Exploring the replication and sharding in MongoDB

Apr-26-2017

Pythian

About me

- Master degree, Software Engineering
- Working with databases since 2007
- MySQL, LAMP, Linux since 2010
- Pythian OSDB managed services since 2014
- Lead Database Consultant since 2016
- C100DBA: MongoDB certified DBA

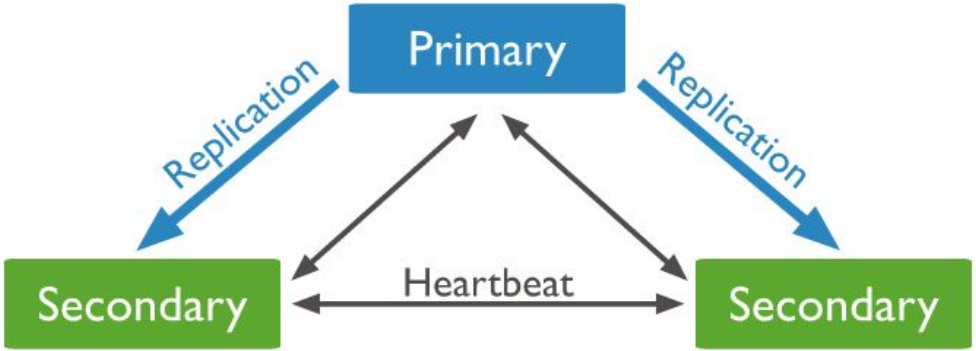
 https://mk.linkedin.com/in/igorle  @igorLE

Overview

- What is replica set, how replication works, replication concepts
- Replica set features, deployment architectures
- Vertical vs Horizontal scaling
- What is a sharded cluster in MongoDB
- Cluster components - shards, config servers, mongos
- Shard keys and chunks
- Hashed vs range based sharding
- QA

Replication

- Group of mongod processes that maintain the same data set
- Redundancy and high availability
- Increased read capacity

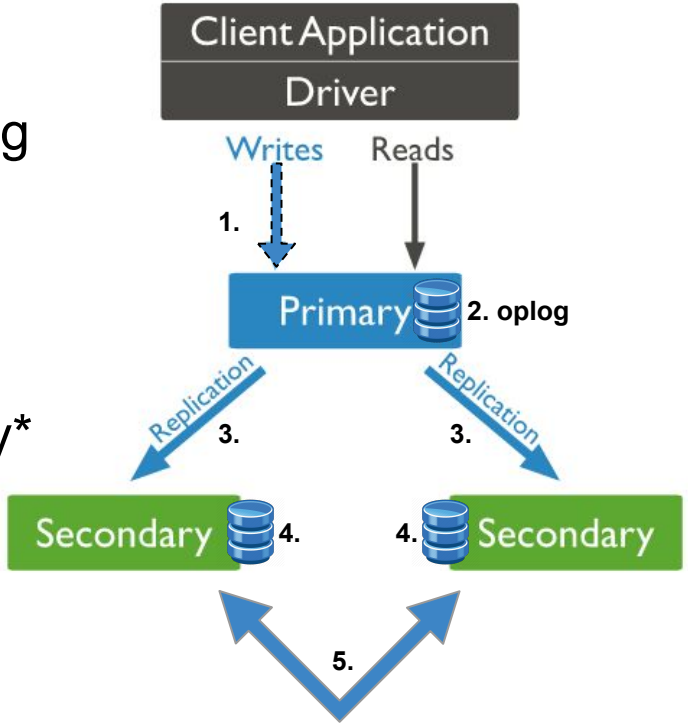


Replication concept

1. Write operations go to the Primary node
2. All changes are recorded into operations log
3. Asynchronous replication to Secondary
4. Secondaries copy the Primary oplog
5. Secondary can use sync source Secondary*

Automatic failover on Primary failure

*settings.chainingAllowed (true by default)



Replica set oplog

- Special capped collection that keeps a rolling record of all operations that modify the data stored in the databases
- Idempotent
- Default oplog size

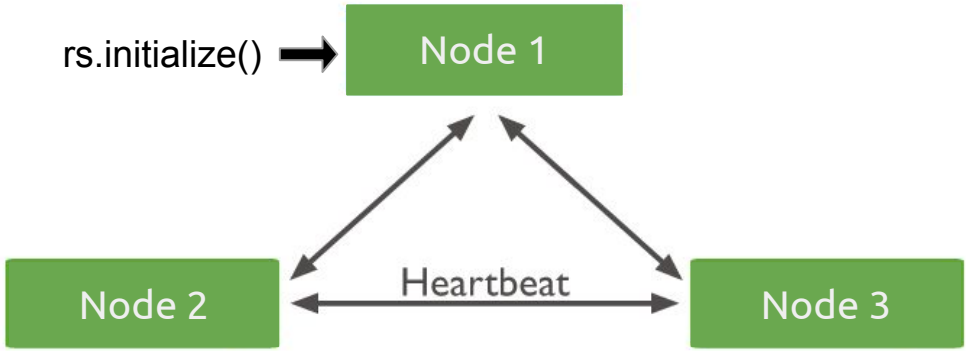
For Unix and Windows systems

Storage Engine	Default Oplog Size	Lower Bound	Upper Bound
In-memory	5% of physical memory	50MB	50GB
WiredTiger	5% of free disk space	990MB	50GB
MMAPv1	5% of free disk space	990MB	50GB

Deployment

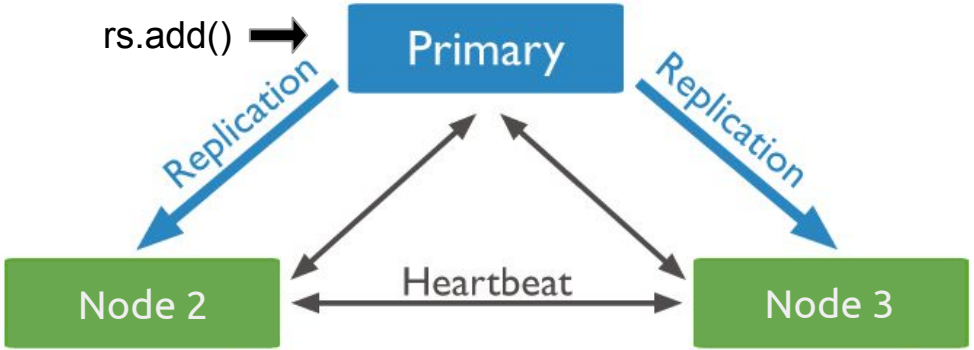
- Start each server with config options for replSet

  /usr/bin/mongod --replSet "myRepl"
- Initiate the replica set on one node - rs.initialize()
- Verify the configuration - rs.conf()



Deployment

- Add the rest of the nodes - rs.add() on the Primary node
  rs.add(“node2:27017”) , rs.add(“node3:27017”)
- Check the status of the replica set - rs.status()

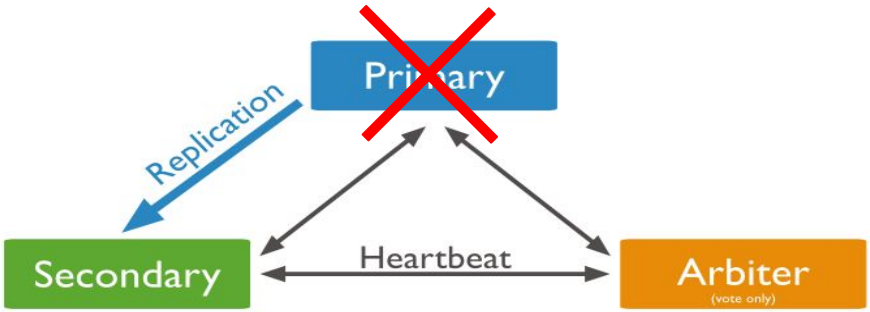


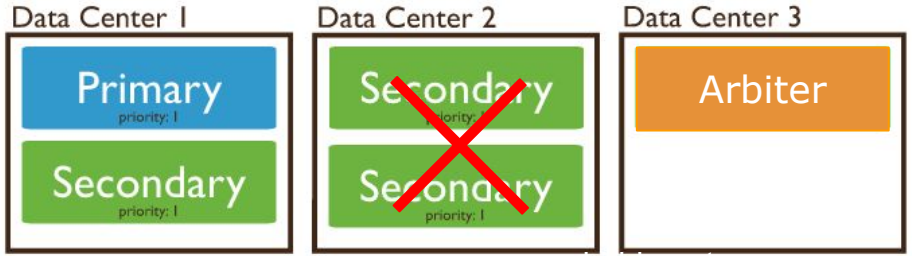
Configuration options

- 50 members per replica set (7 voting members)
- Arbiter node
- Priority 0 node
- Hidden node
- Delayed node
- Write concern
- Read preference

Arbiter node

- Does not hold copy of data
- Votes in elections

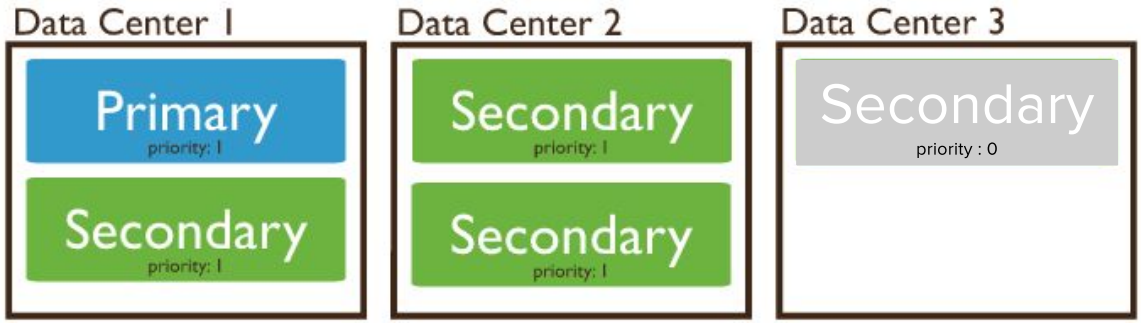




Priority 0 node

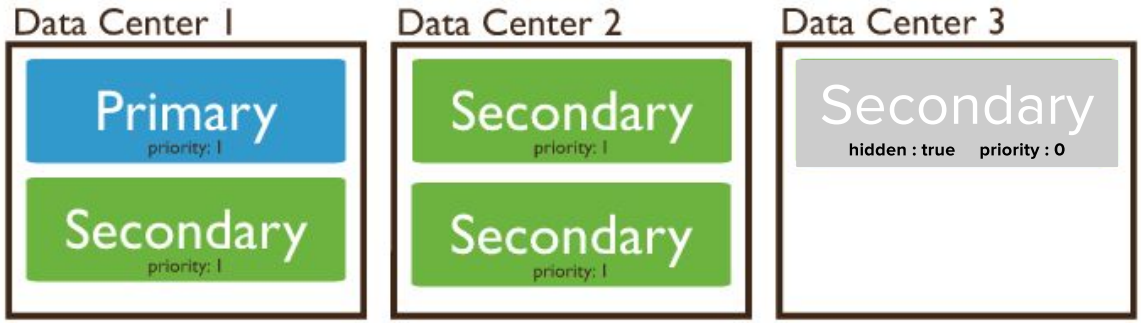
Priority - floating point (i.e. decimal) number between 0 and 1000

- Never becomes primary
- Visible to application
- Node with highest priority is eventually elected as Primary



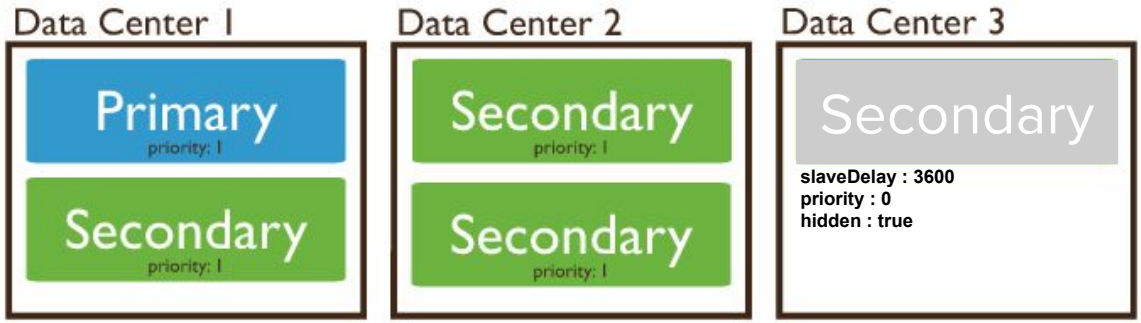
Hidden node

- Never becomes primary
- Not visible to application
- Use cases
  - reporting
  - backups



Delayed node

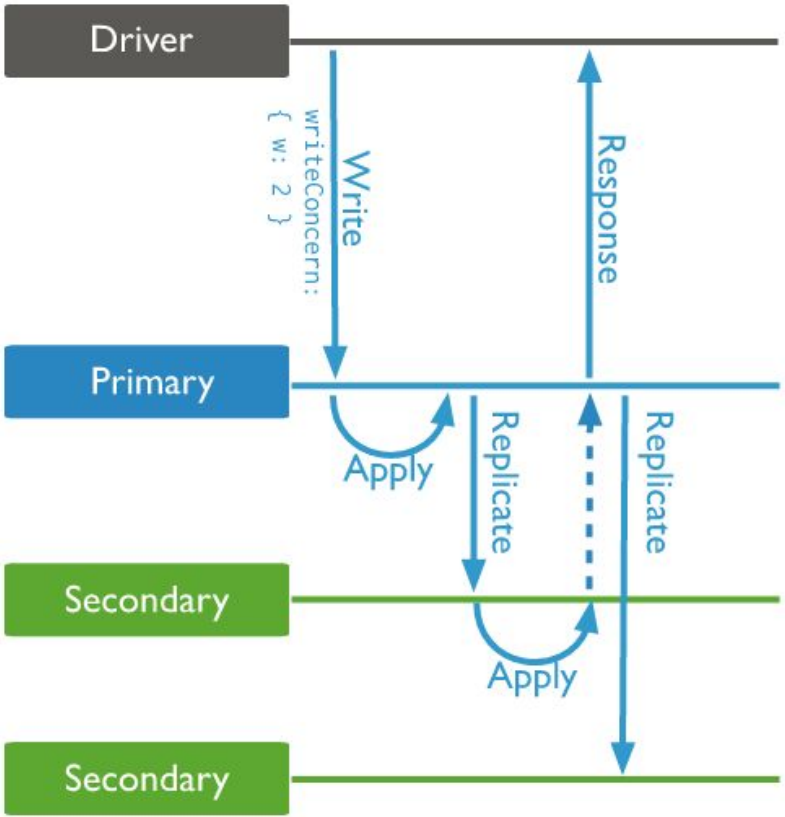
- Must be priority 0 member
- Should be hidden member (not mandatory)
- Has votes 1 by default
- Considerations (oplog size)
- Mainly used for backups



Write concern

{ w: <value>, j: <boolean>, wtimeout: <number> }

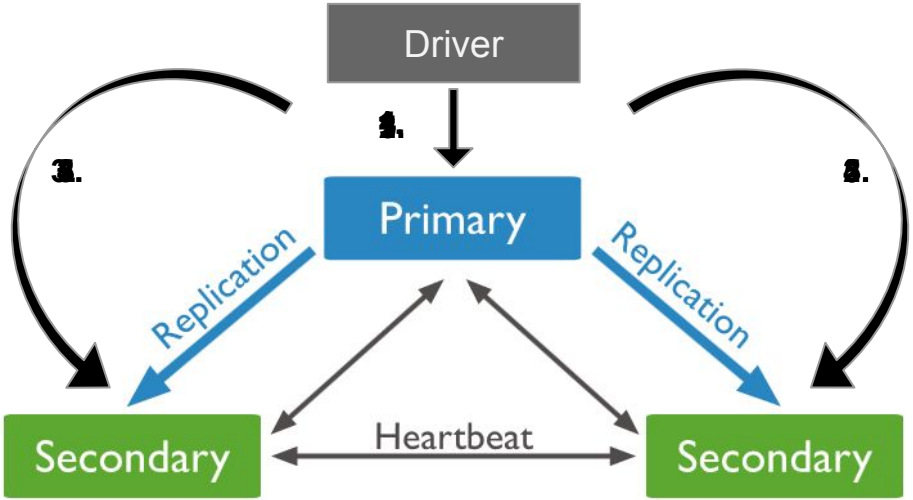
- w - number of mongod instances that acknowledged the write operation
- j - acknowledgement that the write operation has been written to the journal
- wtimeout - time limit to prevent write operations from blocking indefinitely



Read preference

How read operations are routed to replica set members

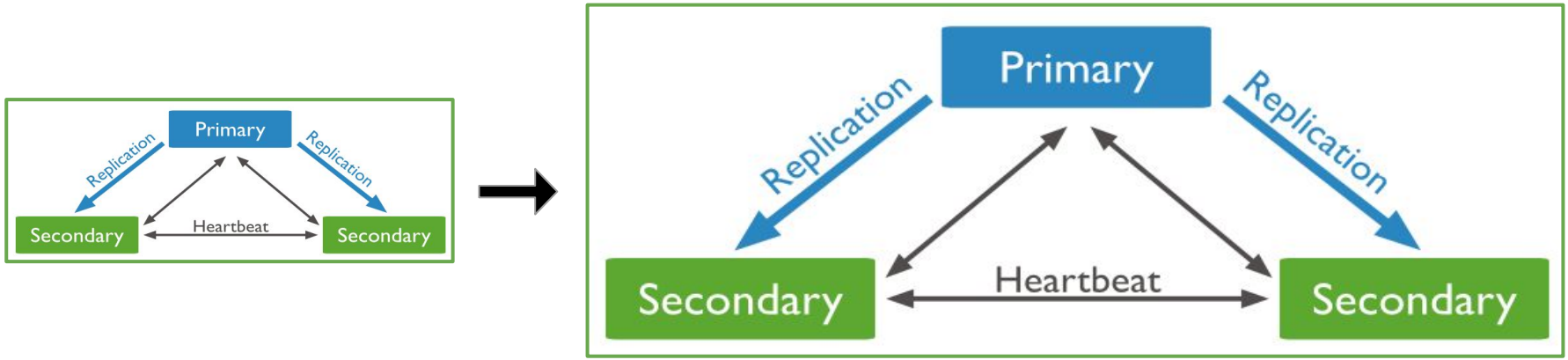
1. primary (by default)
2. primaryPreferred
3. secondary
4. secondaryPreferred
5. nearest (least network latency)



MongoDB 3.4 maxStalenessSeconds (>= 90 seconds)

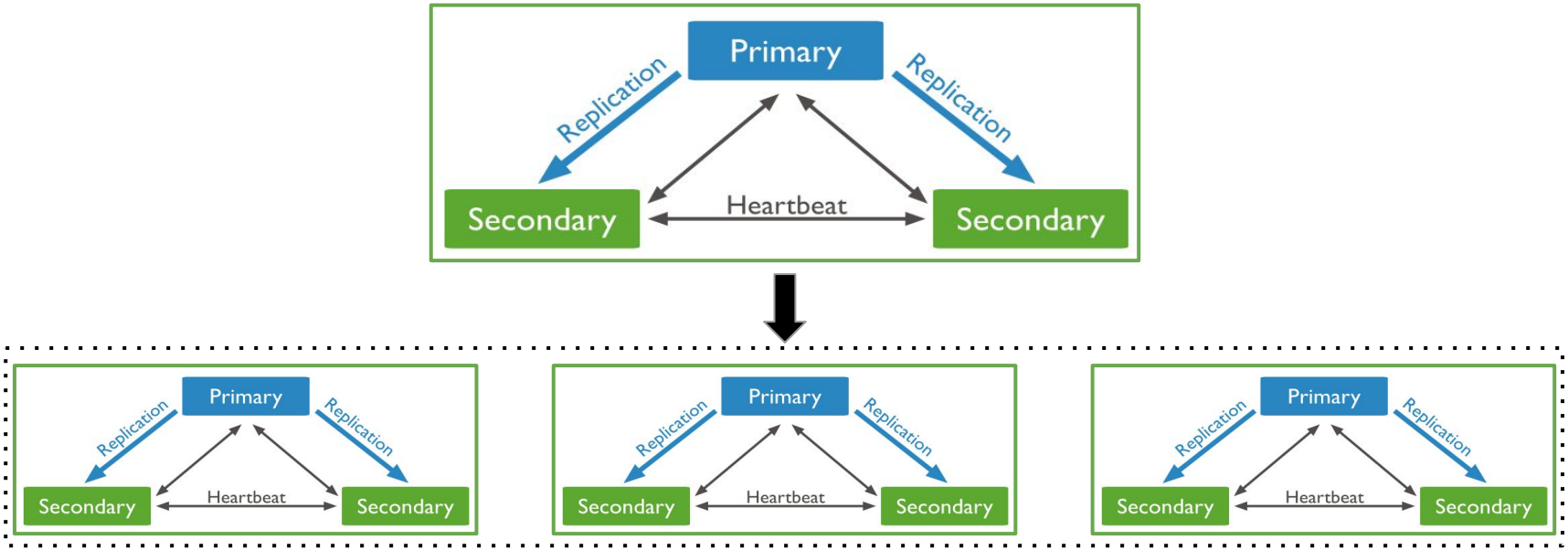
Scaling (vertical)

- CPU
- RAM
- DISK



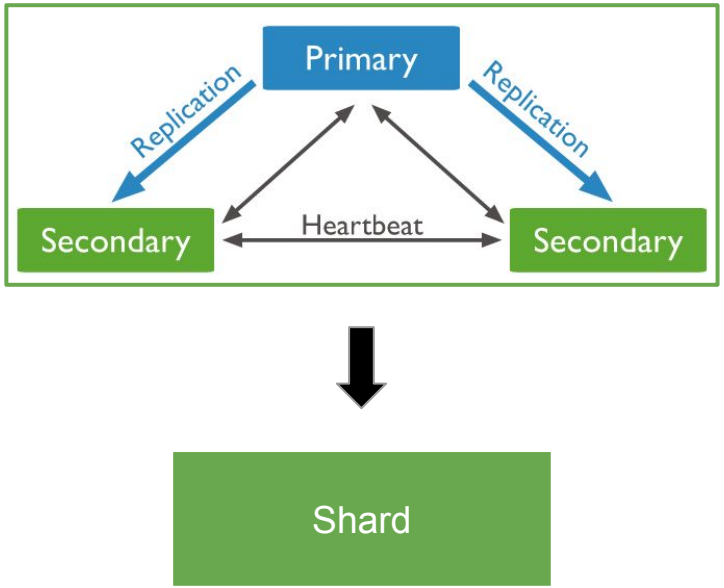
Scaling (horizontal) - Sharding

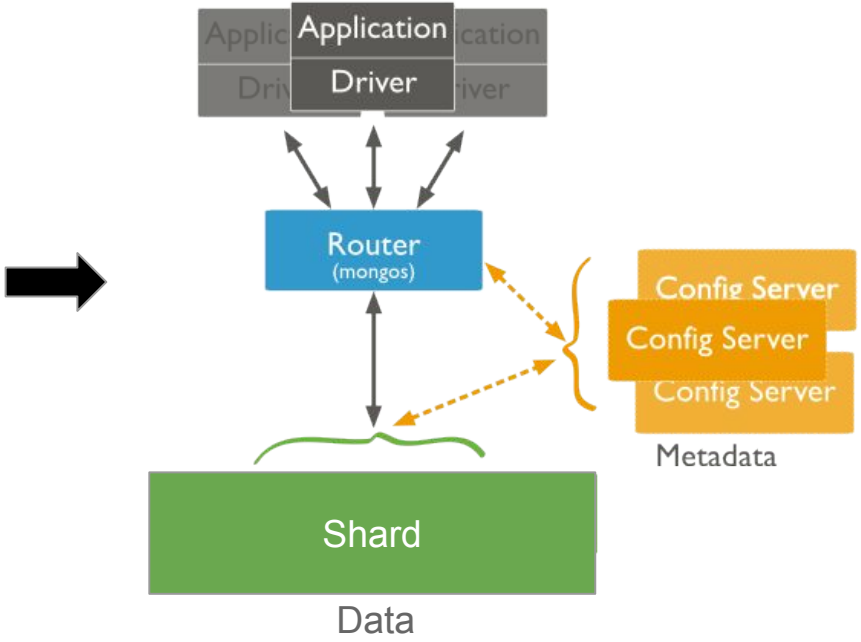
- Method for distributing data across multiple machines
- Splitting data across multiple horizontal partitions



Pythian
love your data

Sharding with MongoDB





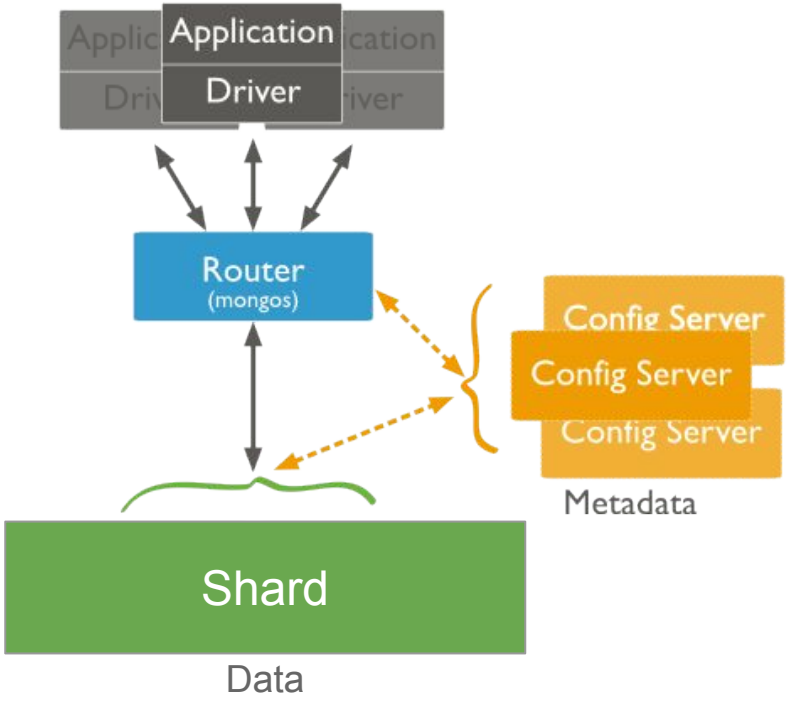
Sharding with MongoDB

- Shard/Replica set

  (subset of the sharded data)
- Config servers

  (metadata and config settings)
- mongos

  (query router, cluster interface)



Sharding with MongoDB

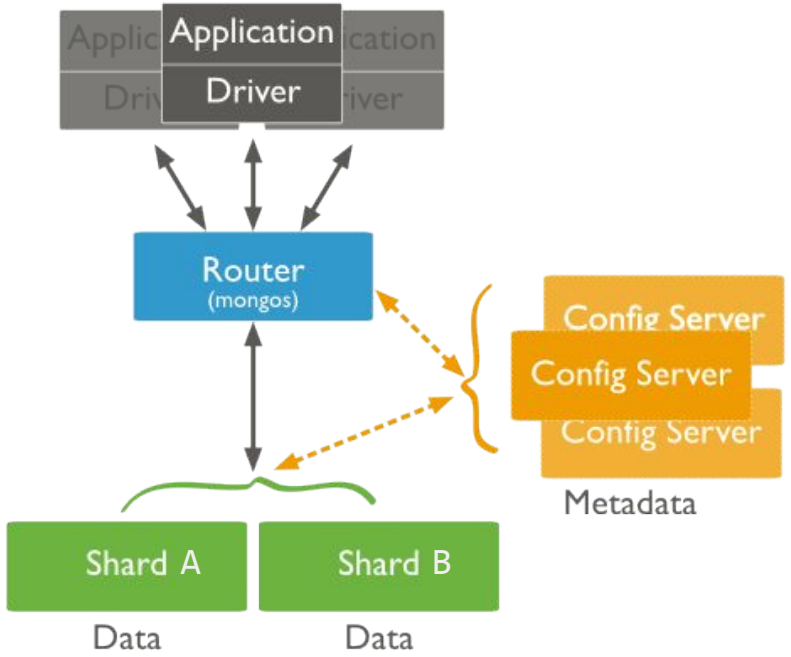
- Shard/Replica set

  (subset of the sharded data)
- Config servers

  (metadata and config settings)
- mongos

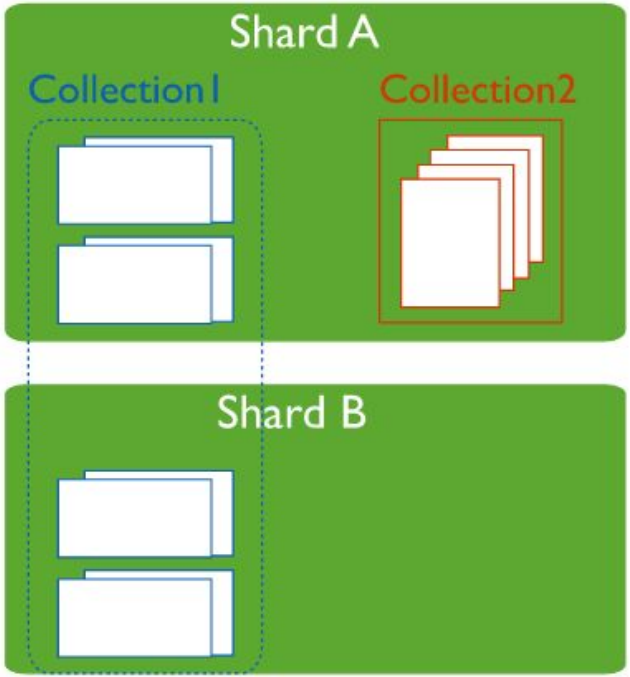
  (query router, cluster interface)

  sh.addShard("shardName")



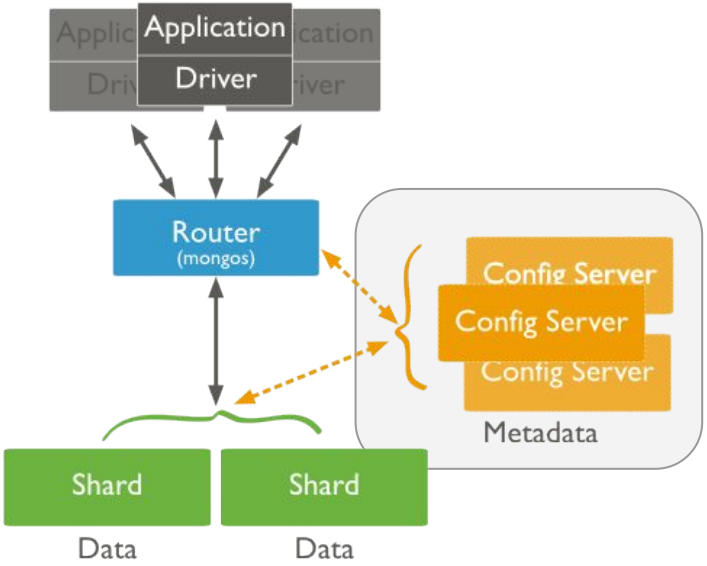
Shards

- Contains subset of sharded data
- Replica set for redundancy and HA
- Primary shard
- Non sharded collections
- --shardsvr in config file (port 27018)



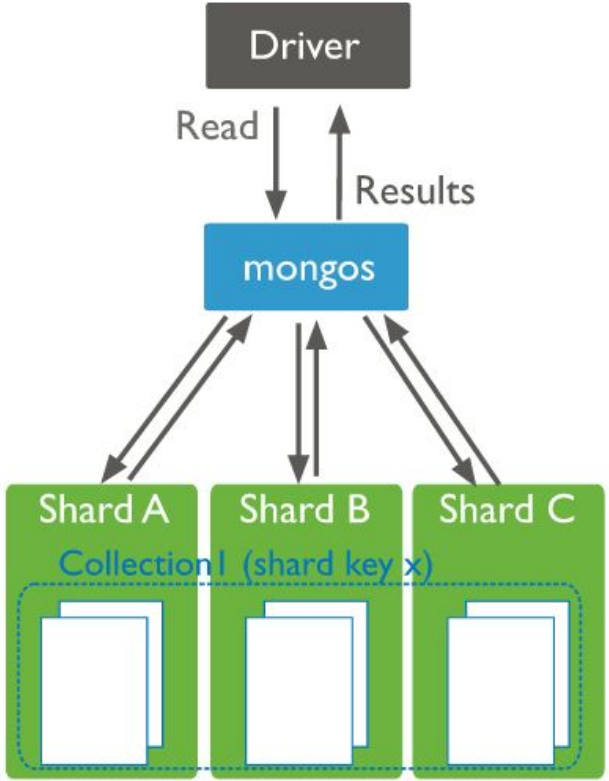
Config servers

- Config servers as replica set only (>= 3.4)
- Stores the metadata for sharded cluster in *config* database
- Authentication configuration information in *admin* database
- Holds balancer on Primary node (>= 3.4)
- --configsvr in config file (port 27019)



mongos

- Caching metadata from config servers
- Routes queries to shards
- No persistent state
- Updates cache on metadata changes
- Holds balancer (mongodb <= 3.2)
- mongos version 3.4 can not connect to earlier mongod version



Sharding collection

- Enable sharding on database

```
sh.enableSharding("users")
```

- Shard collection

```
sh.shardCollection("users.history", { user_id : 1 } )
```

- Shard key - indexed key that exists in every document
  - range based

```
sh.shardCollection("users.history", { user_id : 1 } )
```

  - hashed based

```
sh.shardCollection( "users.history", { user_id : "hashed" } )
```

Shard keys and chunks

Choosing Shard key

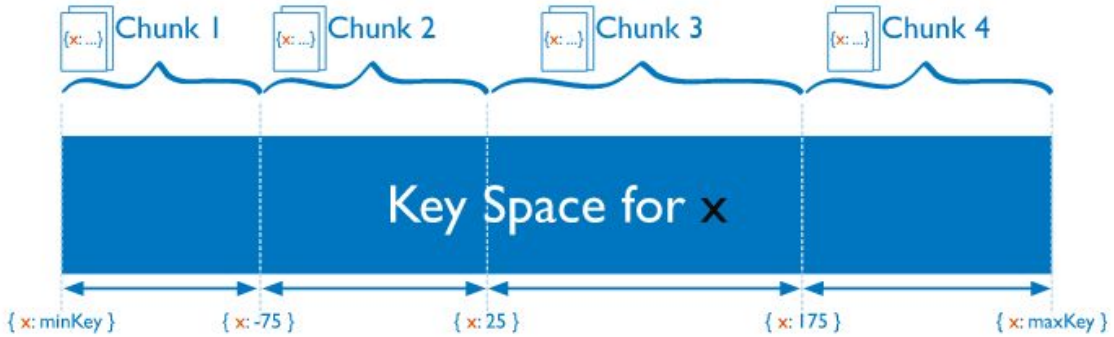
- Large Shard Key Cardinality
- Low Shard Key Frequency
- Non-Monotonically changing shard keys

Chunks

- A contiguous range of shard key values within a particular shard
- Inclusive lower and exclusive upper range based on shard key
- Default size of 64MB

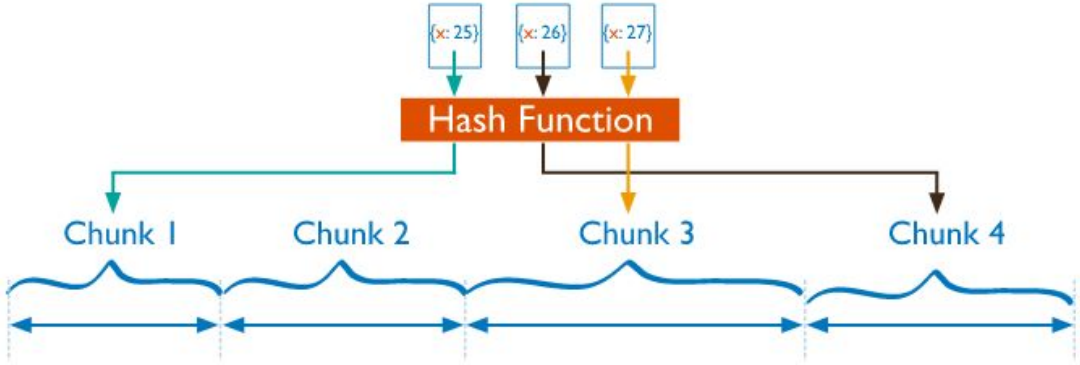
Ranged sharding

- Dividing data into contiguous ranges determined by the shard key values
- Documents with “close” shard key values are likely to be in the same chunk or shard
- Query Isolation - more likely to target single shard



Hashed sharding

- Uses a hashed index of a single field as the shard key to partition data
- More even data distribution at the cost of reducing Query Isolation
- Applications do not need to compute hashes
- Keys that change monotonically like ObjectId or timestamps are ideal



Shard keys limitations

- Must be ascending indexed key or indexed compound keys that exists in every document in the collection
- Cannot be multikey index, a text index or a geospatial index
- Cannot exceed 512 bytes
- Immutable key - can not be updated/changed
- Update operations that affect a single document must include the shard key or the _id field
- No option for sharding if unique indexes on other fields exist
- No option for second unique index if the shard key is unique index

Summary

- Replica set with odd number of voting members
- Hidden or Delayed member for dedicated functions (backups …)
- Dataset fits into single server, keep unsharded deployment
- Horizontal scaling - shards running as replica sets
- Shard keys are immutable with max size of 512 bytes
- Shard keys must exists in every document in the collection
- Ranged sharding may not distribute the data evenly
- Hashed sharding distributes the data randomly

Questions?

We’re Hiring!

https://www.pythian.com/careers/