MongoDB 101 for MySQL DBAs and Developers

Stéphane Combaudon
March 25th, 2015

Agenda

- Introduction – Why MongoDB?
- Basic CRUD
- Schema Design
- Aggregation Framework
- Replication
- Sharding

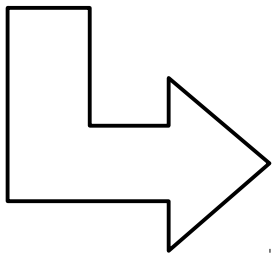
Typical MySQL issues (2007-ish)

- #1 - SQL is difficult to read and write
  - Easy to write queries, hard to write efficient queries
- #2 - Schema changes are hard with large tables
  - But modern apps require flexibility
- #3 - High Avail. and Scalability are complex topics
  - Not a one-size-fits-all option for HA
  - Limited support for sharding

What MongoDB offers - 1

- New developer friendly query language instead of SQL

```
SELECT * FROM people WHERE name = 'Joe'
```



```
db.people.find({name: 'Joe'})
```

- Then DBAs no longer understand queries, but developers do!

What MongoDB offers - 2

- Schemaless design
  - Obvious benefit: no need to change the schema at the db level as there is no schema
- No schema, really?
  - We'll see later that this has a lot of implications, not always for the good

What MongoDB offers - 3

- One HA option: replica sets
  - Asynchronous replication, automatic failover
- Automatic and transparent sharding
  - Sharding at the table level
  - Data is automatically spread over the shards
  - A query router is provided (mongos): sharding is transparent to the application

Wow, looks impressive

- It is! But
  - Still a young product – many bugs
  - Single node performance is not great
  - Schemaless approach brings lots of issues
  - MySQL has improved a lot since 2007
    - Execution of complex queries much better in 5.6+
    - Several good HA options: Galera, GTID, MHA
    - Scaling up is now a good option to avoid sharding

Agenda

- Introduction – Why MongoDB?
- Basic CRUD
- Schema Design
- Aggregation Framework
- Replication
- Sharding

Terminology

MongoDB	MySQL
• mongod: server	• mysqld
• mongo: command-line client	• mysql
• mongos: query router for sharded cluster	• N/A
• Database	• Database
• Collection	• Table
• Document	• Row

JSON

- JavaScript Object Notation
- Lightweight alternative to XML

```
{
  name: 'Joe',
  age: 30
}
```

- MongoDB documents stored as BSON
  - Binary JSON

MongoDB document

- Set of key/value pairs
- A value can be an array or another JSON doc

```
{
  name: 'Joe',
  age: 30,
  likes: ['movies, 'hiking', 'basketball'],
  address: {
    zipcode: 98765,
    city: 'XYZ'
  }
}
```

_id

- Unique identifier of a document
  - Can be set explicitly
  - If not, an “ObjectId” is automatically created for you

```
"_id" : ObjectId("523ef7bf8108101415e7d1d1")
```

- ObjectId is roughly similar to an auto_inc integer with MySQL

Inserting data

```
use test

db.people.insert({name: 'Joe', age: 30})
```

- No CREATE DATABASE, no CREATE TABLE
  - MongoDB automatically creates the objects whenever needed

Retrieving data – Simple edition

- `db.people.find({name: 'Joe'})`
  - Returns JSON documents where the value of the 'name' attribute is 'Joe'
  - MySQL: `SELECT * FROM people WHERE name = 'Joe'`

- `db.people.findOne({name: 'Joe'})`
  - To get only one document
  - MySQL: `SELECT * FROM people WHERE name = 'Joe' LIMIT 1`

Other operations

- Many options for the WHERE conditions
  - AND and OR, inequalities
  - Conditions on subdocuments
  - Conditions on arrays
- Updating, deleting, counting, sorting are also available
- Look at the documentation for details
  - http://docs.mongodb.org/manual/crud/

Indexing

- Indexes are supported
  - Very similar to indexes in MySQL
- Can be used to
  - Filter efficiently
  - Sort efficiently
  - Run index-only queries (covering index)

Agenda

- Introduction – Why MongoDB?
- Basic CRUD
- Schema Design
- Aggregation Framework
- Replication
- Sharding

What schemaless means

- Insert a document

```
db.people.insert({name: 'Joe'})
```

- If our app needs a new field, how to add it?

```
# Insert new document with new field
db.people.insert({name: 'Mike', age: 30})
# Update existing document
db.people.update({name: 'Joe'}, {$set: {age: 25}});
```

- No ALTER TABLE is needed
- Existing documents don't need to be changed

Drawbacks of schemaless

- The database will never enforce the schema
  - Schema checks are deferred to the app

```
# This is allowed, but probably not what you want
db.people.insert({name: 'Joe', country: 'US'})
db.people.insert({name: 'Mike', location: 'US'})
```

- The attribute names are no longer metadata, they are data
  - For a 1M record collection, you'll store the attribute names 1M times. Not very efficient

What non relational means

- Joins are not available, data is denormalized
- 'Embedding' can be used for 1:1 and some 1:N relations instead of joins

```
{
  _id: 123,
  title: 'A nice post'
  comments: [
    {                                                  # Comment #1
      user_id: 345,
      text: 'Thanks for the clarification!'
    },
    {                                                  # Comment #2
      user_id: 456,
      text: 'Not happy with what you wrote'
    }
  ]
}
```

Drawbacks of embedding

- A document is limited to 16MB
  - Can be reached for 1:N relations when N is large
- Embedding does not work with M:N relations
- Not all data access patterns are equal!
- In the previous example
  - Finding the comments of a given post is easy
  - Finding the total number of comments is hard
    - Need to parse the post collection and sum the number of comments for each post

How flexible is MongoDB?

- Schemaless => flexibility to structure collections
- Non relational => working with relations is hard
  - You must know your access patterns before modeling your data
  - What if your access patterns change?

Agenda

- Introduction – Why MongoDB?
- Basic CRUD
- Schema Design
- Aggregation Framework
- Replication
- Sharding

Overview

- Aggregation
  - GROUP BY, SUM(), etc
- Using Map-Reduce is another option
  - But the AF is easier to use
  - And also faster
  - Some complex queries require Map-Reduce

Simple example

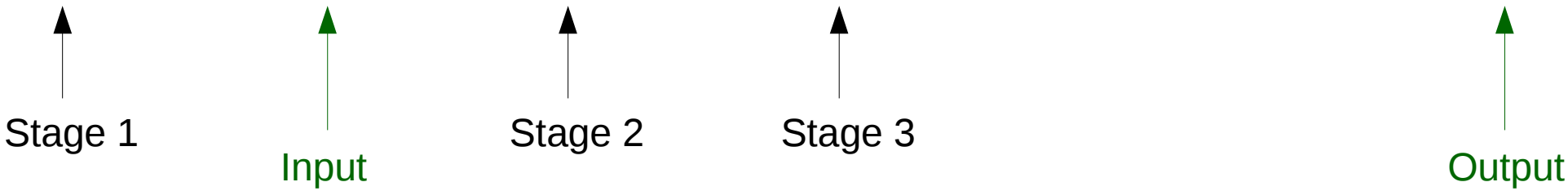
```
db.people.aggregate({$match:{name:{$gte:'M'}}},
{$group:{_id:"$name",total:{$sum:1}}})
```

- Uh, oh! Is that a “simple” example?
  - We'll clarify shortly!
- MySQL equivalent
  - `SELECT name,count(*) AS total FROM people WHERE name >= 'M%' GROUP BY name`
  - So yes, that's pretty simple!

Understanding the AF

- Documents are modified through pipelines
  - Think Unix pipelines

```
grep oo /etc/passwd | sort -rn | awk -F ':' '{print $1,$3,$4}'
```



Back to the simple example

```
db.people.aggregate(
{$match:XXX},          # Pipeline 1, filtering criteria
{$group:XXX}           # Pipeline 2, group by
)
```

- Filtering condition like with find()

```
$match:{name:{$gte:'M'}}
```

- Specifying the grouping field

```
$group:{_id:"$name",...}
```

- Specifying the aggregated fields

```
$group:{..., total:{$sum:1}}
```

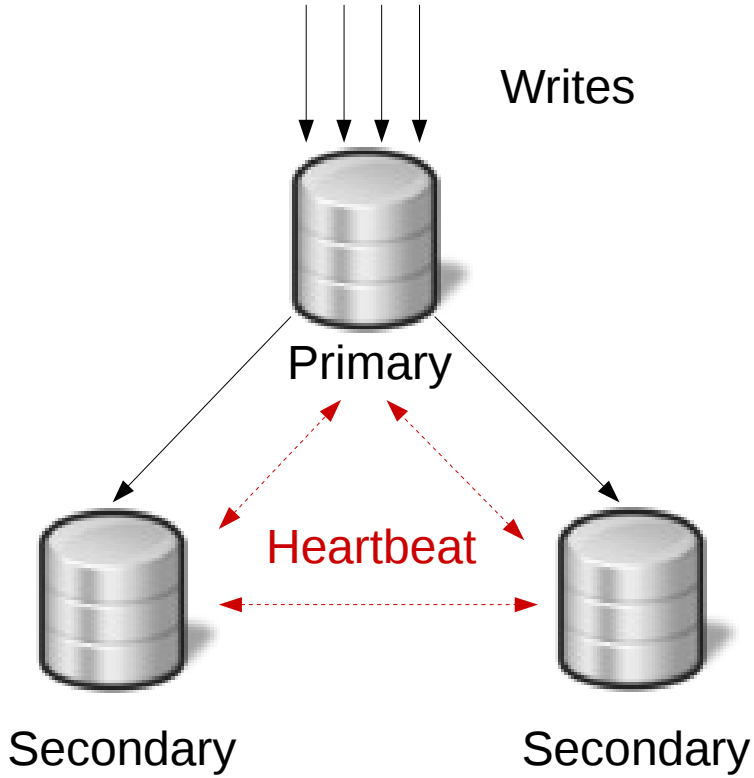
Agenda

- Introduction – Why MongoDB?
- Basic CRUD
- Schema Design
- Aggregation Framework
- Replication
- Sharding

Replication in 60s

- Asynchronous
  - All writes go to a single server (master)
- Secondaries can accept reads
  - By default they don't
- If the master fails, a new master is automatically elected
- An arbiter an be set up to break ties
  - No data is stored on an arbiter

MongoDB replica set



How failover works

- Heartbeats every 2 seconds
- If no reply within 10 seconds: node is marked as down
- If master does not reply: other nodes elect a new master
  - A priority can be set to influence the result
  - Priority 0 or hidden members can't be promoted
  - An arbiter can't be promoted

Write concerns

- “Choose your durability”
- Specifies how many nodes should ack a write
  - w=1: primary only (default)
  - w=2: primary and any secondary
  - w=majority: majority of the nodes
- A high w means safer data but also slower writes

Read preferences

- Where can you read from?
  - primary (default)
  - primaryPreferred
  - secondary
  - secondaryPreferred
  - nearest

Custom tags

- Specific tags can be set to any member
  - {'location': 'US'}, {'disks': 'SSD'}, {'role': 'reporting'} ...
- Write concerns and read preferences can use these tags
  - Useful when reads or writes must run on a given server or a given set of servers

Agenda

- Introduction – Why MongoDB?
- Basic CRUD
- Schema Design
- Aggregation Framework
- Replication
- Sharding

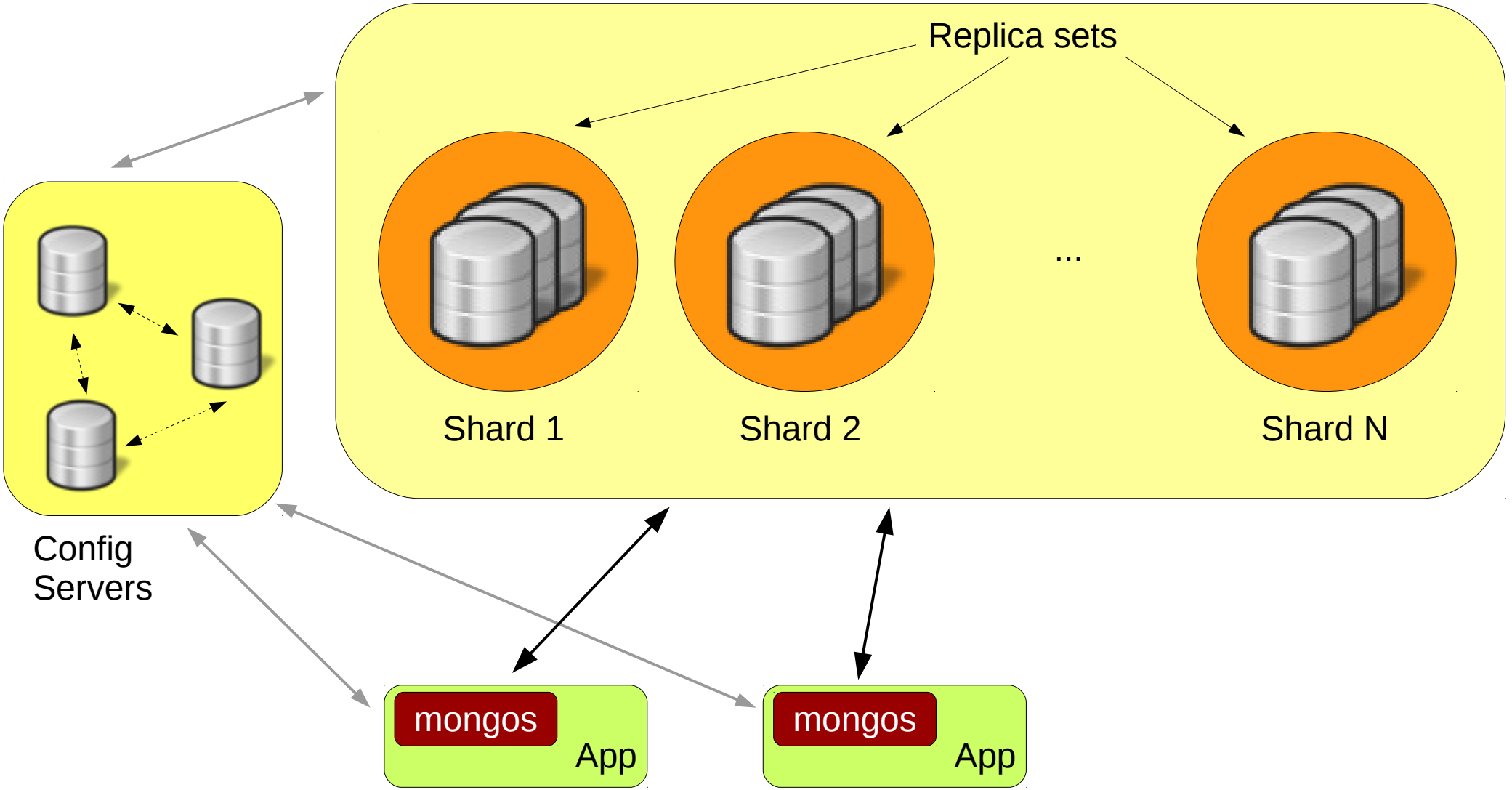
When to shard

- When write capacity exceeds the capacity of a single node
- When the dataset no longer fits in a single server
- When the working set no longer fits in memory

Components of a sharded cluster

- Shards
  - Replica sets to provide high availability
- Config servers
  - MongoDB instances storing the metadata
- mongos instances
  - Route queries to the right shards
  - Cache data from the config servers

Architecture of a sharded cluster



Sharding is automatic

- You specify
  - Which collections are sharded
  - The sharding key for each sharded collection
- Then in the background
  - Data is split in chunks
  - Chunks are spread over all shards
  - If shards or chunks become too large, data is split again and distributed across all shards

Sharding is transparent

- The app doesn't need to know about sharding
  - Connections must be made to mongos and not a specific mongod instance
- Every change in the sharding config is persisted in the config servers
  - And pushed to the mongos instances

Q&A

Thanks for attending!

Feel free to drop me a line at:

stephane.combaudon@percona.com

Special Discount for Webinar Attendees:
Use Code "**WebinarPL**" when registering to receive **20% off any ticket type**.
Valid on new registrations only.

http://www.percona.com/live/mysql-conference-2015/