MongoDB

Replication

Bingo Bango Mongo

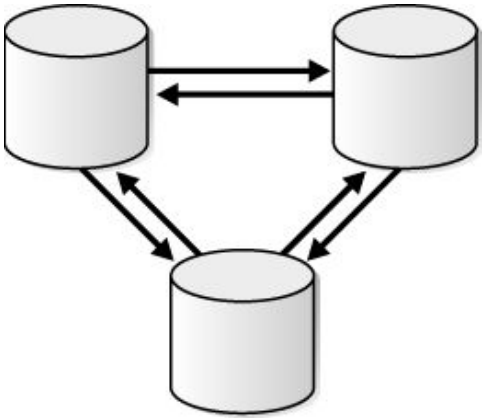
Michelle Duer Ian Percy Billy Haugen Yves Wienecke Tejas Menon

Replication

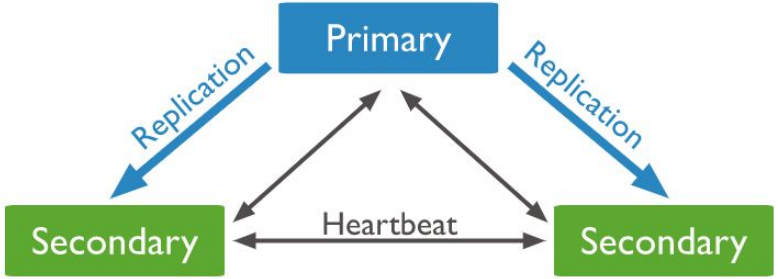
Store data in more than one node or instance while maintaining consistency

Benefits:

- Redundancy
- Availability
- Fault Tolerance
- Increased Read Capacity



Replica Sets



Group of mongod nodes (primary daemon process for the MongoDB system) that maintain the same data set

Contains multiple data bearing nodes and possibly one “arbiter” node

Primary Nodes

- Receives all write operations and changes to data are recorded in an oplog
- Only node capable of confirming writes

Secondary Nodes

- Replicate the primary’s oplog and apply operations to their data

Replica Sets, cont.

Hidden Replica Set Members

- Maintain a copy of primary data, but is invisible to the client.
- Useful for operations such as reporting or backups

Delayed Replica Set Members

- Maintain a copy of replica set data, but with a “delay”
  - Ex - If the current time is 10:00 and the delay on the node is an hour, no operations will show more recent than 9:00
  - Assists with rolling back in the event of failures

Ian Percy

Replica Set Read and Write

Writes can be configured on a scale to write only to the primary or to a certain number of secondary nodes as well

Read can occur from the primary or secondary (depending on choice)

- Readconcern is a setting that allows for reading varying levels of the replicated data
  - Can choose to read data that has been committed to majority of the nodes or most recent

Communication

- Default Port is 27017
- Two types of messages
  - Client requests
  - Database response
- Standard message header

```
struct MsgHeader {
    int32   messageLength; // total message size, including this
    int32   requestID;     // identifier for this message
    int32   responseTo;    // requestID from the original request
                           //   (used in responses from db)
    int32   opCode;        // request type - see table below for details
}
```

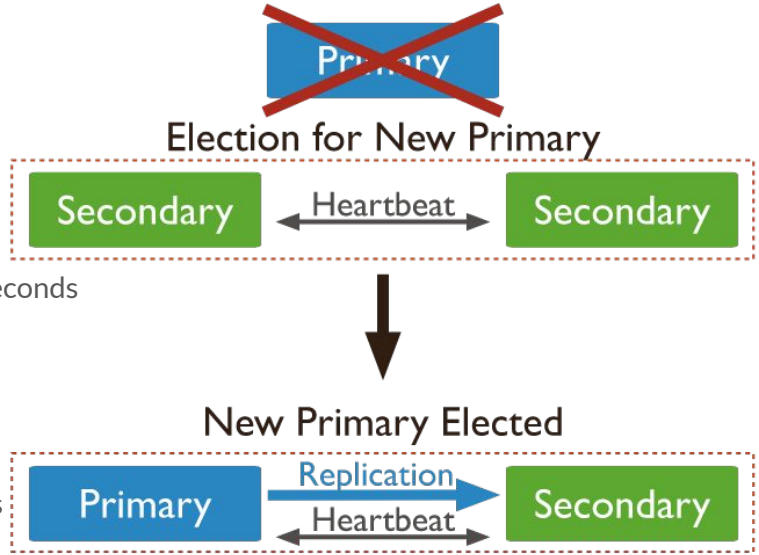
```
struct OP_UPDATE {
    MsgHeader header;             // standard message header
    int32     ZERO;               // 0 - reserved for future use
    cstring   fullCollectionName; // "dbname.collectionname"
    int32     flags;              // bit vector. see below
    document  selector;           // the query to select the document
    document  update;             // specification of the update to perform
}
```

```
struct {
    MsgHeader header;         // standard message header
    int32     responseFlags;  // bit vector - see details below
    int64     cursorID;       // cursor id if client needs to do get more's
    int32     startingFrom;   // where in the cursor this reply is starting
    int32     numberReturned; // number of documents in the reply
    document* documents;      // documents
}
```

Billy Haugen

Voting

- The sets will communicate to each other with a heartbeat
  - Default settings
    - Pings every 2 seconds
    - Considered unavailable if no ping is received for 10 seconds
    - Election timeout at 12 seconds
- Unable to perform writes while an election in process
  - Can do reads if configured to access the secondary
- Network Partitioning
  - Steps down to secondary if it can only see a minority of nodes
  - Secondary sees majority, holds election



Billy Haugen

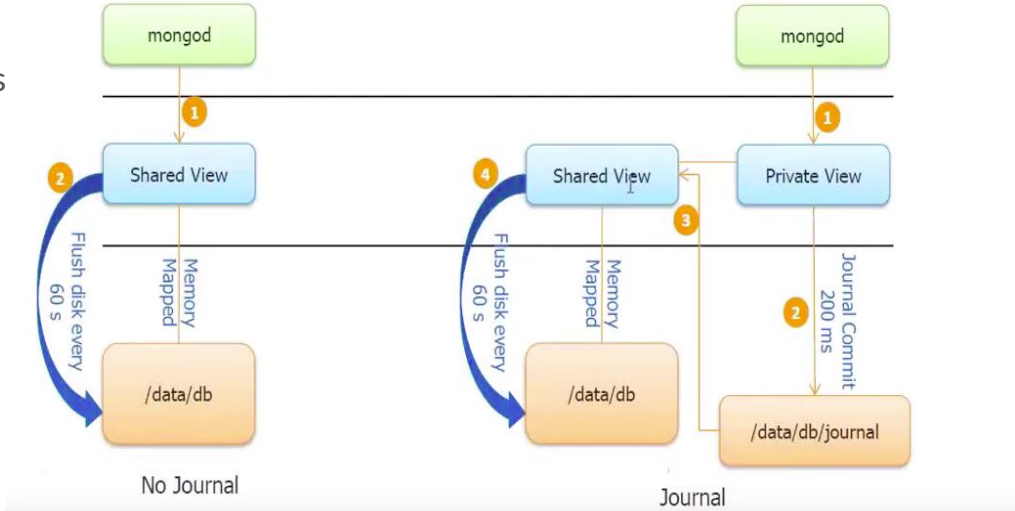
Fault Tolerance

- Journaling against power failures
  - Journaling Mechanics
  - Write Concern
  - Oplogs and Journals

Tejas Menon

Journaling Mechanics

- A method of saving incremental updates to memory before committing them to disk
- Read Operations can continue on secondary servers (if specified) while primary is down
- Different views allows for separation of committed versus uncommitted data.
- Decreasing journal commit intervals increases overall shared view -> disk latency.



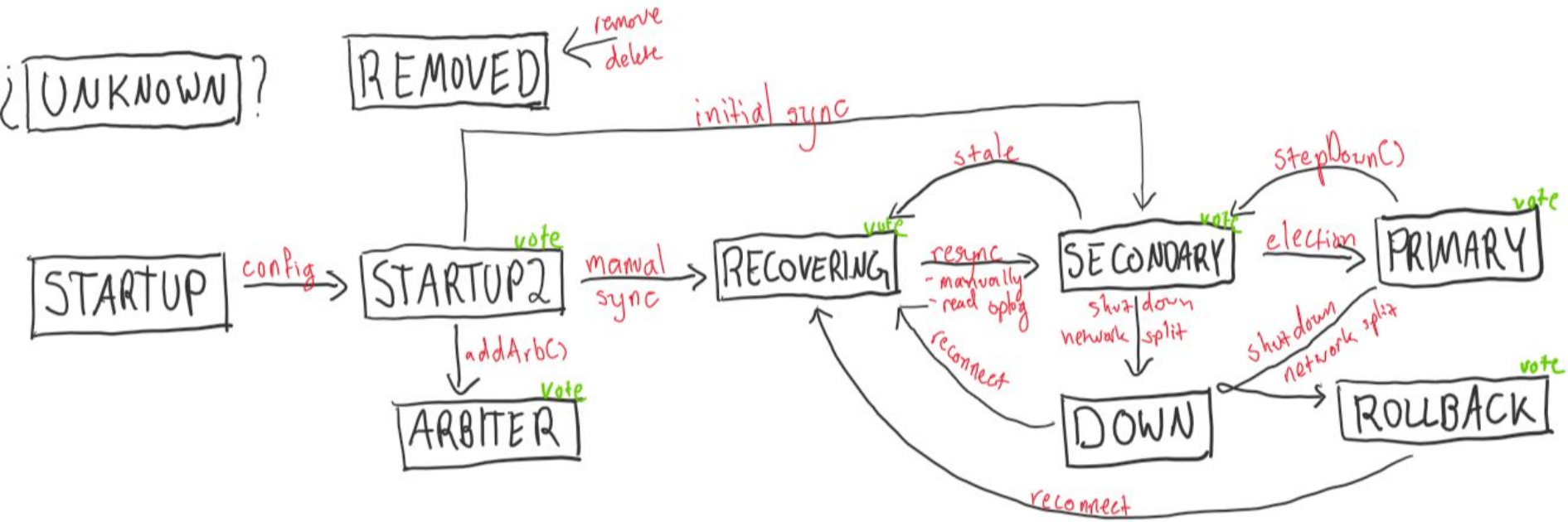
Img: https://www.youtube.com/watch?v=s0LnwqIwMhE
Content: https://docs.mongodb.com/manual/core/journaling/

Tejas Menon

Oplogs and Journals

- Oplogs
  - Oplog maintains all operations performed on the primary which is periodically polled by secondary nodes to apply data changes
  - Operations sometimes get transformed before being stored in the oplog so that they are idempotent (can be safely applied many times).
  - High level instructions to facilitate system-wide *consistency*,
- Journals
  - Journals are a per-server *durability* feature (Can also be switched off)
  - Low level instructions like 'write these bytes to this file at this position'

REPLICA SET MEMBER STATES



YVES WIENECKE

REPLICA SET MEMBER STATES

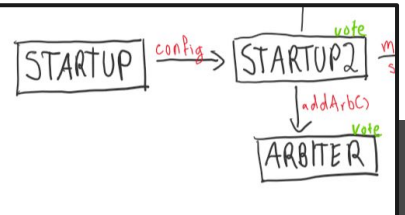
#0
STARTUP
Not active

→ [mongod finishes configuration]

#5
STARTUP2
Active member

→ [rs.addArb()]

#7
ARBITER
Election tie-breaker



REPLICA SET MEMBER STATES

#5
STARTUP2
Active member

[Manual sync]

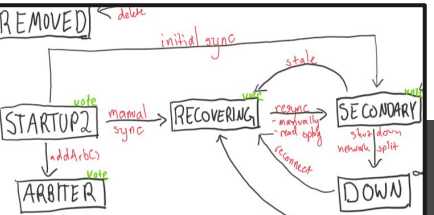
↗

↘

[inital sync]

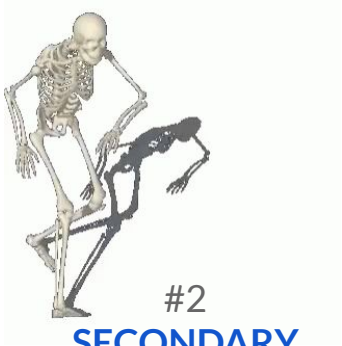
#3
RECOVERING
Applying changes from oplog

#2
SECONDARY
Read-only Replica



YVES WIENECKE

REPLICA SET MEMBER STATES

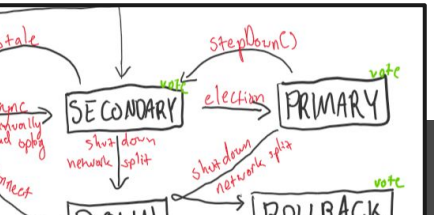


#2
SECONDARY
Read-only Replica

[stepDown(),
reconfig()]

[election]

#1
PRIMARY
Read/Write Node



YVES WIENECKE

REPLICA SET MEMBER STATES

[Stale data]

#2
SECONDARY
Read-only Replica

[stepDown()]

[election]

#1
PRIMARY
Read/Write Node

#3
RECOVERING
Applying changes from oplog

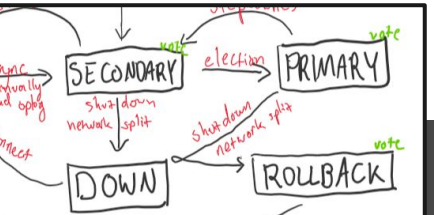
[Shutdown, network split]

#8
DOWN
Unreachable Node

[Shutdown, network split]

[prev primary is reachable again]

#9
ROLLBACK
Old Primary Node



YVES WIENECKE

REPLICA SET MEMBER STATES

#3
RECOVERING
Applying changes from oplog

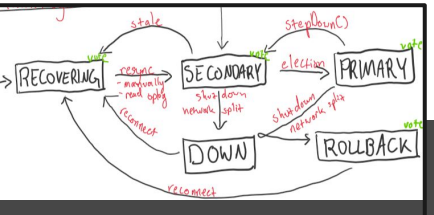
[old oplog]

#8
DOWN
Unreachable Node

[conflicting oplog]

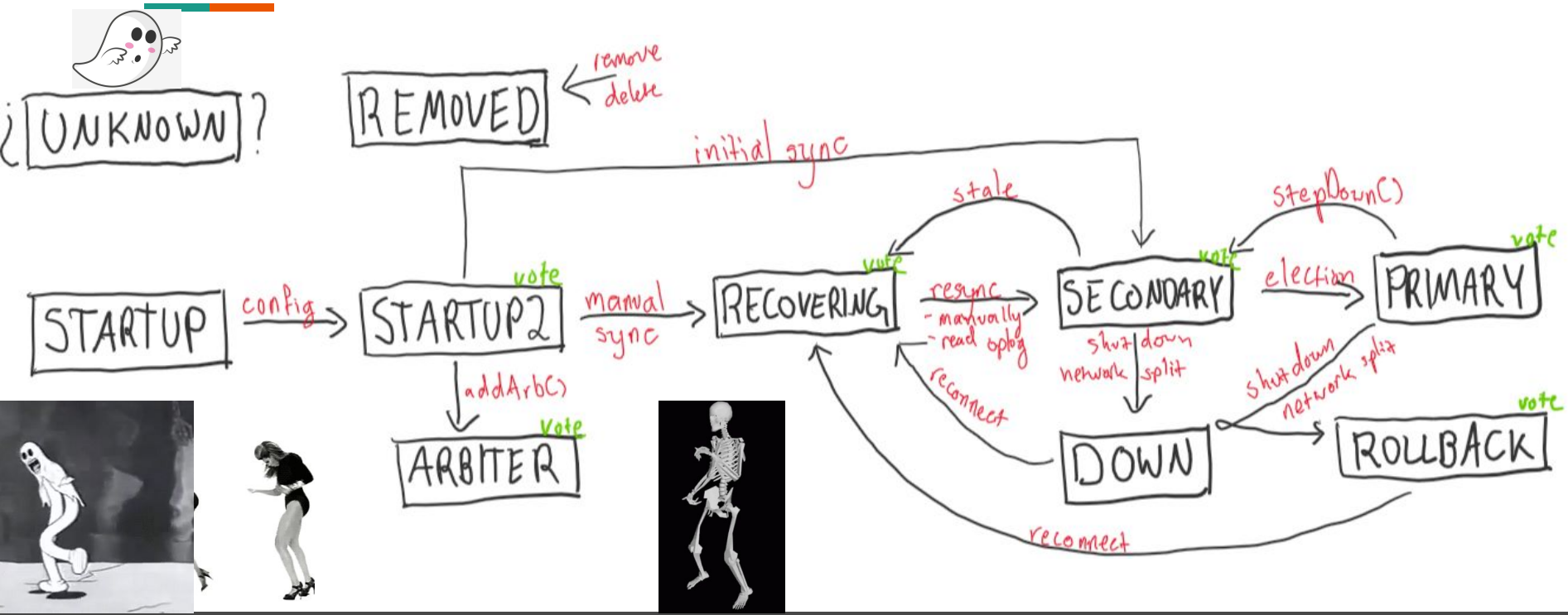
#9
ROLLBACK
Old Primary Node

[resolve conflicts]



YVES WIENECKE

REPLICA SET MEMBER STATES



YVES WIENECKE

Server Selection Algorithm

MongoDB drivers use this read preference algorithm to decide which replica set member is selected for read operations or, if there are multiple *mongos* instances (interfaces between client and sharded cluster), which *mongos* (and thereby cluster) to use.

Design goals for the algorithm where such that the server is selected in a way that is...

1. **predictable:** the behavior should not change
2. **resilient:** able to "adapt to topology changes without raising errors or requiring manual reconfiguration"
3. **low-latency**: if previous characteristics are equal, faster responses to queries/writes are preferred

Server Selection: https://docs.mongodb.com/manual/core/read-preference-mechanics/
Design Goals: https://github.com/mongodb/specifications/blob/master/source/server-selection/server-selection.rst

Michelle Duer

Server Selection Algorithm :: Replica Sets

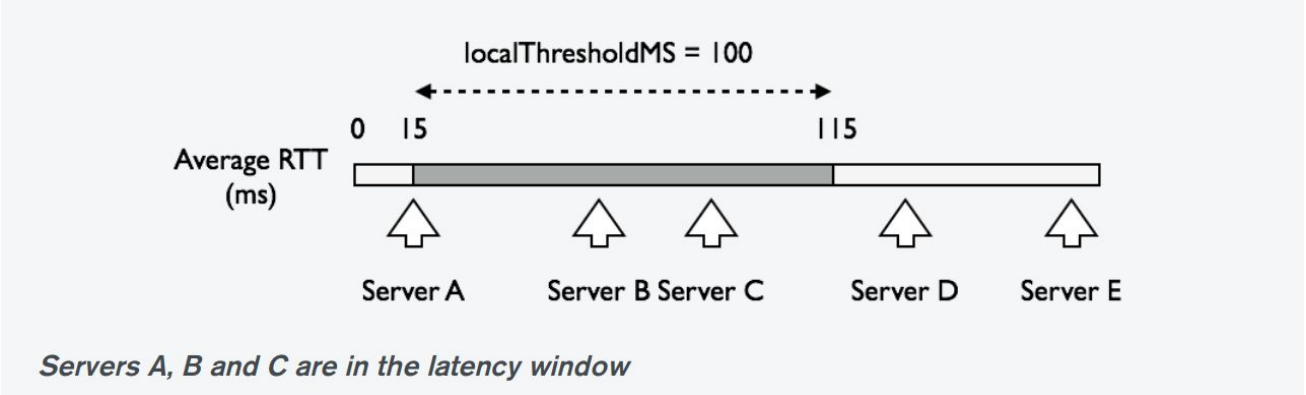
Read Preference	Selection Process
primary	Default setting where the driver selects the primary
secondary **nearest**	1. Driver selects eligible secondary members (where **nearest** also includes primary as eligible). 2. If the list is not empty, the driver narrows the list by finding the *closest* member with the lowest avg network round-trip time = avg*RTT*. Then a latency window is calculated by adding the avg*RTT* to the ping time (defaulted to 15 ms). All of the members that fall into the latency window are eligible. 3. A server is randomly selected from the eligible members whose measurements fell within the latency window.
primaryPreferred	The primary is selected first, and if unavailable, the **secondary** preference is followed.
secondaryPreferred	The **secondary** preference is followed, but if the list is empty, the primary is selected.

Read Preference: https://docs.mongodb.com/manual/core/read-preference-mechanics/
localPingTresholdMs: https://docs.mongodb.com/manual/reference/configuration-options/#replication.localPingThresholdMs

Michelle Duer

Server Selection Algorithm :: Latency Window

RTT (avg network round-trip time) = 15ms **localThresholdMS** (aka ping time) = 100ms
latency window is between 15-115 ms



If these were shard clusters instead, the latency window serves an additional function of providing the driver with the targeted shards for load balancing.

Image: https://www.mongodb.com/blog/post/server-selection-next-generation-mongodb-drivers

Michelle Duer

Server Selection Algorithm :: Sharded Clusters

The algorithm works the same for sharded clusters, except additional options of **maxStalenessSeconds** and **tags** can be applied to further customize read preference for shards.

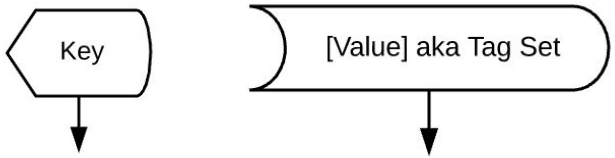
maxStalenessSeconds: setting this value restricts shards that have fallen behind from being selected due to network congestion, low disk throughput, long-running operations, etc

Tag: a {key:value} pair describing a replica set or shard's characteristic for read preference. This characteristic can be defined by the user. The value is a **tag set** consisting of an array of values that could describe the cluster.

Server Selection: https://docs.mongodb.com/manual/core/read-preference-mechanics/
maxStalenessSeconds: https://docs.mongodb.com/manual/core/read-preference/#replica-set-read-preference-max-staleness
Tag definition: https://github.com/mongodb/specifications/blob/master/source/server-selection/server-selection.rst#id20

Server Selection Algorithm :: Tag Option

Example Tags



{ “computer science”: [“CS”, “software engineering”, “programming”, “FAB”] }
{ “southeast buildings”: [“FAB”, “EB”, “Ondine”, “University Pointe”, “Art Building”]}

Example Read Preference:

```
db.collections.find({}).readPref(
        “secondary”, [
                { “computer science”: “FAB” },          // eligible servers have “computer science”:”FAB” tag, but if none exists
                { “southeast buildings”: “FAB” },       // find eligible servers with “southeast buildings”:”FAB” tag
                {}                                      // otherwise, choose any eligible server (no tag match)
        ]
)
```

Tag overview, model and operations: https://www.baeldung.com/mongodb-tagging

Michelle Duer

Server Selection Algorithm :: Client Configuration

Examples:

localThresholdMS can be changed from default of 15 ms for different latency window sizes

serverSelectionTimeoutMS defines a default server selection block for 30,000 ms (½ minute). This default is to allow for primary election completion with the hope of decreasing the default value as the election speeds improve.

heartbeatFrequencyMS is configured at a minimum of 500 ms intervals between server checks to require continuous and necessary monitoring (to update RTT value, server types, tags, discover new secondaries, etc).

smallestMaxStalenessSeconds requires a minimum of 90 seconds, to provide staleness estimates on secondaries

Configuration: https://github.com/mongodb/specifications/blob/master/source/server-selection/server-selection.rst
Heartbeat:
https://github.com/mongodb/specifications/blob/master/source/server-discovery-and-monitoring/server-discovery-and-monitoring.rst

Conclusion :: Is Customization a Good Thing?

One of the big appeals of MongoDB's replication approach is that it is so customizable -- layers upon layers of tweaking that can optimize the replicas and shards to use cases. Points of concern:

- **Human error** :: configuration settings could be changed without considering potential overlap in different layers (e.g. drivers, shards, replica sets, server nodes)
- **Discrepancies in documentation** :: election time metric example

  "The median time before a cluster elects a new primary should not typically exceed 12 seconds, assuming default replica configuration settings. This includes time required to mark the primary as unavailable and call and complete an election. You can tune this time period by modifying the settings.electionTimeoutMillis replication configuration option."

Median Election (Replica Set Elections): https://docs.mongodb.com/manual/core/replica-set-elections/index.html

Michelle Duer

Conclusion :: Is Customization a Good Thing?

“serverSelectionTimeoutMS

This defines how long to block for server selection before throwing an exception. The default is 30,000 (milliseconds). It MUST be configurable at the client level. It MUST NOT be configurable at the level of a database object, collection object, or at the level of an individual query.

This default value was chosen to be sufficient for a typical server primary election to complete. As the server improves the speed of elections, this number may be revised downward.

Users that can tolerate long delays for server selection when the topology is in flux can set this higher. Users that want to "fail fast" when the topology is in flux can set this to a small number.”

serverSelectionTimeoutMS:
https://github.com/mongodb/specifications/blob/master/source/server-selection/server-selection.rst#serverselectiontimeoutms

Conclusion :: Customization != Reliability

- ➔ Should a user in charge of driver configuration assume a safe maximum of completed election within ½ a minute?
- ➔ Should the replica set configuration **settings.electionTimeoutMillis** be decreased to 10 seconds, lower than the stated median of 12 seconds?
- ➔ Consider that both **settings.electionTimeoutMillis** and **serverSelectionTimeoutMS** can be modified to smaller or bigger values. In this way, the configurations start compounding on one another.

A few more notes of observation:

- ➔ The default value for **settings.electionTimeoutMillis is 10 seconds**
- ➔ There is a stated caveat that the settings are dependent on cluster architecture and network latency

Default electionTimeoutMillis: https://docs.mongodb.com/manual/reference/replica-configuration/#rsconf.settings.electionTimeoutMillis
Caveat: https://docs.mongodb.com/manual/core/replica-set-elections/index.html

Conclusion :: Undecided

As a team, we have varying opinions on how we feel about MongoDB after our research.

The customizations can be awesome, but require a lot of architecting and potential benchmarking to really understand the results you will be achieving. Very careful reading of the documentation is necessary.

There is some agreement that MongoDB strongly appeals to a broader market by claiming NoSQL features as well as ACID guarantees and partition tolerance, but understanding the CAP theorem begs some skepticism on these statements. To their credit, they do claim and maintain a large market share of NoSQL clients, which is in their favor.

serverSelectionTimeoutMS:
https://github.com/mongodb/specifications/blob/master/source/server-selection/server-selection.rst#serverselectiontimeoutms

Questions?