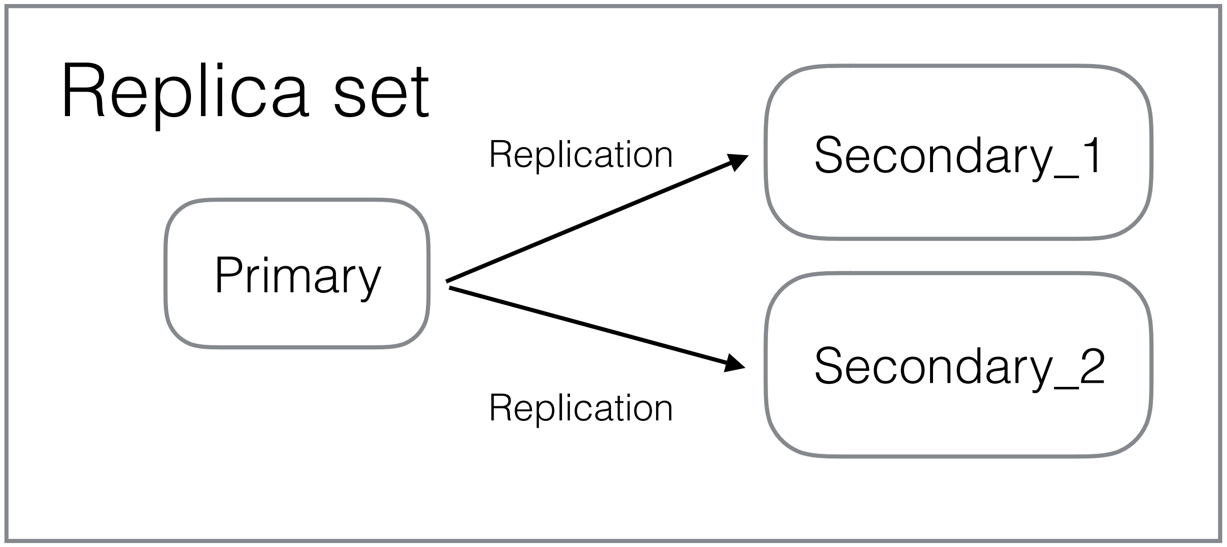
Use compound index!

MongoDB advanced (replication)

Replication is a data synchronization mechanism which maintains updated copies of the source data.

High availability

MongoDB advanced (replication)



MongoDB advanced (replication)

Read preference

This determines where you read data from the replica set.

primary (default), primaryPreferred, secondary, secondaryPreferred, nearest

MongoDB advanced (replication)

Write concern

“w”: “1” means primary only, “n” means extra secondary

“j”: “true” means written to journal (on disk)

MongoDB advanced (replication)

Secondary nodes are not meant for providing extra capacity (horizontal scaling).

Avoid reading from secondary if possible!

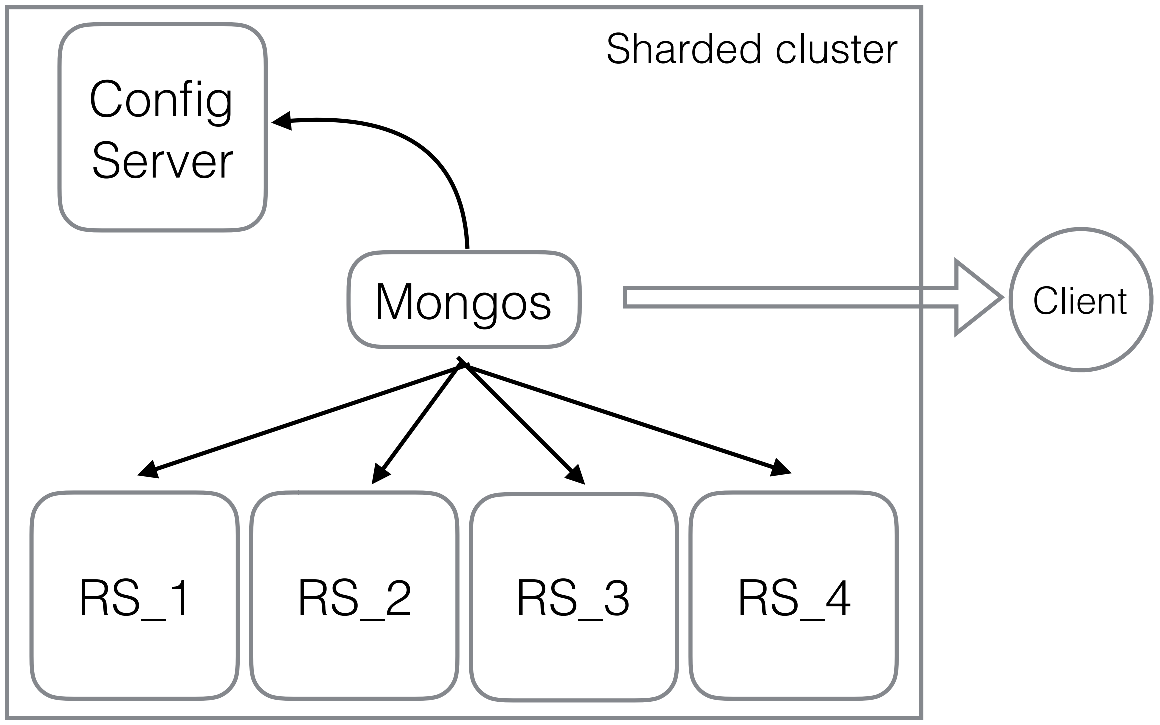
Avoid setting slave_ok!

MongoDB advanced (sharding)

Sharding is a data distribution/partition mechanism.

Horizontal scalability (extra capacity)

MongoDB advanced (sharding)



MongoDB advanced (sharding)

Should we shard every collection? (Absolutely not!!!)

Why?

What if collection is humongous? (hundreds millions or billions docs)

What if QPS is extremely high? (10k+)

When?

You have a good picture of your common query pattern. You are happy with your existing index design.

MongoDB advanced (sharding)

How?

- Shard key.
  - It determines where document is stored.
  - The shard key should have high cardinality and ideally uniform distribution.
  - Use compound shard key (compound index) if necessary.
  - Apply unique constraint if possible.

MongoDB advanced (sharding)

Isolated

Query is executed on one shard.

Broadcast (scatter/gather)

Query is executed on many (all) shards.

**Is it bad? Not necessarily and sometimes it is unavoidable.

MongoDB advanced (sharding)

Sharding Strategy

Hash:

Faster query for single doc retrieval. Uniformly distributed data. Use on hashed index.

Ranged query is likely scatter/gather.

Range (recommended):

Isolated query is common for range based sharding.

Auto/manual data balancing is needed.

MongoDB advanced (sharding)

The worst query

- Query w/o shard key and w/o index!

Why

- Aggregation type of job. (Use data warehouse)
- Production database is not meant for these work.

MongoDB orphan

Orphan document

Dupe or stale.

No meta information in config server.

Randomly scattered on different shards due to chunk balancing.

MongoDB orphan

When

Chunk migration is interrupted and the clean up process fails.

What

Query secondary w/ shard key return documents with same _id.

Query secondary w/o shard key return orphan only.

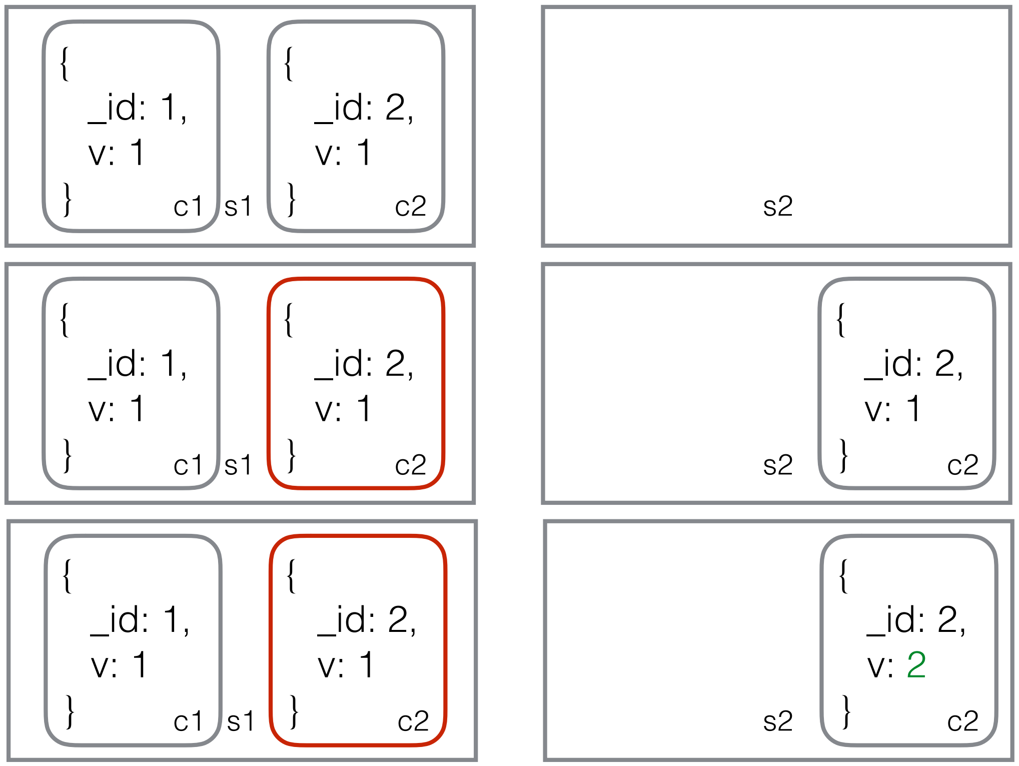
MongoDB orphan

Config Server

s1:[c1, c2]

s1:[c1], s2:[c2]

s1:[c1], s2:[c2]



Query:
1: {_id: 1} -> {_id: 1, v: 1}
1: {_id: 2} -> {_id: 2, v: 2}
2: {_id: {$gte: 1}} -> [{_id1: 1, v: 1}, {_id: 2, v: 1}, {_id: 2, v: 2}] Oops!!!
3: {v: 1}: -> [{_id1: 1, v: 1}, {_id: 2, v: 1}] Oops again and even worse!!!

MongoDB orphan

How

The golden rule is querying PRIMARY only.

Enable versioning.

(Infra will work on orphan clean up and prevention.)

MongoDB best practice

- Create index for your collection. Summarize your query access pattern and figure out the best indexes to use.
- If you don't create index, you should only query the collection by _id.
- Use compound index when single field index cardinality is small or your query always refer to more than one field.
- The order of the compound index matters. E.g. if you query by "a" or "a, b", the compound index should be {a: 1, b: 1} instead of {b: 1, a: 1}. PyMongo uses SON to ensure the ordering of the key in the dict.
- Use sorted index unless you only do single document query.
- Do not create too many indexes on the collection and do not create different index type on the same field. Try to avoid using advanced mongo index feature unless you have a thorough understanding of your business logic and the mongo implementation.

MongoDB best practice

- Always include index in your query. Otherwise you are doing full collection scan. For unsharded super tiny collection, this is fine.

- Avoid hashed index if you often do ranged query.

- Avoid text index and index on array field.

- If you know a unique primary key (order_id, transaction_id), use its value in "_id" field instead of creating a separate one. Use ObjectId() or UUID() function to generate these values from your client side instead of writing your own one.

- If you don't know a unique primary key in advanced, you can opt in to use the auto generated "_id" field. Try to not use the auto generated _id and put something make sense into it if possible. You want to avoid {_id: xxx, order_id: xxx} or {_id: xxx, transaction_id: xxx}, etc.

- If you have more than one field you want to enforce unique constraint, create two separate indexes and enforce the unique constraint. This only works for unsharded collection.

MongoDB best practice

- For sharded collection, if you have more than one field that you want to enforce unique constraint on, create a secondary collection to do so. E.g. {user_id: xxx, email: xxx}. You want to make sure both user_id and email are unique. "user_id" is chosen as the shard key already. Here is how it should look like.
  - User: {_id: xxx, email: yyy} "xxx" is the user id. No need for a separate user_id field.
  - Email: {_id: xxx} "xxx" is the email. No need for a separate email field.
  - No need to call createIndex because unique constrain on "_id" is auto enforced.
  - When you try to upsert a new doc, upsert the email collection first, if it passes, upsert the user collection.

MongoDB best practice

- Avoid creating indexes on fields which could be missing if possible.
- Do not worry about choosing the right shard key at the beginning. Decide the shard key after you are very comfortable with your existing indexes.
- Understand how the collection will grow over time. Use a reference data set such as existing user base, merchants, days, etc. Figure out the correlation whether it is linear, exponential, etc.
- Estimate the rough QPS and find a comparable reference of existing collections.
- Query primary when you want 100% data accuracy.

More details for offline reading.

MongoDB basic

Indexes

(Basic) Mongo creates an "_id" index automatically if you don't explicitly specify one. The index key (a k,v tuple) is {"_id": 1}. Here the "k" is the index field name and the "v" is type of the index.

(Basic) "1" means ascending; "-1" means descending; "hashed" means using a hashing function. Sorted index uses binary search O(logN) while hashed index uses hashing O(1). Hashed index offers high speed query performance when query single document based on index equality comparison. If you do ranged queries a lot, do not used hashed index.

(Highly suggested) Compound indexes are created using a list of fields (the order matters). Stick with "1" or "-1" when creating compound indexes. If you find your query using two or three references fields all the time, compound index is your best friend.

MongoDB basic

Indexes (cont'd)

(Basic) Unique constraint. You CANNOT create unique constraint on non index field. You CANNOT create unique constraint on hashed index. You CAN create a hashed index on a single field and a sorted index on the same field and enable unique constraint, but DO NOT do this.

(Not suggested) Text index allows you to run a query with $text. The index could take a lot of memory which harms the DB performance. Use Solr if you have specific need for text based searching.

(Advanced) TTL Index allows you to create an index on a "Date" field with an expiration period. The document will be deleted at the value of "date" field plus the expiration period. If you want to have a collection only stores a rolling window on a time series data set, TTL index is your friend.

MongoDB basic

Indexes (cont'd)

(MISC.) DO NOT create too many indexes. DO NOT create different type of indexes on the same field. DO NOT create index that no query uses. DO NOT fear to try out different index strategy (when collection size is small) because indexes can be destroyed and recreated.

Read here[1] if you want to learn more.

[1] https://docs.mongodb.com/manual/applications/indexes/

MongoDB advanced (replication)

Read preference

primary: default. Only reads from primary. Suggested!

primaryPreferred: Read from primary as long as it is available. Fail over to read from secondary when primary is not available. Use this when you can tolerate error but want your query always succeed.

secondary: Read from secondary only. NOT suggested but you could do it with understanding of the consequences.

secondaryPreferred: Read from secondary as long as it is available. (*It is unlikely we loose all secondaries. So this is almost equivalent to secondary.)

nearest: Read from node who has least network latency. (*Not useful in our case.)

MongoDB advanced (replication)

Replication is designed to ensure **high availability** of the system when there is a node failure and prevent data loss as well as minimize downtime.

Common misunderstanding

- Replication is NOT designed for providing extra capacity (e.g. increased QPS requirement.) This contradicts standard master-slave setup using MySQL/Postgres which relies on read replicas to increase the capacity.
- Why?
    - Secondary member has exact write traffic compared to primary because it tries to keep in sync.
    - Replication is running async and the delay could cause inconsistent state. (*Chunk migration in the sharded cluster also cause inconsistent state)
- If you often run long running queries on secondary because of the fear of clobbering the primary, it is better to consider a data warehouse solution for this type of jobs.
- Production databases are meant for queries executed in milliseconds level. That is insert/update/delete on single document or relative small amount documents using ranged index. Aggregation type of queries based on complex logic or filtering requirement should be avoided. You can solve it in application level or in a data warehouse solution.

MongoDB advanced (sharding)

What is a shard key?

Shard key determines which replica set will contain the actual document. It tells mongos how to route queries. These meta data information are stored in config server.

Shard key must be an existing index or a prefix of an existing compound index.

Shard key can be non unique but this is highly discouraged. (*It causes chunk size overflow). Use compound shard key if necessary.

Shard key should have high cardinality. Avoid sharding on an enum field.

MongoDB advanced (sharding)

Query isolation

Query with shard key allow mongos only forward the query to the shard which contains the shard key. Reduce unnecessary socket connections/queries to the rest of shards. You should always include shard key in your query if possible.

Broadcast (scatter/gather)

Scatter/gather means mongos forward query to all the shards. This happens when you don't include a shard key or when the shard key range you use happen to cover all the shards. (case of hashed sharding strategy)

Is scatter/gather bad? Not necessarily and it cannot be avoided in certain case. Scatter/gather force mongos to connect to all the shards and retrieve data. The query running on each shard could be still fast if you limit the index range and the total execution time is mongos' overhead plus the execution time on the slowest shard.

Should we avoid scatter/gather? Yes by all means if possible. It saves unnecessary socket connection to shards and avoid running no-op queries.

What is the WORST query? Query **w/o shard** key and **w/o index**. This causes a full collection scan!!! Unless there is an absolute reason you need to do so, do not do it. This usually refers to previously mentioned aggregation type of query.

MongoDB advanced (sharding)

Should we shard every collection? (Absolutely NOT!!!)

Why should we shard a collection?

The collection will grow over time, likely no upper bound, e.g. time series data. If the collection could reach hundreds of millions docs, it's a good idea to shard it.

The QPS is extremely high. E.g. a collection with two docs only. The 50k/s query hits doc1 and the other 50k/s query hits doc2. Each machine only supports up to 60k/s QPS. Horizontal scaling to two shards where each shard contain 1 document will solve the problem. What if the machine only supports 40k/s QPS max? Mongo sharding will fail. (Luckily, super high QPS on a single document does not happen in real life.)

Sharding enables capacity (storage/QPS, etc.) improvement.

MongoDB advanced (sharding)

Sharding Strategy

- Hash based sharding
  - Collection with single doc retrieve based on shard key equality check; use w/ hashed index together (shortest query execution time)
  - Ranged query is likely broadcasted to all shards due to the hash function scattering the documents to different shards when they are created.
- Range based sharding
  - Documents with continuous shard key will be near each other when they are stored in chunks. And these chunks are likely to be on the same shard.
  - Range queries with limited range space should only hit one shard or two at most.
  - Will this cause certain shard overheated (high connection/QPS)? Yes, when collections shares the same primary shard ("main" shard in our case). This is not the problem you need to worry about.
- Range based sharding is more common in general.

MongoDB orphan

What is orphan?

Orphan is a byproduct of sharding in mongo. Orphans are documents which supposed to be deleted after chunk migration. They are never updated after they are generated. They are hard to track down and they cause many confusions.

How is orphan generated?

MongoDB orphan

When does orphan occur?

Orphans are usually created during the chunk migration. If the delete ops get interrupted on the source shard, orphans are created in the source shard. If the copy ops get interrupted on the target shard, orphans are created on the target shard.

Can we not migrate chunks? Of course NOT. We have to migrate chunks to balance the data distribution on the mongo shards. This is for the sake of both storage as well as QPS.

When will you suffer from orphan problem?

Query the secondary on a set of shard keys where mongos forward the query to more than one shard. (Bad but fixable after enable versioning).

Query without shard key. (Very bad unless you are ok to tolerate some level of errors.)

MongoDB orphan

How to solve orphan problem?

Query the primary. MongoDB has a built in SHARD_FILTER stage being executed to remove any document that does not belong to the current shard.

*Enable versioning. This only work when you include the shard key. The shard key ensure that the result set contain orphans along with their genuine copy of docs. Opt in the "orphan_filter" will ensure find() and find_iter() remove orphan automatically based on version value. This requires every doc to be versioned in existing collection.

Eliminate the root cause of the orphan. Run explicit delete on the source shard after a successful chunk migration. Run explicit delete on the target shard after a failed chunk migration. (*We need to test and ensure that this will not create too much extra load on our primary shard. Likely not.)