Sharding in MongoDB 4.2
#what_is_new

Antonios Giannopoulos
DBA @ ObjectRocket by Rackspace
Connect:linkedin.com/in/antonis/ Follow:@iamantonios
Introduction

Antonios Giannopoulos

Database troubleshooter aka troublemaker
@ObjectRocket

Troubleshoot: MongoDB, CockroachDB & Postgres

Troublemaking: All the things

Give your developers their time back.

We’ve got your databases covered.

MongoDB  Elasticsearch  Redis

PostgreSQL  CockroachDB  Hadoop
Overview

- Change the shard key value
- Distributed transactions
- Split Chunks
- Balancer
- Connection Pool
Before we start, presentation examples are based on the following:

**Use case: A virtual Bank**

**Customers**

```javascript
db.bank.ensureIndex({region:1,iban:1});
db.bank.insert({_id:1, name:"Antonios",amount:100, region:"Europe",country:"GR", iban:"GR001"});
db.bank.insert({_id:2, name:"Alex",amount:100, region:"Europe",country:"UK", iban:"UK001"});
db.bank.insert({_id:3, name:"Jon",amount:100, region:"Pacific",country:"AU", iban:"AU001"});
db.bank.insert({_id:4, name:"Jason",amount:100, region:"America", country:"US", iban:"US001"});
```

**Sharded on** `{region, iban}` – iban is unique

```javascript
sh.shardCollection("percona.bank",{region:1,iban:1});
```

**Two Shards rs01, rs02 – With three Zones**

```javascript
sh.addShardTag("rs02", "Europe");
sh.addShardTag("rs01","America");
sh.addShardTag("rs02", "RestoftheWorld");
sh.addTagRange("percona.bank", { region: "Europe" }, { region: "Europe1" }, "Europe");
sh.addTagRange("percona.bank", { region: "America" }, { region: "America1" }, "America");
sh.addTagRange("percona.bank", { region: "Pacific" }, { region: "Pacific1" }, "RestoftheWorld");
```
A good shard key must...

Be immutable...

Let’s examine this throughout an example:

```javascript
mongos> db.bank.findOne({_id:1})
{
    "_id": 1,
    "name": "Antonios",
    "amount": 100,
    "region": "Europe",
    "country": "GR",
    "iban": "GR001"
}
```

```javascript
sh.shardCollection("percona.bank", {region:1,iban:1});
```
sh.addShardTag("rs01","America");
sh.addShardTag("rs02", "Europe");
sh.addShardTag("rs02", "RestoftheWorld");

Customers from America go to rs01
Customers from Europe go to rs02

Move a customer from America to Europe requires document relocation
A good shard key must...

Be immutable...

```javascript
mongos> db.bank.update( { "region":"Europe", "iban":"GR001" }, {$set: { "region":"Europe", "iban":"UK001", country:"UK" }});
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 0,
  "nModified" : 0,
  "writeError" : {
    "code" : 66,
    "errmsg" : "Performing an update on the path 'iban' would modify the immutable field 'iban'"
  }
})

mongos> db.bank.find({ _id:1 })
{ "_id" : 1, "name" : "Antonios", "amount" : 100, "region" : "Europe", "country" : "GR", "iban" : "GR001" }
WriteResult({ "nInserted" : 1 })
mongos> db.bank.remove({ _id:1})
WriteResult({ "nRemoved" : 1 })
mongos> db.bank.find({name:"Antonios"})
{ "_id" : 5, "name" : "Antonios", "amount" : 100, "region" : "Europe", "country" : "UK", "iban" : "UK002" }
```
A good shard key is mutable...

```javascript
mongos> db.adminCommand( { setFeatureCompatibilityVersion: "4.2" } )
{ "ok" : 1,
  "operationTime" : Timestamp(1567596891, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1567596891, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

mongos> db.bank.update( { "region":"Europe",iban:"GR001" },{ $set: { "region": "Europe", iban: "UK001", country: "UK" } });
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

mongos> db.bank.find({name: "Antonios"})
{ "_id" : 1, "name" : "Antonios", "amount" : 100, "region" : "Europe", "country" : "UK", "iban" : "UK001" }
```
shard key is mutable, unless...

Unless the shard key field is the immutable _id field

```javascript
mongos> db.bankII.update( { _id: 1 }, {$set:{_id:100,"region":"Europe", iban:"UK001", country:"UK"}});
WriteResult({
    "nMatched" : 0,
    "nUpserted" : 0,
    "nModified" : 0,
    "writeError" : {
        "code" : 66,
        "errmsg" : "Performing an update on the path '_id' would modify the immutable field '_id'"
    }
});
```

You miss the full shard key in the query

```javascript
mongos> db.bank.update({"region" : "Europe", _id:1},{$set:{"region" : "America", iban:"US001"}})
WriteResult({
    "nMatched" : 0,
    "nUpserted" : 0,
    "nModified" : 0,
    "writeError" : {
        "code" : 31025,
        "errmsg" : "Shard key update is not allowed without specifying the full shard key in the query"
    }
});
```
If the shard key modification does not result in moving the document to another shard, you can specify multiple shard key modification in the bulk operation.

```javascript
mongos> session = db.getMongo().startSession({retryWrites:true});
session { "id": UUID("80953c9f-a016-4525-a204-3166bc71173c") }
mongos> coll = session.getDatabase("percona").bank;
percona.bank
mongos> session.startTransaction();

mongos> coll1.bulkWrite([ ... { updateOne : { ... "filter" : { "region":"Europe",iban:"GR001" }, ... "update" : { $set:{"region":"Europe", iban:"UK002", country:"UK"} } ... } }, ... { updateOne : { ... "filter" : { "region":"Europe",iban:"UK001" }, ... "update" : { $set : { "region":"Europe", iban:"RO001", country:"RO"} } ... } } ... ]);

{ "acknowledged" : true,
  "deletedCount" : 0,
  "insertedCount" : 0,
  "matchedCount" : 2,
  "upsertedCount" : 0,
  "insertedIds" : [],
  "upsertedIds" : [],
}

mongos> session.commitTransaction();
```
the shard key modification does not result in moving the document to another shard, you can specify multiple shard key modification in the bulk operation.
Change the shard key... (?)

You can’t change the fields of the shard key 😞

…but you can re-purpose it 😊

For example, shard key {client_id:1}

**Bucketing:** {client_id:"000"} to {client_id:"000-2019"}

**Locality:** {client_id: “US-000”} , {client_id:"UK-000”}

**Completely repurpose:** A field name is what the application think it is!!!
Distributed Transactions

- Implementation
- Examples
- Considerations
Distributed Transactions

In MongoDB operations on a single document are atomic.

**MongoDB 4.0** supports multi-document transactions on *replica sets* (WiredTiger only)

**MongoDB 4.2** supports distributed transactions, which adds support for multi-document transactions on *sharded clusters*

Change the value of the shard key is nothing more than a *distributed transaction*

Transactions on any distributed system are challenging (anyone disagrees?)

One of the biggest challenges is the “**All or nothing**”
How Transactions work...

If the transaction touches **only one shard**, behavior is similar to a **replica-set** transaction

```
mongos> session = db.getMongo().startSession();
session { "id" : UUID("1c965676-4143-4266-a305-21c02bce9373"), }
mongos> coll1 = session.getDatabase("percona").bank;
percona.bank
mongos> session.startTransaction();
mongos> coll1.update( { "region": "Europe", "iban": "GR001" }, {$inc: {amount: 50}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
mongos> coll1.update( { "region": "Europe", "iban": "UK001" }, {$inc: {amount: -50}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
mongos> session.commitTransaction()
```
One shard involved... continued

```json
"o" : {
    "applyOps" : [
        {
            "op" : "u",
            "ns" : "percona.bank",
            "ui" : UUID("fcc49753-028c-48e8-a113-ac1656d626bb"),
            "o" : {
                "$v" : 1,
                "$set" : {
                    "amount" : 150
                }
            },
            "o2" : {
                "region" : "Europe",
                "iban" : "GR001",
                "_id" : 1
            }
        },
        {
            "op" : "u",
            "ns" : "percona.bank",
            "ui" : UUID("fcc49753-028c-48e8-a113-ac1656d626bb"),
            "o" : {
                "$v" : 1,
                "$set" : {
                    "amount" : 50
                }
            },
            "o2" : {
                "region" : "Europe",
                "iban" : "UK001",
                "_id" : 2
            }
        }
    ]
}
```
How Transactions work...

If the trx touches more than one shard: behavior is similar to a two phase commit

On every distributed transaction a shard acts as Coordinator

A distributed transaction has two states: the Prepare and the Commit state

- Prepare state guarantees the ability to Commit
- All shards must prepare a transaction (w:majority) before Commit
- If any shard fails to Prepare, then no shard will Commit
- Coordinator is responsible for the ack of Prepare and Commit
- Prepared Transactions held in memory, and Replication makes them durable

Confused… Let see an example
Zones: Europe and America are on different shards
How Transactions work...

1. update({EU}, {$inc: {amount: 50}})
   
   Shard becomes coordinator (C)

2. update({US}, {$inc: {amount: -50}})
   
   Both (1) & (2) are now in cache

   Coordinator say prepare (Succeeds)

   (1) & (2) are written in the oplog

   Coordinator say commit (Succeeds)

   (1) & (2) are written in the storage and become visible
The first statements picks a coordinator (first update in our case)

```
{
    "ts": Timestamp(1567866408, 1),
    "t": NumberLong(4),
    "h": NumberLong(0),
    "v": 2,
    "op": "i",
    "ns": "config.transaction_coordinators",
    "uid": UUID("05b05426-8a27-4811-9e31-35a73fabf8dd"),
    "wall": ISODate("2019-09-07T14:26:46.281Z"),
    "o": {
        "_id": {
            "lsid": {
                "id": UUID("36d196dc-19b8-41c6-810d-fdd7ac68b3f6"),
                "uid": BinData(0,"47DEpJ8HSa+/TImW+5JCeuQeRkm5NMP.JWZG3hSU Fu=")
            },
            "txnNumber": NumberLong(0)
        },
        "participants": [{
            "rs01",
            "rs02"
        }]
    }
}
```
2+ shards involved... continued

Coordinator says:: Lets prepare Olog entries from rs01 and rs02

```
{ "ts" : Timestamp(1567866406, 2), "t" : NumberLong(5), "h" : NumberLong(0), "y" : 2, "op" : "c", "ns" : "admin.$cmd", "wall" : ISODate("2019-09-07T14:26:46.298Z"), "lsid" : { "id" : UUID("36d196dc-19b8-41c6-810d-fdd7ac08b3f6"), "uid" : BinData(0,"47DEQpj8HBSa/7IMw5JceuQeRkm5NMpJWZG3hSU FU="), "txnNumber" : NumberLong(0), "prevOpTime" : { "ts" : Timestamp(0, 0), "t" : NumberLong(-1) }, "o" : { "applyOps" : [ { "op" : "u", "ns" : "percona.bank", "ui" : UUID("ffe49763-878c-48e8-a113-ac1656d626bb"), "o" : { "$v" : 1, "$set" : { "amount" : 50 } }, "o2" : { "region" : "America", "iban" : "US001", "_id" : 4 } ], ["prepare" : true] } }
```

```
{ "ts" : Timestamp(1567866406, 2), "t" : NumberLong(4), "h" : NumberLong(0), "y" : 2, "op" : "c", "ns" : "admin.$cmd", "wall" : ISODate("2019-09-07T14:26:46.291Z"), "lsid" : { "id" : UUID("36d196dc-19b8-41c6-810d-fdd7ac08b3f6"), "uid" : BinData(0,"47DEQpj8HBSa/7IMw5JceuQeRkm5NMpJWZG3hSU FU="), "txnNumber" : NumberLong(0), "prevOpTime" : { "ts" : Timestamp(0, 0), "t" : NumberLong(-1) }, "o" : { "applyOps" : [ { "op" : "u", "ns" : "percona.bank", "ui" : UUID("ffe49763-878c-48e8-a113-ac1656d626bb"), "o" : { "$v" : 1, "$set" : { "amount" : 200 } }, "o2" : { "region" : "Europe", "iban" : "GR001", "_id" : 1 } ], ["prepare" : true] } }
```
Coordinator says: Lets commit (Coordinator’s oplog)

```
"t" : NumberLong(4),
"h" : NumberLong(0),
"v" : 2,
"op" : "u",
"ns" : "config.transaction coordinators",
"uid" : UUID("d5b65428-8a27-48ff-9e51-33a73fabf6dd"),
"o2" : {
    "_id" : {
        "lsid" : {
            "id" : UUID("36d196dc-19b8-41c6-810d-fdd7ac08b3f6"),
            "uid" : BinData(0,"47DE0pj9HBSa+/TlmW+3JCe1QeRkm5NmpjWZG3hSuFU=")
        },
        "txnNumber" : NumberLong(0)
    }
},
"wall" : ISODate("2019-09-07T14:26:46.307Z"),
"o" : {
    "_id" : {
        "lsid" : {
            "id" : UUID("36d196dc-19b8-41c6-810d-fdd7ac08b3f6"),
            "uid" : BinData(0,"47DE0pj9HBSa+/TlmW+3JCe1QeRkm5NmpjWZG3hSuFU=")
        },
        "txnNumber" : NumberLong(0)
    },
    "participants" : [
        "rs01",
        "rs02"
    ],
    "decision" : {
        "decision" : "commit",
        "commitTimestamp" : Timestamp(1567866406, 2)
    }
}
```
2+ shards involved... continued

Coordinator says: Lets commit,
Olog entries from rs01 and rs02

```
{ "ts" : Timestamp(1567866406, 4), "t" : NumberLong(5), "h" : NumberLong(0), "v" : 2, "op" : "c", "ns" : "admin.$cmd", "wall" : ISODate("2019-09-07T14:26:46.313Z"), "lsid" : { "id" : UUID("36d196dc-19b8-41c6-810d-fdd7ac08b3f6"), "uid" : BinData(0,"47DEqip8H8Sa+/T1mW+SJC0eRkm5N MPsJWZG3hSuF= "), "txnNumber" : NumberLong(0), "prevOpTime" : { "ts" : Timestamp(1567866406, 2), "t" : NumberLong(5) }, "o" : { "commitTransaction" : 1 } }, "commitTimestamp" : Timestamp(1567866406, 2) }
```

```
{ "ts" : Timestamp(1567866406, 4), "t" : NumberLong(4), "h" : NumberLong(0), "v" : 2, "op" : "c", "ns" : "admin.$cmd", "wall" : ISODate("2019-09-07T14:26:46.312Z"), "lsid" : { "id" : UUID("36d196dc-19b8-41c6-810d-fdd7ac08b3f6"), "uid" : BinData(0,"47DEqip8H8Sa+/T1mW+SJC0eRkm5N MPsJWZG3hSuF= "), "txnNumber" : NumberLong(0), "prevOpTime" : { "ts" : Timestamp(1567866406, 2), "t" : NumberLong(4) }, "o" : { "commitTransaction" : 1 } }, "commitTimestamp" : Timestamp(1567866406, 2) }
```
Transactions & the oplog...

The 16MB limit removed in 4.2

Transactions break into a chain of events

`prevOptime`: connects the chain

`partialTnx`: create the chain

*The oplog entries are truncated*
Considerations

db.adminCommand( { setFeatureCompatibilityVersion: "4.2" } )

You will need the latest drivers

`writeConcernMajorityJournalDefault` must be set to true

Set `maxTimeMS` on commit, else it would default `transactionLifetimeLimitSeconds`

Chunk migrations:
A chunk migration waits for transaction lock on chunks documents
If a chunk migration is ongoing transaction may fail

`db.serverStatus().shardingStatistics.countDonorMoveChunkLockTimeout`
Considerations ... continued

Multi shard transactions will fail, if an **arbiter** is in place:

```javascript
mongodb> session = db.getMongo().startSession();
session { "id" : UUID("b8dbf7e4-c6ed-49ae-a939-468e4623fba9") }
mongodb> coll1 = session.getDatabase("percona").bank;
percona.ban
mongodb> session.startTransaction()
mongodb> coll1.update( { "region":"Europe",iban:"UK001" },{$inc:{amount:-50}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
mongodb> coll1.update( { "region":"America",iban:"US001" },{$inc:{amount:-50}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
mongodb> session.commitTransaction()
   "ok" : 0,
   "errmsg" : "from shard rs02 :: caused by :: 'prepareTransaction' is not supported for replica sets with arbiters",
   "code" : 148,
   "codeName" : "ReadConcernMajorityNotEnabled",
   "operationTime" : Timestamp(1568045424, 4),
   "$clusterTime" : {
      "clusterTime" : Timestamp(1568045424, 4),
      "signature" : {
         "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
         "keyId" : NumberLong(0)
      }
   },
   "recoveryToken" : {
      "recoveryShardId" : "rs02"
   }
}
```
Considerations ... continued

There are restrictions on certain operators
- Same restrictions as 4.0 with the addition,
- You cannot write to **capped** collections.
- You cannot specify **killCursors** as the first operation in a transaction.

Outside Reads During Commit
- Read concern **snapshot** wait for all writes of a transaction to be visible.
- Other read concerns (**local** or **majority**) do not wait for all writes of a transaction to be visible but instead read the before-transaction version of the documents available.

Reconsider backup strategy (**mongodump**)
Considerations… Failovers

Elections:
- Majority commit or Failed to prepare

Startup Recovery:
- Consistent point in time -> noted on prepare trx table -> Recover -> Check if any prepared trx needs to be applied
- Prepare transactions are immutable
- Conflicts handled by the Primary
- Reads are not allowed while recovering

Initial sync – same as startup recovery

Rollback:
- Rollback to stable timestamp WT-3387
- Move to Common point with prepare trx table
- After Common point act as Primary
Performance

Single shard transactions should have the same cost as replica-set transactions

Multi shard transactions are more expensive compared to ReplicaSet ones’

Transactions saved in cache – more RAM may needed

Remote shards may slow down due to network latency

Don’t give up on the MongoDB data modeling

Use transactions whenever is absolutely necessary

Try to hit as less shards as possible

Read many, Write one is optimized
Miscellaneous Changes

- Chunk Split
- Balancer
- Connection Pool
Responsible for AutoSplit...

Prior to 4.2: Mongos
- Each mongos keeps its own statistics
- May lead to jumbo chunks
- May lead into too many split requests
- Especially with high number of mongos

In 4.2: The responsibility passed to Shards
SERVER-9287
Balancer

The `balancerStart` command and the mongo shell helper methods `sh.startBalancer()` and `sh.setBalancerState(true)` also enable **auto-splitting** for the sharded cluster.

To disable auto-splitting when the balancer is enabled, you can use `sh.disableAutoSplit()`.

The `balancerStop` command and the mongo shell helper methods `sh.stopBalancer()` and `sh.setBalancerState(false)` also disable **auto-splitting** for the sharded cluster.

To enable auto-splitting when the balancer is disabled, you can use `sh.enableAutoSplit()`.

The mongo methods `sh.enableBalancing(namespace)` & `sh.disableBalancing(namespace)` have no affect on the **auto-splitting**.
Mongos Connection Pool

`ShardingTaskExecutorPoolReplicaSetMatching`: determines the minimum size limit of the mongos instance’s connection pools to the sharded cluster’s replica set secondaries.

```
$ db.adminCommand( { setParameter: 1, ShardingTaskExecutorPoolReplicaSetMatching: <value> } )
```

, where `<value>`:

- **matchPrimaryNode**: the minimum size limit of each secondary of that replica set is equal to the size of its connection pool to the primary.
- **matchBusiestNode**: the minimum size limit is the largest among the active connections counts to the primary and each secondary members.
- **Disabled**: the minimum number of connections in the mongos instance’s connection pool to each secondary is equal to the `ShardingTaskExecutorPoolMinSize`.
Recap & Takeways

- The shard key value is mutable
- Transactions are supported on sharded clusters
- On a single shard same performance as Replset transactions
- On multiple shards there is a performance overhead
- Transaction 16MiB limit lifted
- Split is now running on the shards
Questions?
Rate My Session

New Indexing and Aggregation Pipeline Capabilities in MongoDB 4.2

I had problems sleeping but not anymore

New Indexing and Aggregation Pipeline Capabilities in MongoDB 4.2

I had problems sleeping, but I took a quick nap

New Indexing and Aggregation Pipeline Capabilities in MongoDB 4.2

I still have problems sleeping, but count bugs in 4.2 helps me sleep
We’re Hiring!
Looking to join a dynamic & innovative team?

https://www.objectrocket.com/careers/
Thank you!

Address:
9001 N Interstate Hwy 35 #150, Austin, TX 78753

Support:
US Toll free: 1-855-722-8165
UK Toll free +448081686840
support@objectrocket.com

Sales:
1-888-440-3242
sales@objectrocket.com

www.objectrocket.com