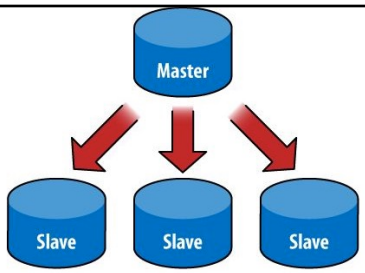
REPLICATION

100

100

Master-Slave Replication



- Start the master

```
$ mkdir -p ~/dbs/master
$ ./mongod --dbpath ~/dbs/master --port 10000
--master
```

- Start a slave

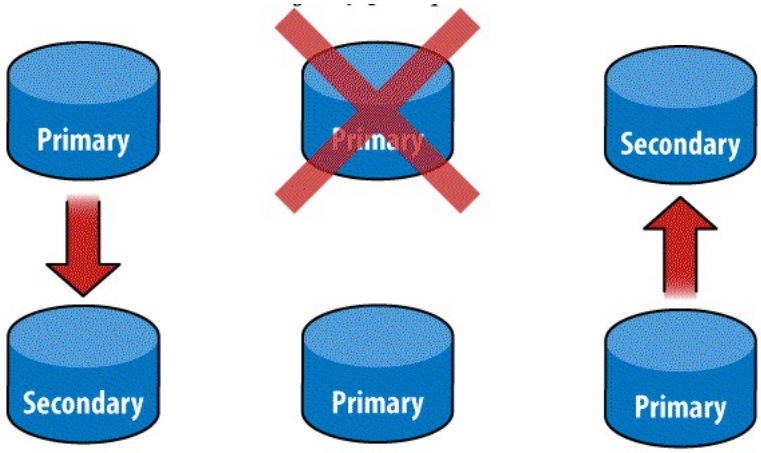
```
$ mkdir -p ~/dbs/slave
$ ./mongod --dbpath ~/dbs/slave --port 10001
--slave --source localhost:10000
```

101

101

Replica Sets

- Replication with automatic failover



102

102

Replica Sets

- Start a server in the replica set “blort”

```
$ mkdir -p ~/dbs/node1 ~/dbs/node2
$ ./mongod --dbpath ~/dbs/node1 --port 10001
           --replSet blort/morton:10002
```

- Start a second server

```
$ ./mongod --dbpath ~/dbs/node2 --port 10002
           --replSet blort/morton:10001
```

- Start a third server

```
$ ./mongod --dbpath ~/dbs/node3 --port 10003
           --replSet blort/morton:10001
$ ./mongod --dbpath ~/dbs/node3 --port 10003
        --replSet blort/morton:10001,morton:10002
```

103

103

Initializing a Replica Set

```
$ ./mongo morton:10001/admin
MongoDB shell version: 1.5.3
connecting to localhost:10001/admin
> db.runCommand({"replSetInitiate" : {
 "_id" : "blort",
 "members" : [
    {
      "_id" : 1,
      "host" : "morton:10001”
    },
    {
      "_id" : 2,
      "host" : "morton:10002”
    }
  ]
}})
```

104

104

Types of Nodes

- Standard
  - Replicate data
  - May become primary
  - Participates in voting for primary
- Passive
  - Replicate data
  - Participates in voting
- Arbiter
  - Voting only

105

105

Node Priority

- Priority = 0 ⇒ passive node
- Priority > 0 ⇒ primary based on priority
  - Freshness of data breaks ties

```
members.push({ "_id" : 3,
               "host" : "morton:10003",
               "priority" : 40  });

members.push({ "_id" : 4,
               "host" : "morton:10004",
               "arbiterOnly" : true ... });
```

106

106

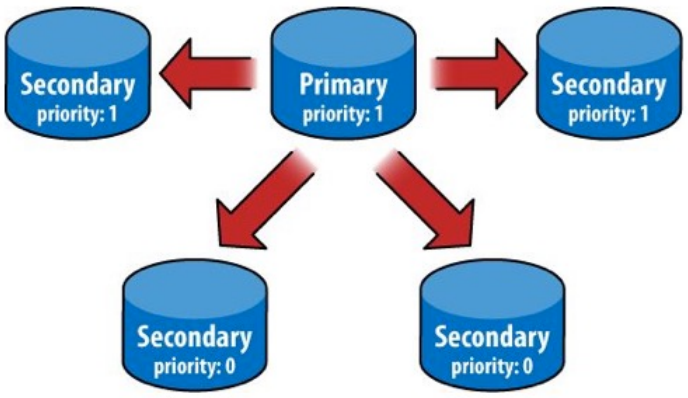
Failover and Primary Election

- Primary: track nodes using heartbeat
  - No quorum ⇒ fall back to secondary
  - Prevent *split brain* (network partition)
- Primary assumed most up-to-date
  - Recovery: nodes *resync* with new primary
  - Later ops rolled back, up-to-date copy from primary

107

107

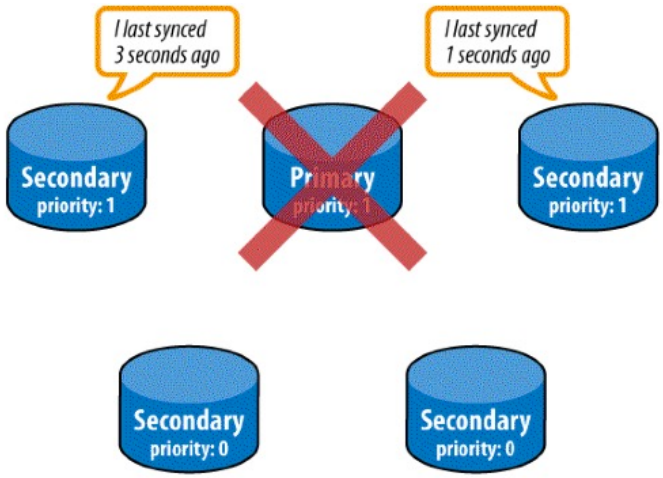
Failover and Primary Election



108

108

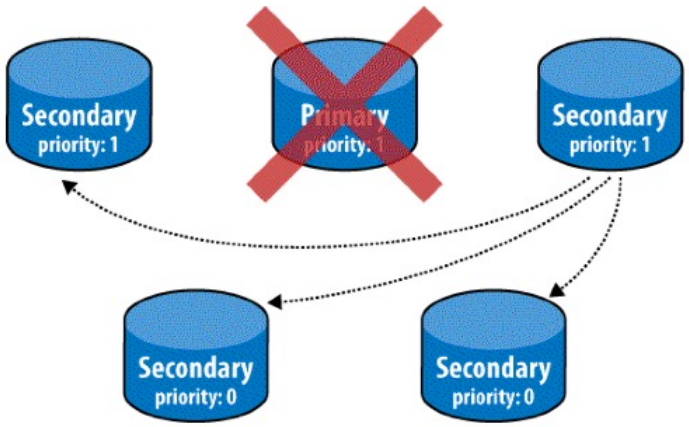
Failover and Primary Election



109

109

Failover and Primary Election



110

110

Slave Use Cases

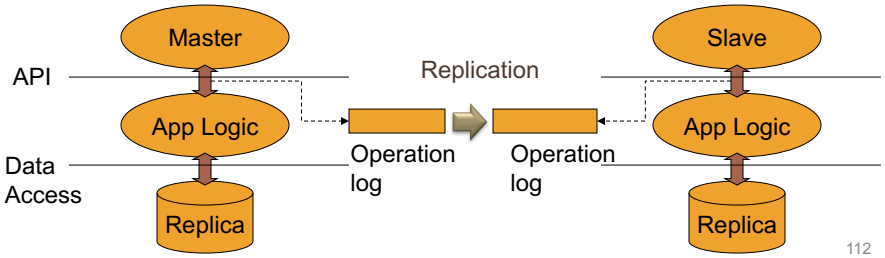
- Backing up data
- Read scaling
  - Send queries directly to slave
  - `slaveOkay` query option
- Off-loading data processing
  - Run slave with both `--slave` and `--master`
  - Distinguish locally updated & mirrored data

111

111

How Replication is done

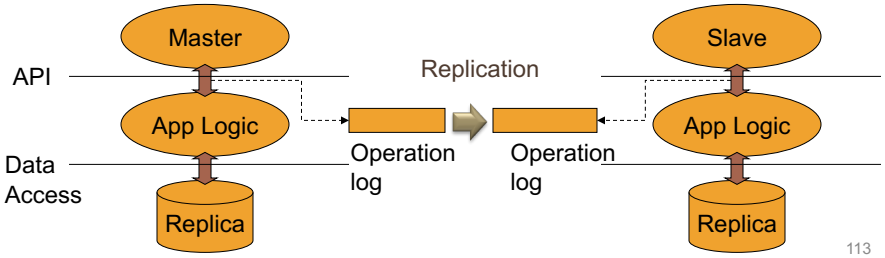
- Oplog: operation log
  - Logs include timestamps
  - Ops transformed to be idempotent
    e.g. x++ ⇒ x=3
  - Stored in capped collection



112

How Replication is done

- Syncing:
  - Initially slave copies all data
  - Thereafter queries oplog
  - Out of sync: slave too far behind
    - Avoid: ensure oplog is large enough



113

Blocking for Replication

- Provide guarantee about replication

```
db.runCommand({getLastError: 1, w: N});
```

  - Wait for N replicas to ack (incl master)
  - N < 2 $\Rightarrow$ don't block
  - N = 2 $\Rightarrow$ block until one slave acks

- Tradeoff: reliability vs performance
  - Pick N =2 or N = 3 for important operations

114

114

SHARDING

115

115

Sharding

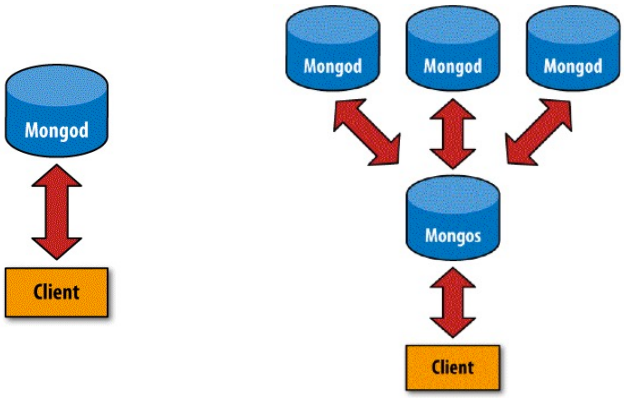
- Manual sharding
  - Connections to several databases
  - Adding/removing nodes
  - Redistributing data
- Autosharding
  - Automatic data splitting & distribution
  - Handled by cluster

116

116

Autosharding in MongoDB

- Break up data into chunks
- Router (mongos) routes requests



117

117

Shard Keys

- Define shard key e.g. timestamp
- Partition data based on ranges
- Incrementing shard keys

  $[t_0,t_1), [t_1,t_2), \ldots, [t_k,\infty)$

  $\Rightarrow [t_0,t_1), [t_1,t_2), \ldots, [t_k,t_{k+1}), [t_{k+1},\infty)$
- Random shard keys
  - Uniformly distribute high write load
  - E.g. hash of timestamp
  - Similar to choosing keys

118

118

Shard Keys

- Suppose collection sharded on name key:
  - A-F, G-P, Q-Z
- Example operations

  `db.people.find({"name" : "Susan"})`
  - Send to Q-Z shard

  `db.people.find({"name" : {"$lt" : "L"}})`
  - Send to A-F, then G-P, responses to client

  `db.people.find().sort({"email" : 1})`
  - Query all shards, merge sort of results
  - Cursors for each shard, batched results to client

  `db.people.find({"email" : "joe@example.com"})`
  - Send to all shard

119

119

Setting up Sharding

- Run config server

```
$ mkdir -p ~/dbs/config
$ ./mongod --dbpath ~/dbs/config --port 20000
```

- Run router

```
$ ./mongos --port 30000 --configdb localhost:20000
```

- Run shard

```
$ mkdir -p ~/dbs/shard1
$ ./mongod --dbpath ~/dbs/shard1 --port 10000
```

- Connect shard to cluster

```
$ ./mongo localhost:30000/admin
> db.runCommand({addshard : "localhost:10000",
                 allowLocal : true})
```

120

Sharding Data

- Ex: Shard `bar` collection in `foo` database on `_id` key
- Enable sharding

```
db.runCommand({"enablesharding" : "foo"})
```

- Shard the collection

```
db.runCommand({"shardcollection" : "foo.bar",
               "key" : {"_id" : 1}})
```

- Run shard

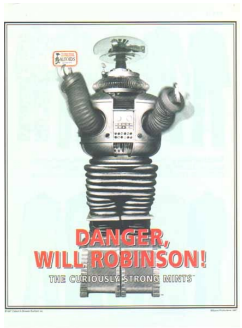
121

Robust Config

- Multiple config servers
  - Connected to router (mongos)

```
./mongos --configdb localhost:20001,
                    localhost:20002,
                    localhost:20003
```

  - Synchronize using 2PC



- Multiple routers
  - Ex: one router per app server

- Replicated shards

```
db.runCommand({"addshard" :
               "foo/prod.example.com:27017"})
```


122