Replication Advanced

ANJU MUNOTH

Replica Set Data Synchronization

- In order to maintain up-to-date copies of the shared data set, secondary members of a replica set sync or replicate data from other members.
- MongoDB uses two forms of data synchronization: initial sync to populate new members with the full data set, and replication to apply ongoing changes to the entire data set.

Initial Sync

- Initial sync copies all the data from one member of the replica set to another member.
- Starting in MongoDB 4.4,can specify the preferred initial sync source using the initialSyncSourceReadPreference parameter.
- This parameter can only be specified when starting the mongod.

Initial Sync Process

1. Clones all databases except the local database.

- ➢ To clone, the mongod scans every collection in each source database and inserts all data into its own copies of these collections.
- Initial sync builds all collection indexes as the documents are copied for each collection. In earlier versions of MongoDB, only the _id indexes are built during this stage.
- Initial sync pulls newly added oplog records during the data copy. Ensure that the target member has enough disk space in the local database to temporarily store these oplog records for the duration of this data copy stage.
- Applies all changes to the data set. Using the oplog from the source, the mongod updates its data set to reflect the current state of the replica set.
- When the initial sync finishes, the member transitions from STARTUP2 to SECONDARY.

Fault Tolerance

- If a secondary performing initial sync encounters a non-transient (i.e. persistent) network error during the sync process, the secondary restarts the initial sync process from the beginning.
- Starting in MongoDB 4.4, a secondary performing initial sync can attempt to resume the sync process if interrupted by a transient (i.e. temporary) network error, collection drop, or collection rename.
- The sync source must also run MongoDB 4.4 to support resumable initial sync.
- If the sync source runs MongoDB 4.2 or earlier, the secondary must restart the initial sync process as if it encountered a non-transient network error.
- By default, the secondary tries to resume initial sync for 24 hours.
- MongoDB 4.4 adds the initialSyncTransientErrorRetryPeriodSeconds server parameter for controlling the amount of time the secondary attempts to resume initial sync
- If the secondary cannot successfully resume the initial sync process during the configured time period, it selects a new healthy source from the replica set and restarts the initial synchronization process from the beginning.
- The secondary attempts to restart the initial sync up to 10 times before returning a fatal error.

Initial Sync Source Selection

- Initial sync source selection depends on the value of the mongod startup parameter initialSyncSourceReadPreference (new in 4.4):
- For initialSyncSourceReadPreference set to primary (default if chaining is disabled), select the primary as the sync source.
- If the primary is unavailable or unreachable, log an error and periodically check for primary availability.
- For initialSyncSourceReadPreference set to primaryPreferred (default for voting replica set members), attempt to select the primary as the sync source.
- If the primary is unavailable or unreachable, perform sync source selection from the remaining replica set members.
- For all other supported read modes, perform sync source selection from the replica set members.

Initial Sync Source Selection

- Members performing initial sync source selection make two passes through the list of all replica set members:
  - Sync Source Selection (First Pass)
  - Sync Source Selection (Second Pass)
- If the member cannot select an initial sync source after two passes, it logs an error and waits 1 second before restarting the selection process.
- The secondary mongod can restart the initial sync source selection process up to 10 times before exiting with an error.

Sync Source Selection (First Pass)

Member applies the following criteria to each replica set member when making the first pass for selecting a initial sync source:

- The sync source must be in the PRIMARY or SECONDARY replication state.
- The sync source must be online and reachable.
- If initialSyncSourceReadPreference is secondary or secondaryPreferred, the sync source must be a secondary.
- The sync source must be visible.
- The sync source must be within 30 seconds of the newest oplog entry on the primary.
- If the member builds indexes, the sync source must build indexes.
- If the member votes in replica set elections, the sync source must also vote.
- If the member is not a delayed member, the sync source must not be delayed.
- If the member is a delayed member, the sync source must have a shorter configured delay.
- The sync source must be faster (i.e. lower latency) than the current best sync source.

Sync Source Selection (Second Pass)

If no candidate sync sources remain after the first pass, the member performs a second pass with relaxed criteria.

- member applies the following criteria to each replica set member when making the second pass for selecting a initial sync source:
- The sync source must be in the PRIMARY or SECONDARY replication state.
- The sync source must be online and reachable.
- If initialSyncSourceReadPreference is secondary, the sync source must be a secondary.
- If the member builds indexes, the sync source must build indexes.
- The sync source must be faster (i.e. lower latency) than the current best sync source.

Streaming Replication

- Starting in MongoDB 4.4, sync from sources send a continuous stream of oplog entries to their syncing secondaries.
- Streaming replication mitigates replication lag in high-load and high-latency networks.

It also:

- Reduces staleness for reads from secondaries.
- Reduces risk of losing write operations with w: 1 due to primary failover.
- Reduces latency on write operations with w: "majority" and w: >1 (that is, any write concern that requires waiting for replication).

Streaming Replication

- Prior to MongoDB 4.4, secondaries fetched batches of oplog entries by issuing a request to their sync from source and waiting for a response.
- This required a network roundtrip for each batch of oplog entries.
- MongoDB 4.4 adds the oplogFetcherUsesExhaust startup parameter for disabling streaming replication and using the older replication behavior.
- Set the oplogFetcherUsesExhaust parameter to false only if there are any resource constraints on the sync from source or if you wish to limit MongoDB's usage of network bandwidth for replication.

Multithreaded Replication

- MongoDB applies write operations in batches using multiple threads to improve concurrency.
- MongoDB groups batches by document id (WiredTiger) and simultaneously applies each group of operations using a different thread.
- MongoDB always applies write operations to a given document in their original write order.
- Starting in MongoDB 4.0, read operations that target secondaries and are configured with a read concern level of "local" or "majority" will now read from a WiredTiger snapshot of the data if the read takes place on a secondary where replication batches are being applied.
- Reading from a snapshot guarantees a consistent view of the data, and allows the read to occur simultaneously with the ongoing replication without the need for a lock.
- As a result, secondary reads requiring these read concern levels no longer need to wait for replication batches to be applied, and can be handled as they are received.

Flow Control

- Starting in MongoDB 4.2, administrators can limit the rate at which the primary applies its writes with the goal of keeping the majority committed lag under a configurable maximum value flowControlTargetLagSeconds.

- By default, flow control is enabled.
- For flow control to engage, the replica set/sharded cluster must have: featureCompatibilityVersion (FCV) of 4.2 and read concern majority enabled.

Replication Sync Source Selection

- Replication sync source selection depends on the replica set chaining setting:
- With chaining enabled (default), perform sync source selection from the replica set members.
- With chaining disabled, select the primary as the sync source.
- If the primary is unavailable or unreachable, log an error and periodically check for primary availability.
- Members performing replication sync source selection make two passes through the list of all replica set members:
  - Sync Source Selection (First Pass)
  - Sync Source Selection (Second Pass)
- If no candidate sync sources remain after the first pass, the member performs a second pass with relaxed criteria.

Sync Source Selection (First Pass)

Member applies the following criteria to each replica set member when making the first pass for selecting a replication sync source:

- The sync source must be in the PRIMARY or SECONDARY replication state.
- The sync source must be online and reachable.
- The sync source must have newer oplog entries than the member (i.e. the sync source is ahead of the member).
- The sync source must be visible.
- The sync source must be within 30 seconds of the newest oplog entry on the primary.
- If the member builds indexes, the sync source must build indexes.
- If the member votes in replica set elections, the sync source must also vote.
- If the member is not a delayed member, the sync source must not be delayed.
- If the member is a delayed member, the sync source must have a shorter configured delay.
- The sync source must be faster (i.e. lower latency) than the current best sync source.

Sync Source Selection (Second Pass)

member applies the following criteria to each replica set member when making the second pass for selecting a replication sync source:

- The sync source must be in the PRIMARY or SECONDARY replication state.
- The sync source must be online and reachable.
- If the member builds indexes, the sync source must build indexes.
- The sync source must be faster (i.e. lower latency) than the current best sync source.

➢ **If the member cannot select a sync source after two passes, it logs an error and waits 1 second before restarting the selection process.**

Replica set Elections

Replica sets can trigger an election in response to a variety of events, such as:

- Adding a new node to the replica set,
- initiating a replica set,
- performing replica set maintenance using methods such as rs.stepDown() or rs.reconfig(), and
- the secondary members losing connectivity to the primary for more than the configured timeout (10 seconds by default).

Replica set Elections

- median time before a cluster elects a new primary should not typically exceed 12 seconds, assuming default replica configuration settings.
- Includes time required to mark the primary as unavailable and call and complete an election.
- Can tune this time period by modifying the **settings.electionTimeoutMillis** replication configuration option.
- Factors such as network latency may extend the time required for replica set elections to complete, which in turn affects the amount of time your cluster may operate without a primary.
- MongoDB drivers can detect the loss of the primary and automatically retry certain write operations a single time, providing additional built-in handling of automatic failovers and elections:
- MongoDB 4.2-compatible drivers enable retryable writes by default
- MongoDB 4.0 and 3.6-compatible drivers must explicitly enable retryable writes by including retryWrites=true in the connection string.

Replica Set Configuration

- Can access the configuration of a replica set using **the rs.conf()** method or the **replSetGetConfig** command.

- To modify the configuration for a replica set, use the rs.reconfig() method, passing a configuration document to the method

Replica Set Configuration Document Example

- Replica Set Configuration Document Example.docx

members[n].buildIndexes

- Type: Boolean ;Default: true
- A boolean that indicates whether the mongod builds indexes on this member.
- Can only set this value when adding a member to a replica set.
- Cannot change members[n].buildIndexes field after the member has been added to the set.
- Do not set to false for mongod instances that receive queries from clients.
- Setting buildIndexes to false may be useful if all the following conditions are true:
  - Only using this instance to perform backups using mongodump, and
  - this member will receive no queries, and
  - index creation and maintenance overburdens the host system.
- Even if set to false, secondaries will build indexes on the _id field in order to facilitate operations required for replication.

settings.chainingAllowed

- Optional.Type: Boolean;Default: true

- When settings.chainingAllowed is true, the replica set allows secondary members to replicate from other secondary members.
- When settings.chainingAllowed is false, secondaries can replicate only from the primary.

settings.getLastErrorDefaults

- Optional.Type: document

- A document that specifies the write concern for the replica set.
- The replica set will use this write concern only when write operations or getLastError specify no other write concern.
- If settings.getLastErrorDefaults is not set, the default write concern for the replica set only requires confirmation from the primary.

settings.catchUpTimeoutMillis

- Default: -1, infinite catchup time.
- Time limit in milliseconds for a newly elected primary to sync (catch up) with the other replica set members that may have more recent writes.
- Infinite or high time limits may reduce the amount of data that the other members would need to roll back after an election but may increase the failover time.
- The newly elected primary ends the catchup period early once it is fully caught up with other members of the set.
- During the catchup period, the newly elected primary is unavailable for writes from clients.
- Use replSetAbortPrimaryCatchUp to abort the catchup then complete the transition to primary.

settings.catchUpTakeoverDelayMillis

- Default: 30000 (30 seconds)
- Time in milliseconds a node waits to initiate a catchup takeover after determining it is ahead of the current primary.
- During a catchup takeover, the node ahead of the current primary initiates an election to become the new primary of the replica set.
- After the node initiating the takeover determines that it is ahead of the current primary, it waits the specified number of milliseconds and then verifies the following:
  - It is still ahead of the current primary,
  - It is the most up-to-date node among all available nodes,
  - The current primary is currently catching up to it.
- Once determining that all of these conditions are met, the node initiating the takeover immediately runs for election.

replSetGetStatus

- replSetGetStatus command returns the status of the replica set from the point of view of the server that processed the command.
- replSetGetStatus must be run against the admin database.
- The mongod instance must be a replica set member for replSetGetStatus to return successfully.
- Data provided by this command derives from data included in heartbeats sent to the server by other members of the replica set.
- Because of the frequency of heartbeats, these data can be several seconds out of date.

replSetGetStatus

- Syntax: Run it on the admin database

```
db.adminCommand( { replSetGetStatus: 1 } )
```

Output:

replSetGetStatus.docx

replSetGetStatus.initialSyncStatus.databases.<dbname>

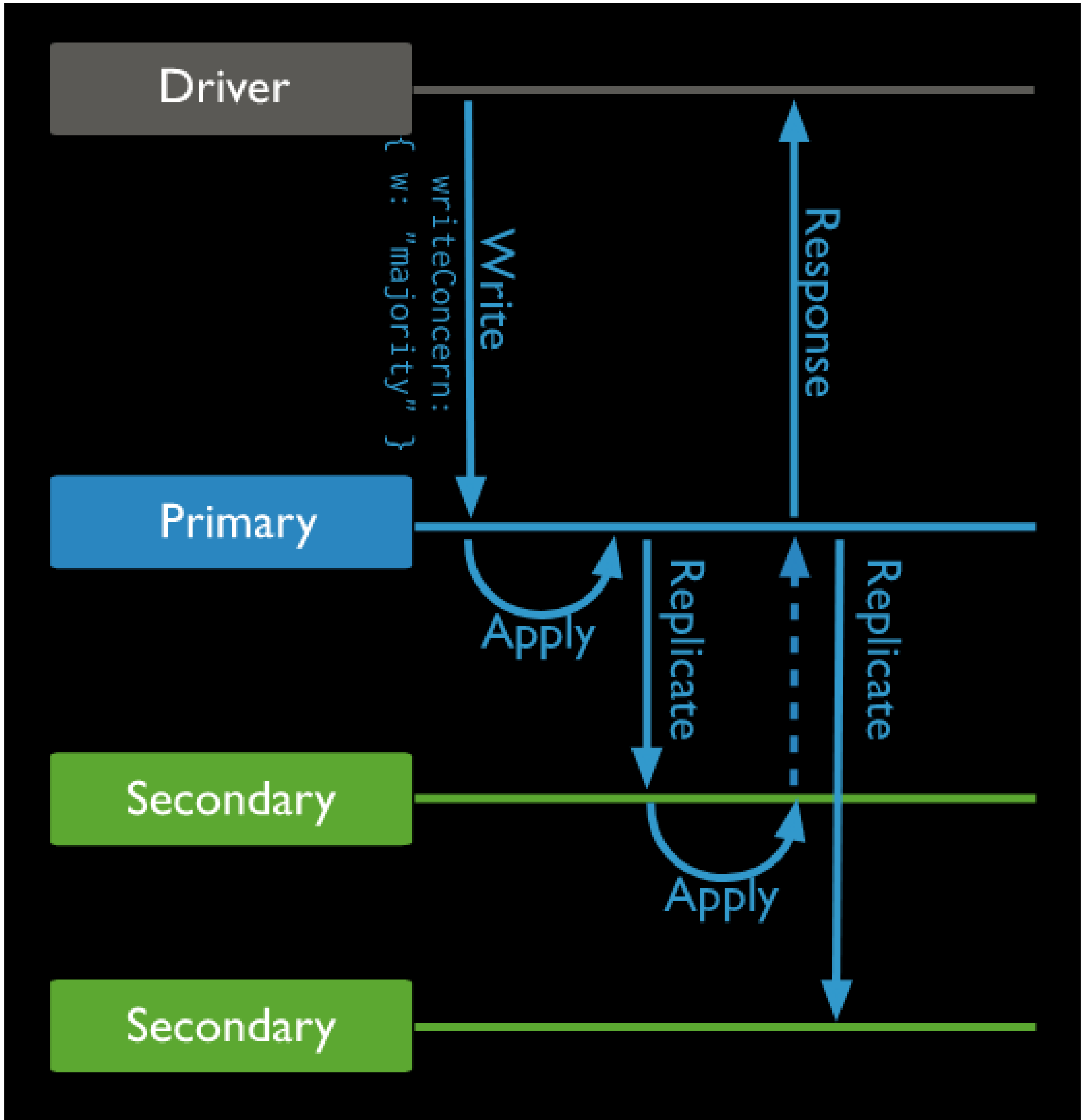
- For each database, a document that returns information regarding the progress of the cloning of that database.

replSetGetStatus.initialSyncStatus.databases.<dbname>

```
{
  "collections" : <number of collections to clone in the database>,
  "clonedCollections" : <number of collections cloned to date>,
  "start" : <start date and time for the database clone>,
  "end" : <end date and time for the database clone>,
  "elapsedMillis" : <duration of the database clone>,
  "<db>.<collection>" : {
    "documentsToCopy" : <number of documents to copy>,
    "documentsCopied" : <number of documents copied to date>,
    "indexes" :  <number of indexes>,
    "fetchedBatches" :  <number of batches of documents fetched to
date>,
    "start" :  <start date and time for the collection clone>,
    "end" : <end date and time for the collection clone>,
    "elapsedMillis" : <duration of the collection clone>,
    "receivedBatches" : <number of batches of documents received to
date>  // Added in 4.2
  }
}
```

Write Concern for Replica Sets

- Write concern for replica sets describe the number of data-bearing members (i.e. the primary and secondaries, but not arbiters) that must acknowledge a write operation before the operation returns as successful.
- A member can only acknowledge a write operation after it has received and applied the write successfully.
- For replica sets, the default write concern of w: 1 requires that only the primary replica set member acknowledge the write before returning write concern acknowledgment
- Can specify an integer value greater than 1 to require acknowledgment from the primary and as many secondaries as needed to meet the specified value, up to the total number of data-bearing members in the replica set.
- Write operations with a write concern of "majority" require acknowledgement that the write operations have propagated to a calculated majority of the data-bearing voting members.
- For clusters where members have journaling enabled, combining "majority" write concern with j : true can prevent rollback of write concern acknowledged data.


Driver
Primary
Secondary
Secondary
Write
writeConcern:
{ w: "majority" }
Response
Apply
Replicate
Replicate
Apply

Write Concern for Replica Sets

- An application that issues a write operation that requires write concern acknowledgment waits until the primary receives acknowledgment from the required number of members for the specified write concern.
- For write concern of w greater than 1 or w : "majority", the primary waits until the required number of secondaries acknowledge the write before returning write concern acknowledgment.
- For write concern of w: 1, the primary can return write concern acknowledgment as soon as it locally applies the write since it is eligible for contributing to the requested write concern.
- The more members that acknowledge a write, the less likely the written data could roll back if the primary fails.
- However, specifying a high write concern can increase latency as the client must wait until it receives the requested level of write concern acknowledgment.
- Selecting the ideal write concern for any given write operation depends on your application’s performance goals and data durability requirements.

Modify Default Write Concern

- Can modify the default write concern for a replica set by setting the settings.getLastErrorDefaults setting in the replica set configuration.
- The following sequence of commands creates a configuration that waits for the write operation to complete on a majority of the voting members before returning:

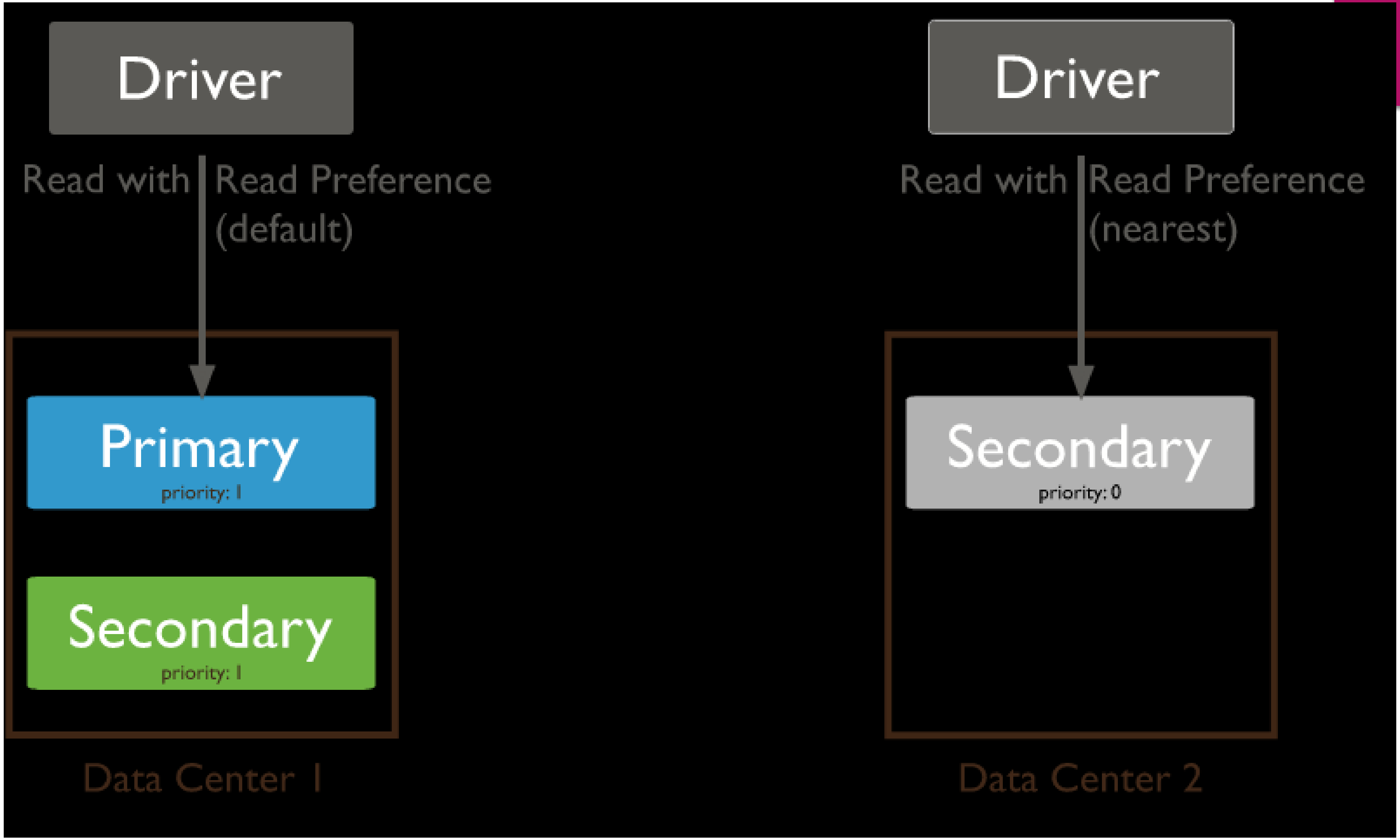
```
cfg = rs.conf()
cfg.settings.getLastErrorDefaults = { w: "majority", wtimeout: 5000 }
rs.reconfig(cfg)
```

- If you issue a write operation with a specific write concern, the write operation uses its own write concern instead of the default.

Read Preference

- Read preference describes how MongoDB clients route read operations to the members of a replica set.
- By default, an application directs its read operations to the primary member in a replica set (i.e. read preference mode “primary”). But, clients can specify a read preference to send read operations to secondaries.

- Read preference consists of the read preference mode and optionally, a tag set, the maxStalenessSeconds option


Driver
Read with Read Preference (default)
Primary
priority: 1
Secondary
priority: 1
Data Center 1
Driver
Read with Read Preference (nearest)
Secondary
priority: 0
Data Center 2

Read preference mode

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 on sharded clusters.
`nearest`	Operations read from member of the replica set with the least network latency, irrespective of the member's type.

Primary preferred

- When the primaryPreferred read preference includes a maxStalenessSeconds value and there is no primary from which to read, the client estimates how stale each secondary is by comparing the secondary's last write to that of the secondary with the most recent write.
- The client then directs the read operation to a secondary whose estimated lag is less than or equal to maxStalenessSeconds.

Configure Read Preference

- When using a MongoDB driver, you can specify the read preference using the driver's read preference API.
- Can also set the read preference (except for the hedged read option) when connecting to the replica set or sharded cluster.
- For a given read preference, the MongoDB drivers use the same member selection logic.
- When using the mongo shell, use cursor.readPref() and Mongo.setReadPref()

Specify Read Preference Mode

- The following operation sets the read preference mode to target the read to a secondary member. This implicitly allows reads from secondaries.

-
```
db.getMongo().setReadPref('secondary')
```

Read Preference for Replica Sets

- Server selection occurs once per operation and is governed by the read preference and localThresholdMS settings to determine member eligibility for reads.
- Read preference is re-evaluated for each operation.

Read Preference Mode	Selection Process
`primary` (Default)	1.The driver selects the primary.
`secondary`	1.The driver assembles a list of eligible secondary members. maxStalenessSeconds and tag sets specified in the read preference can further restrict the eligibility of the members. 2.If the list of eligible members is not empty, the driver determines which eligible member is the "closest" (i.e. the member with the lowest average network round-trip-time) and calculates a latency window by adding the average round-trip-time of this "closest" server and the `localThresholdMS`. The driver uses this latency window to pare down the list of eligible members to those members that fall within this window. 3.From this list of eligible members that fall within the latency window, the driver randomly chooses an eligible member.

maxStalenessSeconds

- Replica set members can lag behind the primary due to network congestion, low disk throughput, long-running operations, etc.
- The read preference maxStalenessSeconds option lets you specify a maximum replication lag, or "staleness", for reads from secondaries.
- When a secondary's estimated staleness exceeds maxStalenessSeconds, the client stops using it for read operations
- **maxStalenessSeconds read preference option is intended for applications that read from secondaries and want to avoid reading from a secondary that has fallen overly far behind in replicating the primary's writes.**
- For example, a secondary might stop replicating due to a network outage between itself and the primary. In that case, the client should stop reading from the secondary until an administrator resolves the outage and the secondary catches up.

maxStalenessSeconds

can specify maxStalenessSeconds with the following read preference modes:

- primaryPreferred
- secondary
- secondaryPreferred
- nearest
- Max staleness is not compatible with mode primary and only applies when selecting a secondary member of a set for a read operation.

maxStalenessSeconds

- When selecting a server for a read operation with maxStalenessSeconds, clients estimate how stale each secondary is by comparing the secondary's last write to that of the primary.
- The client will then direct the read operation to a secondary whose estimated lag is less than or equal to maxStalenessSeconds.
- If there is no primary, the client uses the secondary with the most recent write for the comparison.
- By default, there is no maximum staleness and clients will not consider a secondary's lag when choosing where to direct a read operation.
- Must specify a maxStalenessSeconds value of 90 seconds or longer: specifying a smaller maxStalenessSeconds value will raise an error.
- Clients estimate secondaries' staleness by periodically checking the latest write date of each replica set member.
- Since these checks are infrequent, the staleness estimate is coarse.
- Thus, clients cannot enforce a maxStalenessSeconds value of less than 90 seconds.

rs.stepDown()

- Instructs the primary of the replica set to become a secondary.
- After the primary steps down, eligble secondaries will hold an election for primary.
- The method does not immediately step down the primary.
- If no electable secondaries are up to date with the primary, the primary waits up to secondaryCatchUpPeriodSecs (by default 10 seconds) for a secondary to catch up.
- Once an electable secondary is available, the method steps down the primary.
- Once stepped down, the original primary becomes a secondary and is ineligible from becoming primary again for the remainder of time specified by stepDownSecs.

rs.stepDown()

rs.stepDown(stepDownSecs, secondaryCatchUpPeriodSecs)

- stepDownSecs number
  - The number of seconds to step down the primary, during which time the stepdown member is ineligible for becoming primary. If you specify a non-numeric value, the command uses 60 seconds.
  - The stepdown period starts from the time that the mongod receives the command. The stepdown period must be greater than the secondaryCatchUpPeriodSecs.
- secondaryCatchUpPeriodSecs number
  - Optional. The number of seconds that mongod will wait for an electable secondary to catch up to the primary.
  - When specified, secondaryCatchUpPeriodSecs overrides the default wait time of 10 seconds.

Behavior

- rs.stepDown() method attempts to terminate long running user operations that block the primary from stepping down, such as an index build, a write operation or a map-reduce job.
- The method then initiates a catchup period where it waits up to secondaryCatchUpPeriodSeconds, by default 10 seconds, for a secondary to become up-to-date with the primary.
- The primary only steps down if a secondary is up-to-date with the primary during the catchup period to prevent rollbacks.
- If no electable secondary meets this criterion by the end of the waiting period, the primary does not step down and the method errors.
- Once the primary steps down successfully, that node cannot become the primary for the remainder of the stepDownSecs period, which began when the node received the method

Writes During Stepdown

- All writes to the primary fail during the period starting when the rs.stepDown() method is received until either a new primary is elected, or if there are no electable secondaries, the original primary resumes normal operation.
- Writes that were in progress when rs.stepDown() is run are killed. In-progress transactions also fail with "TransientTransactionError" and can be retried as a whole.
- The time period where writes fail is at maximum:
  - secondaryCatchUpPeriodSecs (10s by default) + electionTimeoutMillis (10s by default).

local Database

- Every mongod instance has its own local database, which stores data used in the replication process, and other instance-specific data.
- The local database is invisible to replication: collections in the local database are not replicated.

Collection on all mongod Instances

- local.startup_log
- local.system.replset
- local.oplog.rs
- local.replset.minvalid

local.startup_log

- On startup, each mongod instance inserts a document into startup_log with diagnostic information about the mongod instance itself and host information. startup_log is a capped collection.
- This information is primarily useful for diagnostic purposes.

```
{
 "_id" : "<string>",
 "hostname" : "<string>",
 "startTime" : ISODate("<date>"),
 "startTimeLocal" : "<string>",
 "cmdLine" : {
      "dbpath" : "<path>",
      "<option>" : <value>
 },
 "pid" : <number>,
 "buildinfo" : {
      "version" : "<string>",
      "gitVersion" : "<string>",
      "sysInfo" : "<string>",
      "loaderFlags" : "<string>",
      "compilerFlags" : "<string>",
      "allocator" : "<string>",
      "versionArray" : [ <num>, <num>, <...> ],
      "javascriptEngine" : "<string>",
      "bits" : <number>,
      "debug" : <boolean>,
      "maxBsonObjectSize" : <number>
 }   }
```

Prototype of a document from the `startup_log` collection:

Local database collections

local.system.replset

- local.system.replset holds the replica set's configuration object as its single document.
- To view the object's configuration information, issue rs.conf() from the mongo shell.
- Can also query this collection directly.

local.oplog.rs

- local.oplog.rs is the capped collection that holds the oplog.
- You set its size at creation using the oplogSizeMB setting.
- To resize the oplog after replica set initiation, use the Change the Size of the Oplog procedure.

Replica Set Member States

PRIMARY

- Members in PRIMARY state accept write operations. A replica set has at most one primary at a time.
- A SECONDARY member becomes primary after an election.
- Members in the PRIMARY state are eligible to vote.

SECONDARY

- Members in SECONDARY state replicate the primary’s data set and can be configured to accept read operations.
- Secondaries are eligible to vote in elections, and may be elected to the PRIMARY state if the primary becomes unavailable.

ARBITER

- Members in ARBITER state do not replicate data or accept write operations.
- They are eligible to vote, and exist solely to break a tie during elections.
- Replica sets should only have a member in the ARBITER state if the set would otherwise have an even number of voting members, and could suffer from tied elections. There should only be at most one arbiter configured in any replica set

Replica Set Member States

STARTUP

- Each member of a replica set starts up in STARTUP state.
- mongod then loads that member's replica set configuration, and transitions the member's state to STARTUP2 or ARBITER.
- Members in STARTUP are not eligible to vote, as they are not yet a recognized member of any replica set.

STARTUP2

- Each data-bearing member of a replica set enters the STARTUP2 state as soon as mongod finishes loading that member's configuration, at which time it becomes an active member of the replica set and is eligible to vote.
- The member then decides whether or not to undertake an initial sync.
- If a member begins an initial sync, the member remains in STARTUP2 until all data is copied and all indexes are built.
- Afterwards, the member transitions to RECOVERING.

Replica Set Member States

RECOVERING

- A member of a replica set enters RECOVERING state when it is not ready to accept reads.
- The RECOVERING state can occur during normal operation, and doesn't necessarily reflect an error condition.
- Members in the RECOVERING state are eligible to vote in elections, but are not eligible to enter the PRIMARY state.
- A member transitions from RECOVERING to SECONDARY after replicating enough data to guarantee a consistent view of the data for client reads.
- The only difference between RECOVERING and SECONDARY states is that RECOVERING prohibits client reads and SECONDARY permits them.
- SECONDARY state does not guarantee anything about the staleness of the data with respect to the primary.
- Due to overload, a secondary may fall far enough behind the other members of the replica set such that it may need to resync with the rest of the set.
- When this happens, the member enters the RECOVERING state and requires manual intervention.

Replica Set Member States

- ROLLBACK
- Whenever the replica set replaces a primary in an election, the old primary may contain documents that did not replicate to the secondary members.
- In this case, the old primary member reverts those writes.
- During rollback, the member will have ROLLBACK state.
- Members in the ROLLBACK state are eligible to vote in elections.

Replica Set Member States

Error States

- Members in any error state can’t vote.
- UNKNOWN
  - Members that have never communicated status information to the replica set are in the UNKNOWN state.
- DOWN
  - Members that lose their connection to the replica set are seen as DOWN by the remaining members of the set.
- REMOVED
  - Members that are removed from the replica set enter the REMOVED state. When members enter the REMOVED state, the logs will mark this event with a replSet REMOVED message entry.