PERCONA

TRAINING

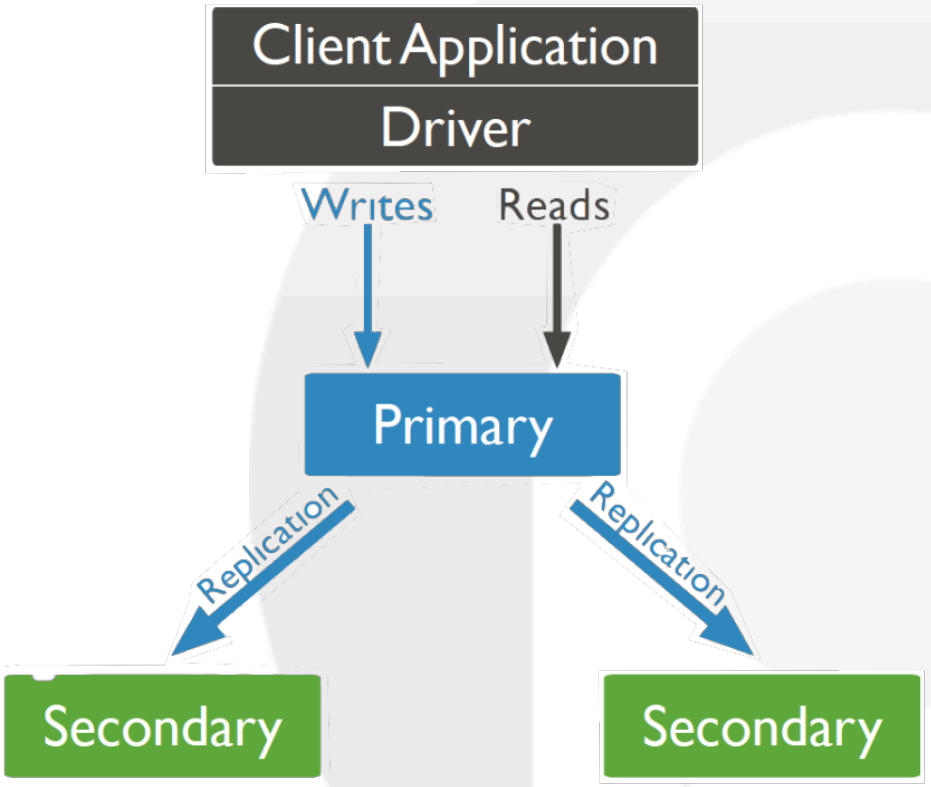
Replication

http://www.percona.com/training/

Replication

HOW IT WORKS

Replication Overview



The `local` database

- Stores data used in the replication process, and other instance-specific data
- Collections in the local database are not replicated
- Contains the Operations Log (Oplog)

Operations Log (Oplog)

- The operations log (oplog), is a special capped collection that is the basis for replication
- The oplog maintains one entry for each document affected by every write operation
- Secondaries copy operations from the oplog of their sync source
- Each operation in the oplog is idempotent

Operations Log (Oplog)

- Secondaries collect oplog entries in batches grouped by document id
- MongoDB applies batches in parallel with different threads
- Before MongoDB 4.0, read operations on secondaries would be blocked until any ongoing replication completes

Operations Log (Oplog)

```
db.foo.remove({ age : 30 })
```

This will be represented in the oplog:

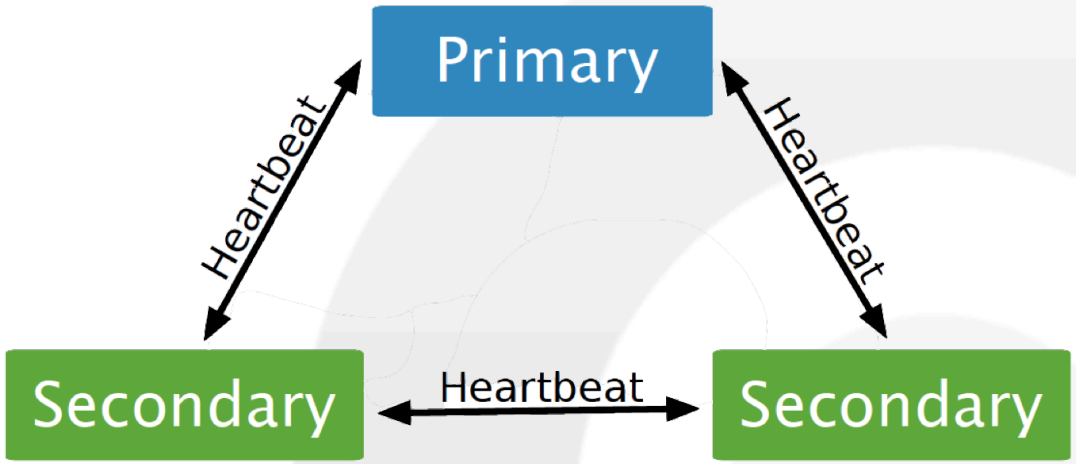
```
{ "ts" : Timestamp(1407159845, 5), "h" : NumberLong("-7046124876919269 08"),
"v" : 2, "op" : "d", "ns" : "bar.foo", "b" : true, "o" : { "_id" : 65 } }
{ "ts" : Timestamp(1407159845, 1), "h" : NumberLong("6014126345225019794"),
"v" : 2, "op" : "d", "ns" : "bar.foo", "b" : true, "o" : { "_id" : 333 } }
{ "ts" : Timestamp(1407159845, 4), "h" : NumberLong("8178791764238465439"),
"v" : 2, "op" : "d", "ns" : "bar.foo", "b" : true, "o" : { "_id" : 447 } }
{ "ts" : Timestamp(1407159845, 3), "h" : NumberLong("-1707391001705528381"),
"v" : 2, "op" : "d", "ns" : "bar.foo", "b" : true, "o" : { "_id" : 1033 } }
{ "ts" : Timestamp(1407159845, 2), "h" : NumberLong("-6814297392442406598"),
"v" : 2, "op" : "d", "ns" : "bar.foo", "b" : true, "o" : { "_id" : 9971 } }
```

Viewing oplog size

```
rs.printReplicationInfo()

configured oplog size:   10.10546875MB
log length start to end: 94400 (26.22hrs)
oplog first event time:  Mon Mar 19 2012 13:50:38 GMT-0400 (EDT)
oplog last event time:   Wed Oct 03 2012 14:59:10 GMT-0400 (EDT)
now:                     Wed Oct 03 2012 15:00:21 GMT-0400 (EDT)
```

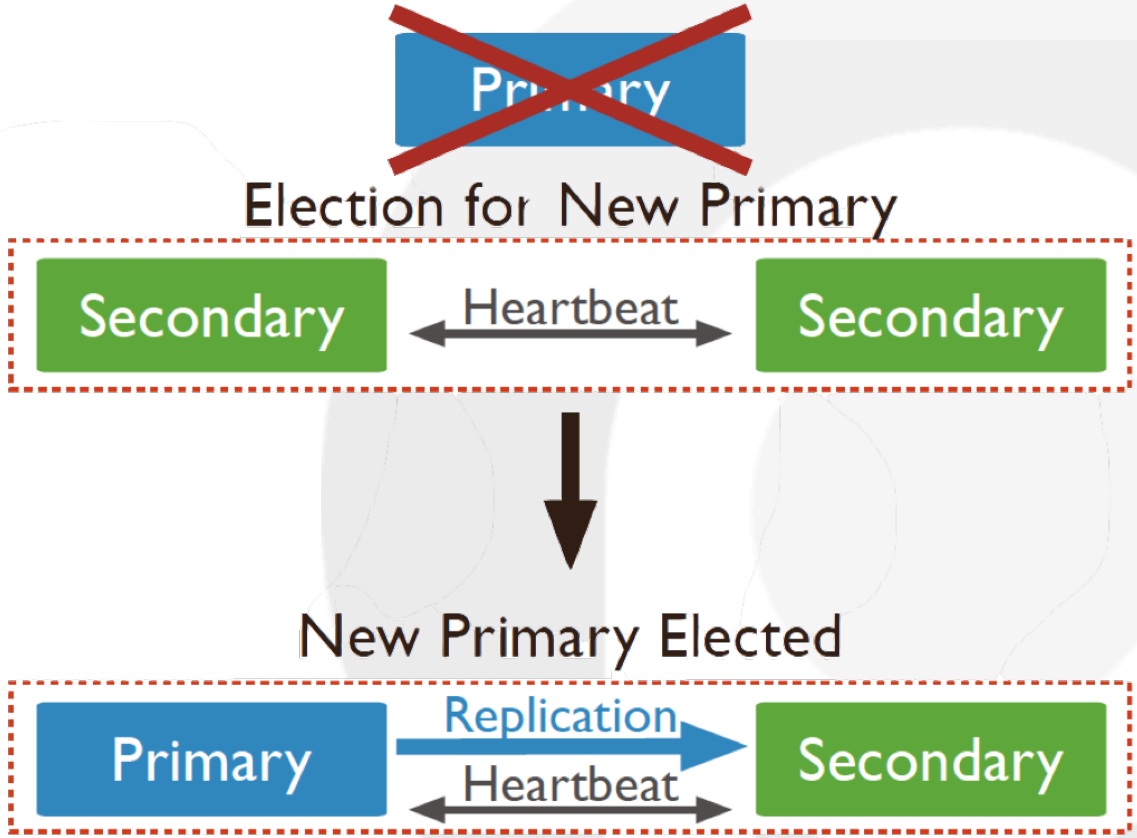
Replication Hearbeat



Election of New Primary

- When a primary does not communicate with the other members of the set for more than the configured `electionTimeoutMillis period` (10 seconds by default), an eligible secondary calls for an election to nominate itself as the new primary

Election of New Primary



Change Streams

- Access real-time data changes without tailing the oplog
- Applications can use change streams to subscribe to all data changes on a single collection, a database, or an entire deployment
- Change Streams rely on the aggregation framework
- Applications can also filter for specific changes or transform the notifications at will

Replication

CONFIGURATION

Replication Methods

- rs.initiate()
- rs.reconfig()
- rs.add()
- rs.addArb()
- rs.remove()

Replication Methods (2)

- rs.freeze()
- rs.stepDown()
- rs.syncFrom()
- rs.slaveOk()

How to Check Runtime Config and Status

- Inspect the replication configuration

```
rs.conf()
```

- Inspect the current status

```
rs.status()
```

- Check lag

```
rs.printSlaveReplicationInfo()
```

Configuration Options

- Available configuration values

```
{
  _id: <string>,
  version: <int>,
  protocolVersion: <number>,
  writeConcernMajorityJournalDefault: <boolean>,
  configsvr: <boolean>,
  members: [
    {
      _id: <int>,
      host: <string>,
      arbiterOnly: <boolean>,
      buildIndexes: <boolean>,
      hidden: <boolean>,
      priority: <number>,
      tags: <document>,
      slaveDelay: <int>,
      votes: <number>
    },
    ...
  ],
  settings: {
    chainingAllowed : <boolean>,
    heartbeatIntervalMillis : <int>,
    heartbeatTimeoutSecs: <int>,
    electionTimeoutMillis : <int>,
    catchUpTimeoutMillis : <int>,
    getLastErrorModes : <document>,
    getLastErrorDefaults : <document>,
    replicaSetId: <ObjectId>
  }
}
```

Replica Set Member States

- STARTUP
  - not yet member of any set
- PRIMARY
- SECONDARY
- ARBITER

Replica Set Member States (2)

- ROLLBACK
  - actively doing rollback
- RECOVERING
  - completing rollback or resync
- STARTUP2
  - running initial sync

Note: while on these states, nodes can still vote but not serve reads

Replica Set Member States (3)

- UNKNOWN
- DOWN
- REMOVED
  - was once a member of replica set

Replication Exercises

1. Stop any running mongod processes

```
sudo service mongod stop
```

2. Create data directories for 3 replica set members

```
sudo mkdir -p /mongodb/data/rs{1,2,3}
```

3. Give permissions

```
sudo chown mongod: /mongodb/data/*
```

Replication Exercises

3. Launch Each Member

- Launch each member on different `screen` session, we will not fork the processes in the background for this exercise

```
screen -S rs1 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs1 \
  --port 27017 --oplogSize 200 --wiredTigerCacheSizeGB 0.25

screen -S rs2 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs2 \
  --port 27018 --oplogSize 200 --wiredTigerCacheSizeGB 0.25

screen -S rs3 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs3 \
  --port 27019 --oplogSize 200 --wiredTigerCacheSizeGB 0.25
```

Replication Exercises

4. Configure the Replica Set

```
mongo // connect to the default port 27017

rs.initiate()
// wait a few seconds
rs.status()
```

Replication Exercises

5. Add the other nodes to the replica set

```
rs.add ('<HOSTNAME>:27018')
rs.addArb('<HOSTNAME>:27019')
// Keep running rs.status() until there's a primary and 2 secondaries
rs.status()
```

Replication

Lab: Problems that may occur

- `bindIp` parameter is incorrectly set
- Replica set configuration may need to be explicitly specified to use a different hostname:

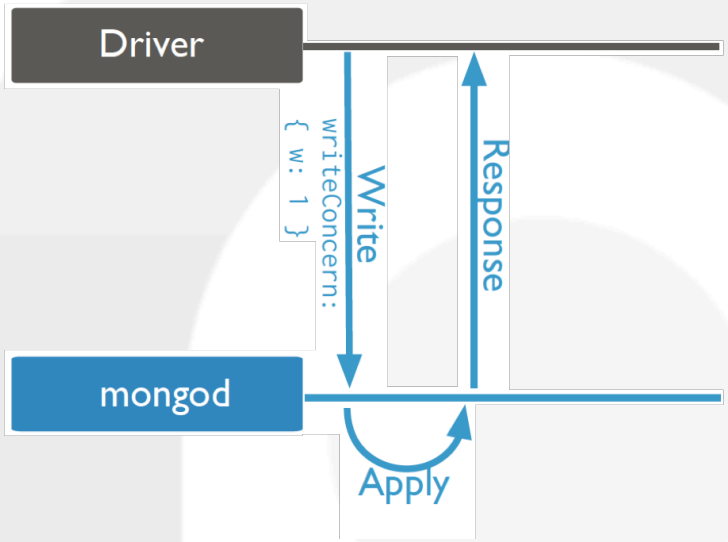
```
> conf = {
  _id: "<REPLICA-SET-NAME>",
  members: [
    { _id : 0, host : "<HOSTNAME>:27017"},
    { _id : 1, host : "<HOSTNAME>:27018"},
    { _id : 2, host : "<HOSTNAME>:27019", "arbiterOnly" : true},
  ]
}
> rs.initiate(conf)
```

Replication

READS AND WRITES

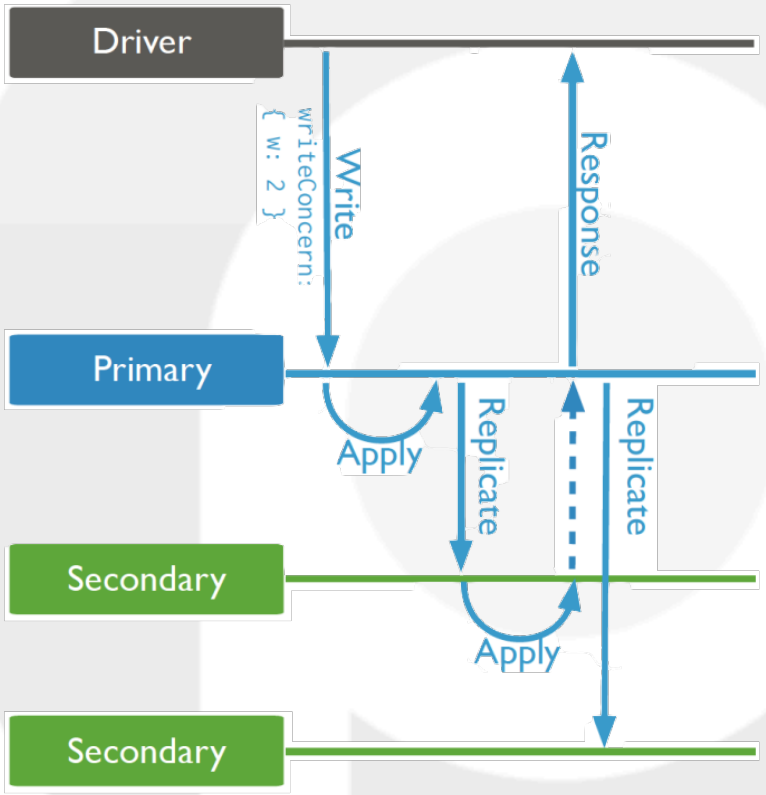
Replication and Write Concern

Write Concern: `{ w : 1 }`



Replication and Write Concern

Write Concern: `{ w : 2 }`



Replication and Write Concern

Write Concern: `{ w : "majority" }`

- Ensures the primary recorded the write to the on-disk journal
- Ensures write operations have propagated to a majority of a replica set's voting members
- Avoids hard-coding assumptions about the size of your replica set into your application
- Using *majority* trades off performance for durability
- It is suitable for critical writes and to avoid rollbacks

Replication and Read Concern

- Allows you to control the consistency and isolation properties of the data read from replica sets

```
readConcern: { level: <"local"|"available"|"majority"|linearizable"|"snapshot"> }
```

`local` Read Concern

- Default value for reads against primary
- The query returns the instance's most recent data
- Provides no guarantee that the data has been written to a majority of the replica set members (i.e. may be rolled back if primary crashes)

`available` Read Concern

- Default for reads against secondaries (*)
- Data might be rolled back
- Doesn't query shard's primary or config servers for updated metadata
  - Orphaned documents might be returned if chunk is being migrated

(*): except for causally consistent sessions in a sharded cluster

`majority` Read Concern

The query returns the instance's most recent data having been written to a majority of members in the replica set

- Documents returned are guaranteed to not roll back
- You can still read stale data in certain circumstances
  - e.g. a new primary was elected and updated data not still propagated

`linearizable` Read Concern

- Wait `maxTimeMS` for concurrently executing writes to propagate to a majority of replica set members before returning results
- Documents returned are guaranteed to not roll back
- Valid for read operations on the primary only

`snapshot` Read Concern

- Read concern `snapshot` is only available for multi-document transactions
- Controls the consistency of data your transactions read
- Affected by wether the transaction is part of a causally consistent session or not
- Use to ensure you read your own writes

Replication and Read Preference

MongoDB drivers support the following read preferences favoring the primary node:

- `primary`: default. Don't use secondaries.
- `primaryPreferred`: Read from the primary but if it is unavailable, read from secondary members.

Replication and Read Preference

MongoDB drivers also support the following read preferences:

- `secondary`: All operations read from the secondary members
- `secondaryPreferred`: Read from secondary members but if none are available, read from the primary.
- `nearest`: Read from member of the replica set with the least network latency, regardless of the member's type.

Replication

CHALLENGES

Replication Challenges

- Latency and lag
  - Starting in 4.2 MongoDB introduced the Flow control mechanism (10s target)
  - As lag gets close to the target, operations are required to get tickets
- Challenges with `chainingAllowed`
  - Allow replication from a secondary
  - Enabled by default
  - Disabling might cause extra load on primary

Replication

TAGS

Replication Tags

- Direct read operations to specific members
- Develop custom write concerns for multi-DC

Replication Tags

- Add tags to the members

```
conf = rs.conf()
conf.members[0].tags = { dc : "east", use : "production" }
conf.members[1].tags = { dc : "east", use : "reporting" }
conf.members[2].tags = { use : "production" }
rs.reconfig(conf)
```

- Specify tag sets in the read preference

```
db.collection.find({}).readPref( "secondary", [ { "dc": "east", "usage": "production" } ] )
```

Replication

LAB EXERCISES

Replication Exercises

1. Write to the Primary

```
use training
db.testcol.insert({ a: 1 })
db.testcol.count()
exit // Or Ctrl-d
```

2. Read from a Secondary

```
// Connect to one of the secondaries. E.g.:
mongo --port 27018
use training
db.testcol.find()
```

- What happens?

Replication Exercises

3. Try the read again

```
rs.slaveOk()
db.testcol.find()
```

4. Review the Oplog

```
use local
db.oplog.rs.find()
```

- Can you find your write?

Changing Member Priority

1. Login to the primary e.g.

```
mongo --port 27017
```

2. change priority of one SECONDARY

```
var conf = rs.conf()
conf.members[1].priority = 2

# verify with by displaying the conf variable
conf
```

3. Apply the configuration change

```
rs.reconfig(conf)
```

Changing Member Priority

4. Verify with rs.conf()

```
rs.conf().members[1]
```

- Changing priority causes an election, if new priority is highest
- Changing hidden + priority does not (unless current primary)

Using tags

1. Add tags to some member

```
conf = rs.conf()
conf.members[1].tags = { dc : "east", use : "reporting" }
rs.reconfig(conf)
```

2. Read from the tagged member

```
use training
db.testcol.find({}).readPref( "secondary", [ { "dc": "east", "usage": "production" } ] )
```

Resize the oplog

Perform on all secondaries first, then primary

1. Check current size in MB

```
use local
db.oplog.rs.stats().maxSize/1024/1024
```

2. Change the size (unit is MB)

```
db.adminCommand({replSetResizeOplog: 1, size: 1000})
```

- Online resize has to be > 990MB

3. Compact to reclaim space (only when secondary!)

```
db.runCommand({ "compact" : "oplog.rs" })
```

PERCONA

TRAINING

Questions