

MongoDB 4.0 Features - Transactions & More

Alexander Rubin
Principal Architect



About me

My name is Alexander Rubin

- Working with MySQL for over 12 years
 - Started at MySQL AB, then Sun Microsystems,
 - then Oracle (MySQL Consulting)
 - Joined Percona 5 years ago
- Working with MongoDB for 3 years

MongoDB 4.0 new features

- Will only focus on MongoDB 4.0 server features
 - *will not talk about cloud offering*
- Biggest feature is **transactions!**
- Other server features include:
 - *Non blocking reads*
 - *Improved Sharding*
 - *and more*

MongoDB 4.0 Transactions

MongoDB transaction explained

- Usually transactions are explained with bank transaction
 - ... *Financial applications are less common with MongoDB*
 - ... *but gaming / and mobile application are!*



Pokemon Go: a case for transactions



A Complex Exchange Transaction:

- Remove pokemon from player **a**
- Add that pokemon to player **b**
- Remove “payment” (stardust item) from **a** and **b**
- Add “bonus” (pokemon candy) to **a** and **b**
- (internal) may need to record transaction info:
 - *date and time*
 - *etc*

Pokemon Go: a case for transactions



Before transactions: MongoDB 3.x

- There are tons of ways to simulate transactions
- Common approach: simulate redo log:
 - Use *exchange_transaction_log* collection
 - Add everything that is happening inside transaction
 - Eventually (async or sync) make the actual changes
 - Catch any exceptions / cleanup on error

Pokemon Go: MongoDB 4.0



// this is basic design of what Pokemon exchange can look like

```
session = db.getMongo().startSession()
session.startTransaction()
session.getDatabase("p").player_items.update(
    {player_id : 123}, { $inc: { stardust: -1 } } );
session.getDatabase("p").player_items.update(
    {player_id : 321}, { $inc: { stardust: -1 } } );
...
session.getDatabase("p").player_pokemons.update(
    {player_id : 321},
    $push: {pokemon_id: 987});
...
session.commitTransaction()
```

<https://support.pokemongo.nianticlabs.com/hc/en-us/articles/360001518407-Trading-Pokémon>

Transactions In MongoDB 4.0

```
session = db.getMongo().startSession()
```

- Session is a concept introduced in MongoDB 3.6.
- Session is a "context"
 - mainly used for multi-document transactions.
- Session have lsid, txnNumber, stmtIds that identifies session, transaction and operation (does not need to be set)

Current transaction implementation (4.0)

- **Requires WiredTiger**
- In MongoDB < 4.0, an operation on a single document is atomic, but not on multiple documents (or multiple collections)
- Starting with 4.0, MongoDB provides the ability to perform multi-document transactions against **replica sets**
- When a transaction commits, all data changes made in the transaction are saved.
- If any operation in the transaction fails, the transaction aborts
 - all data changes made in the transaction are discarded without ever becoming visible.
- Until a transaction commits, no write operations in the transaction are visible outside the transaction (the transaction is still in memory).
 - If there is no commit the transaction will be aborted (not rolled back)

Transactions: ACID

Atomicity

- "all or nothing" guarantee for multi-document transaction operations
- Data changes are only made visible outside the transaction **if it is successful**.
- If a transaction **fails**, all of the data changes from the transaction is **discarded**.

Transactions: ACID

Consistency

- Handled by MongoDB
 - example: trying to change a value that fails schema validation, will cause inconsistent data
 - (permitted transactions should not corrupt data)

Transactions: ACID

Isolation

- Snapshot isolation level
 - ... creates a WiredTiger snapshot at the beginning of the transaction
 - ... uses this snapshot to provide transactional reads throughout the transaction.

Transactions: ACID

Durability

- When a transactions use WriteConcern {j: true} (default), MongoDB will guarantee that it is returned after the transaction log is committed.
- Even if a crash occurs, MongoDB can recover according to the transaction log.
- If the {j: true} level is not specified, even after the transaction is successfully committed, the transaction may be rolled back (in case of crash recovery,)

Transactions and ReplicaSet

Transaction and Replica

- In the replica set configuration, an oplog will be recorded on commit
 - ... including all the operations in the transaction.
- The slave node pulls the oplog and replays the transaction operations locally

(the document size limit is 16 MB, so whole transaction can not exceed 16MB)

Transaction and WiredTiger

Unified Transaction Timing

- WiredTiger has supported transactions for a long time
 - (guarantee the modification atomicity of data, index, and oplog)
- The problem was with timing:
 - MongoDB used the **oplog timestamps** to identify the global order
 - WiredTiger used the **internal transaction IDs** to identify the global order
- MongoDB version 4.0 / WiredTiger 3.0 introduced transaction timestamps
 - MongoDB can now explicitly assign a commit timestamp to the WiredTiger transaction (read "as of" a timestamp)
 - When the oplog is replayed, the read on the slave node will no longer conflict with the replayed oplog, and the read request will not be blocked by replaying the oplog.

Marathon to transactions: the homestretch

MongoDB 3.0	MongoDB 3.2	MongoDB 3.4	MongoDB 3.6	MongoDB 4.0 Single Replica Set Transactions	MongoDB 4.2 Sharded Transactions
New Storage engine (WiredTiger)	Enhanced replication protocol: stricter consistency & durability	Shard membership awareness	Consistent secondary reads in sharded clusters	Storage support for prepared transactions	Transaction - compatible chunk migration
	WiredTiger default storage engine		Logical sessions	Make catalog timestamp-aware	More extensive Wired Tiger repair
	Config server manageability improvements		Retryable writes	Replica set point-in-time reads	Transaction manager
	Read concern "majority"		Causal Consistency	Recoverable rollback via WT checkpoints	Global point-in-time reads
			Cluster-wide logical clock	Recover to a timestamp	Oplog applier prepare support for transactions
			Storage API to changes to use timestamps	Sharded catalog improvements	
			Collection catalog versioning		
			Make collection drops two phase		
			UUIDs in sharding		
			Fast in-place updates to large documents in WT		

- Done
- In Progress
- Sharded Transaction Feature

Transaction isolation level in MongoDB 4.0

Snapshot Isolation

- MongoDB 4.0 implements snapshot isolation for the transactions
- The pending uncommitted changes are only visible inside the session context
 - (the session which has started the transaction)
 - are not visible outside.

Snapshot Isolation: Example

- Connection 1:

```
foo:PRIMARY> session = db.getMongo().startSession()
session { "id" : UUID("bdd82af7-ab9d-4cd3-9238-f08ee928f31e") }
foo:PRIMARY> session.startTransaction()
foo:PRIMARY> session.getDatabase("percona").test.insert({today : new Date()})
WriteResult({ "nInserted" : 1 })
```

- Connection 2:

```
foo:PRIMARY> session = db.getMongo().startSession()
session { "id" : UUID("eb628bfd-425e-450c-a51b-733435474eaa") }
foo:PRIMARY> session.startTransaction()
foo:PRIMARY> session.getDatabase("percona").test.find()
// nothing
```

Snapshot Isolation: Example

- Connection 1: commit
`foo:PRIMARY> session.commitTransaction()`
- Connection 2: inside of the **original** session
`foo:PRIMARY> session.getDatabase("percona").test.find()`
`foo:PRIMARY> // nothing`
- Connection 2: **outside** of the session
`foo:PRIMARY> db.test.find()`
`{ "_id" : ObjectId("5b21361252bbe6e5b9a70a4e"), "today" : ISODate("2018-06-13T15:19:46.645Z") }`
`{ "_id" : ObjectId("5b21361252bbe6e5b9a70a4f"), "some_value" : "abc" }`

MongoDB transactions: conflict

How does MongoDB handles conflicts:

- Let's say we are updating the same row. First we create a record, trx, in the collection:

```
use percona
```

```
db.test.insert({trx : 0})
```

- Then we create session1 and update trx to change from 0 to 1:

```
foo:PRIMARY> session = db.getMongo().startSession()
```

```
session { "id" : UUID("0b7b8ce0-919a-401a-af01-69fe90876301") }
```

```
foo:PRIMARY> session.startTransaction()
```

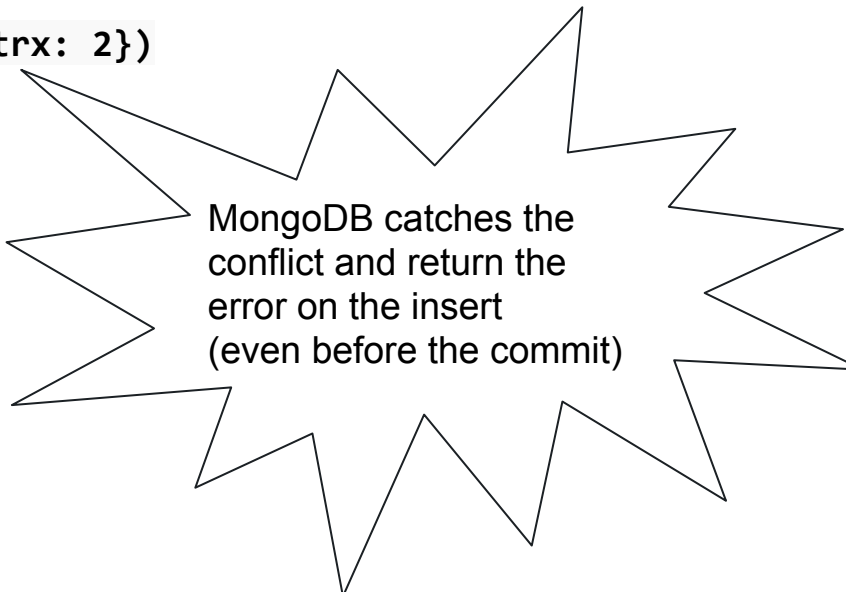
```
foo:PRIMARY> session.getDatabase("percona").test.update({trx : 0}, {trx: 1})
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

MongoDB transactions: conflict

- Then (before committing) create another session which will try to change from 0 to 2:

```
foo:PRIMARY> session = db.getMongo().startSession()
session { "id" : UUID("b312c662-247c-47c5-b0c9-23d77f4e9f6d") }
foo:PRIMARY> session.startTransaction()
foo:PRIMARY> session.getDatabase("percona").test.update({trx : 0}, {trx: 2})
WriteCommandError({
  "errorLabels" : [
    "TransientTransactionError"
  ],
  "operationTime" : Timestamp(1529675754, 1),
  "ok" : 0,
  "errmsg" : "WriteConflict",
  "code" : 112,
  "codeName" : "WriteConflict",
  "$clusterTime" : {
    "clusterTime" : Timestamp(1529675754, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
})
```



MongoDB catches the conflict and return the error on the insert (even before the commit)

Write Concern and Transactions

MongoDB is a distributed database- be aware of the different options for consistency.

Write concern describes the level of acknowledgement requested from MongoDB for write operations

- For multi-document transactions, you set the write concern at the **transaction level**, not at the individual operation level.

```
Session.startTransaction({ writeConcern: { w: <level> } })  
                        // w: 1, majority
```

If you commit using "w: 1" write concern, your transaction can be rolled back during the failover process.

Read Concern and Transactions

The readConcern option allows you to control the consistency and isolation properties of the data read from replica sets and replica set shards.

- For multi-document transactions, you set the read concern at the **transaction level**, not at the individual operation level.
- If unspecified at the transaction start, transactions use the session-level read concern or, if that is unset, the client-level read concern.

```
Session.startTransaction({ readConcern: { level: <level> } })
```

Read Concern “Snapshot” and Transactions

- Read concern "snapshot" is only available for multi-document transactions.
- Multi-document transactions support read concern "snapshot" as well as "local", and "majority".

```
Session.startTransaction({ readConcern: { level: "snapshot" } })
```

MongoDB 4.0: Other features and improvements

Not only transactions

MongoDB 4.0: Not only transactions

Non-Blocking Secondary Reads

- **MongoDB previously blocked secondary reads** while oplog entries were applied.
 - when the replication threads were writing to the database the readers must wait
- MongoDB 4.0 adds the ability to read from secondaries while replication is simultaneously processing writes.

MongoDB 4.0: Not only transactions

Extensions to Change Streams

- Change streams introduced in version 3.6 helps applications to access real-time data changes (similar to tail the collection but better as it is replication aware).
- In 4.0 Change Streams can be configured to track changes across an entire database or whole cluster. Also, it will return a cluster time associated with an event (to provide an associated wall clock time for the event)

MongoDB 4.0: Not only transactions

Data Type Conversions

- A new expression \$convert has been added to the aggregation framework
<https://docs.mongodb.com/manual/reference/operator/aggregation/convert/>
 - Helps for ETL workloads
 - Also allow the MongoDB BI Connector to push down work to MongoDB Server (and avoid sending data over the wire)

MongoDB 4.0: Not only transactions

SHA-2 Authentication

- With MongoDB 4.0, authentication has been updated to the latest SHA-2 family (SHA-256), providing a stronger alternative to SHA-1

MongoDB 4.0: Not only transactions

Improved Sharding

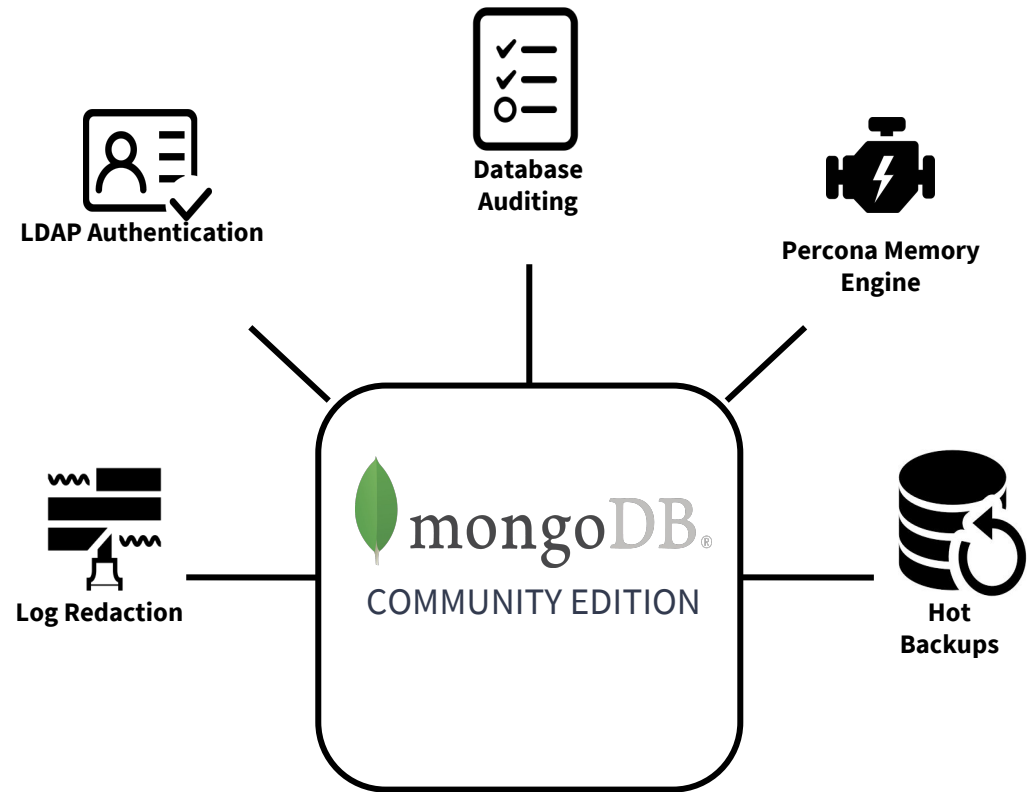
- **Sharded migrations are now up to 40% faster** helping for better distribution of the data.
- Operators can now list and kill queries running in a sharded cluster directly on a mongos node.
- Slow Query Logging on mongos: ability to enable profile on mongos

Percona Server for MongoDB



PERCONA
Server for MongoDB

Free and open source



Links

- [MongoDB Transaction Documentation](#)
- MongoDB 4.0 and Transactions: [interviews with MongoDB engineers](#)
- [MongoDB Transactions: Your Very First Transaction with MongoDB 4.0](#)
- [MongoDB Server 4.0: What's New](#)
- [MongoDB World 2018 keynote](#)



PERCONA
LIVE EUROPE
FRANKFURT

Full Agenda is Live!

Percona Live Europe

Connect. Accelerate. Innovate.

Join the open source community in Frankfurt, Germany, to learn about core topics in MySQL, MongoDB, PostgreSQL and other open source databases.

Reserve Your Seat

Frankfurt 5-7 November 2018

[Buy Your Tickets](#) >

Thank you!



Alexander Rubin

<https://www.linkedin.com/in/alexanderrubin>



**Champions OF Unbiased
Open Source Database Solutions**