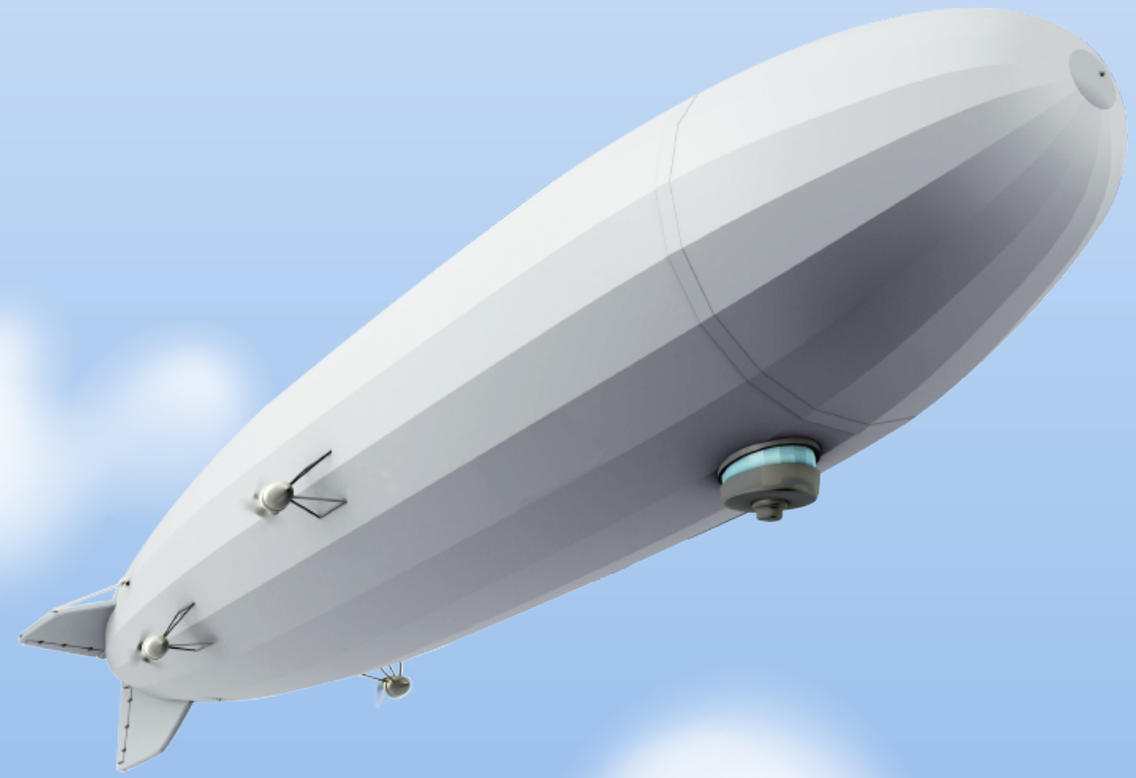


Scaling with MongoDB

by Michael Schurter 2011

@schmichael

What is MongoDB? Community

- Developer by 10gen
- AGPL Database
- Apache drivers
- JIRA issue tracking
- On GitHub

mongoDB

What is MongoDB? Architecture

- Server
  - Database
    - Collection (table)
      - Document (BSON; like a row)
        - Fields (columns)

What is MongoDB? Documents (2)

- **BSON**
  - Open standard: bsonspec.org
  - Binary JSON or protobufs without the schema
    - Objects/Sub-documents/mappings
    - Arrays
    - UTF-8 Strings
    - Floats, Integers (32 or 64 bit)
    - Timestamps, DateTimes
    - Booleans
    - Binary
    - Assorted others specific to Mongo's use case (Regex, ObjectID)

What is MongoDB? Querying

- **Querying**
  - Dynamic queries (JavaScript code or objects)
  - Map/Reduce (JavaScript functions)
  - Secondary indexes (B-tree, R-tree for simple geospatial)

What is MongoDB? Querying (2)

- Search by value or inside an array:
- `db.collection.find({tags: "some tag"})`
  - ⇒ `[{...}, {...}, ...]`
- Update tag lists:
- `update({app: "...", $addToSet: {tags: "another tag"})`
- No escaping values
- Declarative syntax makes for easy composition

What is MongoDB? Operations

- Replication
  - Master/Slave
  - Replica ~~Pairs~~ Sets
- Auto-sharding (data partitioning)
- Tools
  - mongo shell, mongostat
  - mongo{dump,restore,export,import}

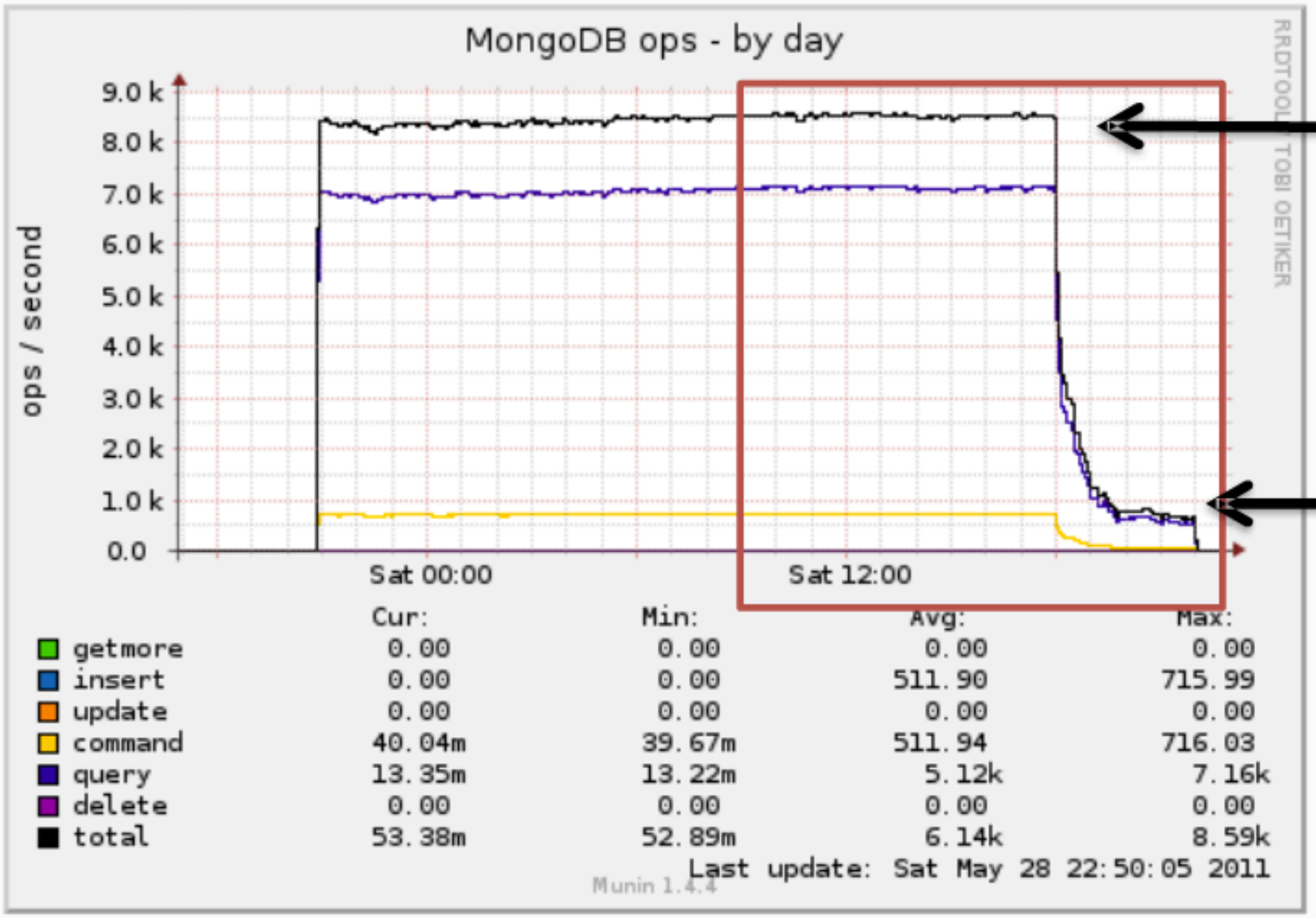
What is(n't) Mongo? Durability

- Single server durability added in 1.8 (off by default)
  - Preference is Replica Sets
- No guarantee your data was written with **safe=True**
  - ***Use safe=True***
- No guarantee your data was replicated without w=1
- If a server goes down, trash it's data and use a slave

What is MongoDB? Performance

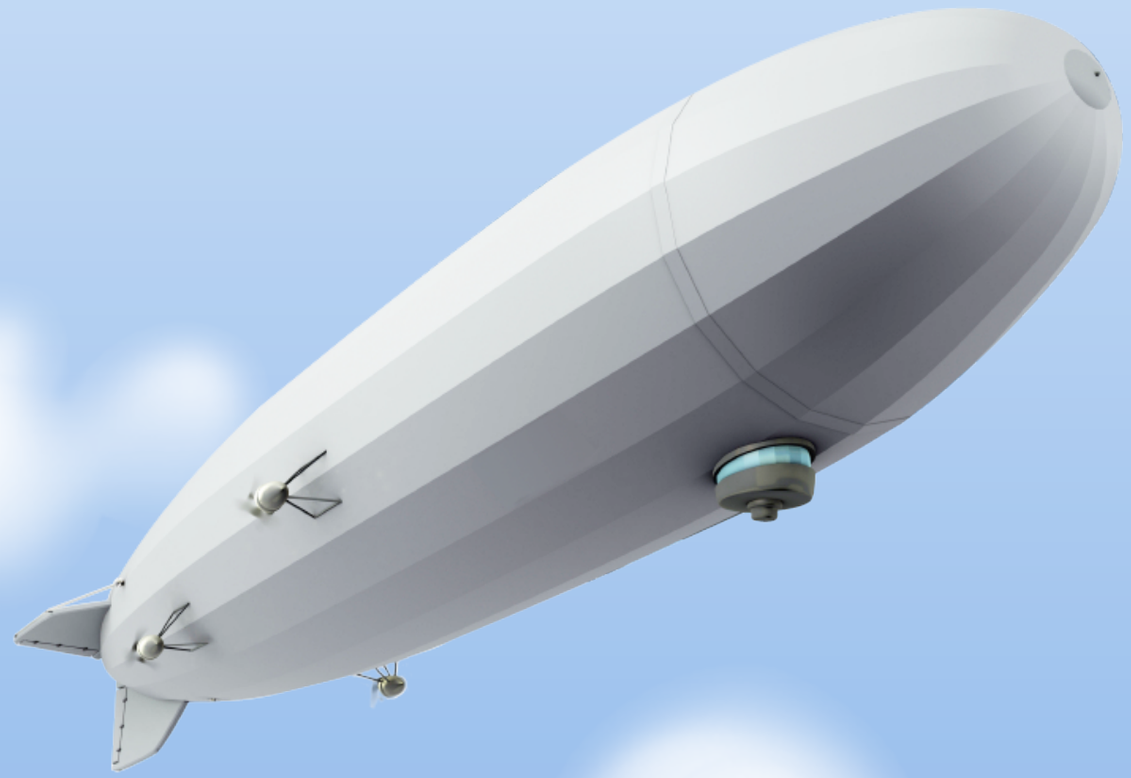
- I hear it's fast
- It is until:
  - Your data no longer fits in memory
  - Your indexes no longer fit in memory
  - You miss the index (full collection scan)
  - You use safe=True
  - You use w>0
  - You turn on journaling/durability

Ops per second



In RAM

Not In RAM



MOAR RAM

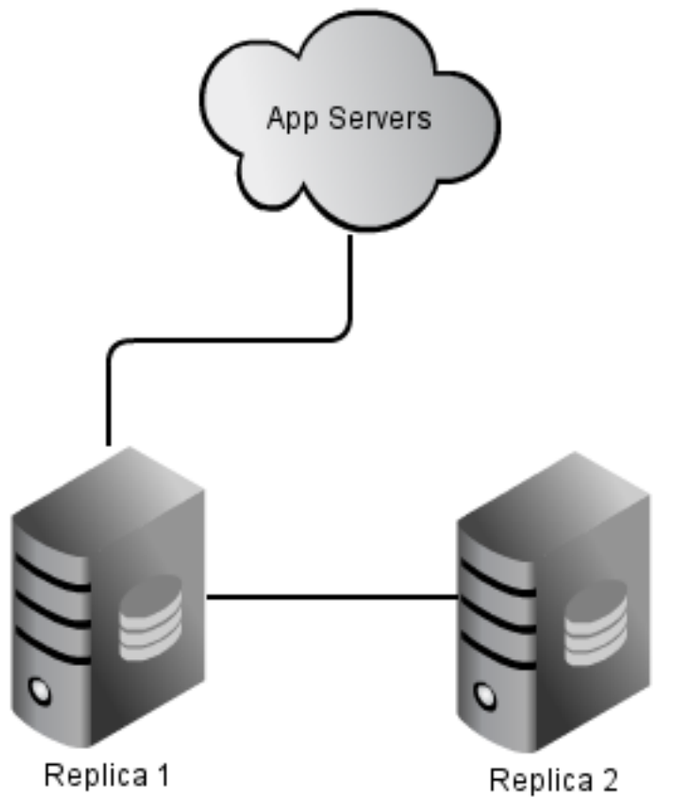
Rinse; repeat.

In the beginning

- Project at YouGov prior to Urban Airship
  - User/group database
  - Was custom datastore, migrated to MongoDB 1.2
- Highly recommended to Michael Richardson
  - Single PostgreSQL instance dying under load
  - I snuck into Urban Airship before anything blew up

Early perf. issue: syncdelay

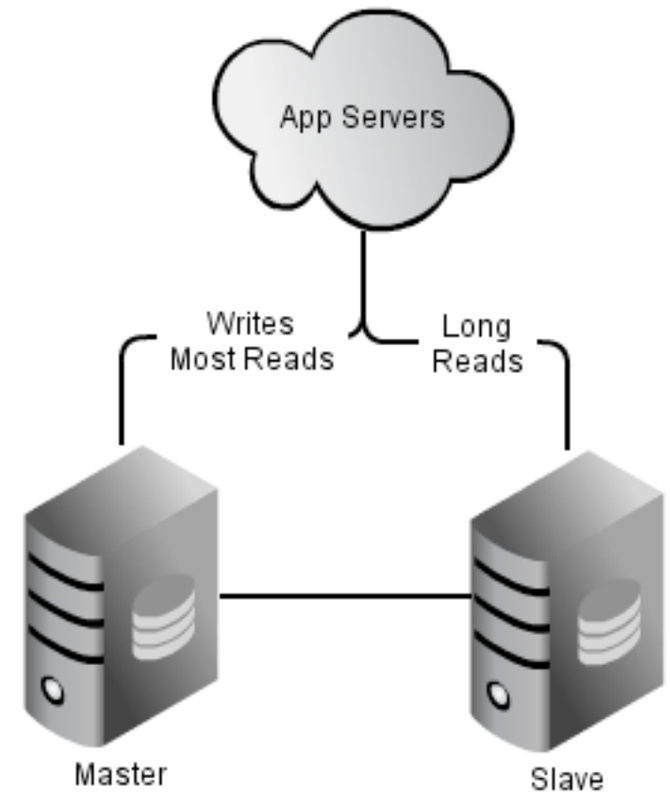
- The theory: Durability within a certain time-frame. (default: 60s)
- Barely documented
- Never worked for us
  - Syncs would cause infinite block loops: block for 30s, apply backed up writes, block another 30s, never catch up.
  - Just let the kernel handle flushing


App Servers
Replica 1
Replica 2

Replication

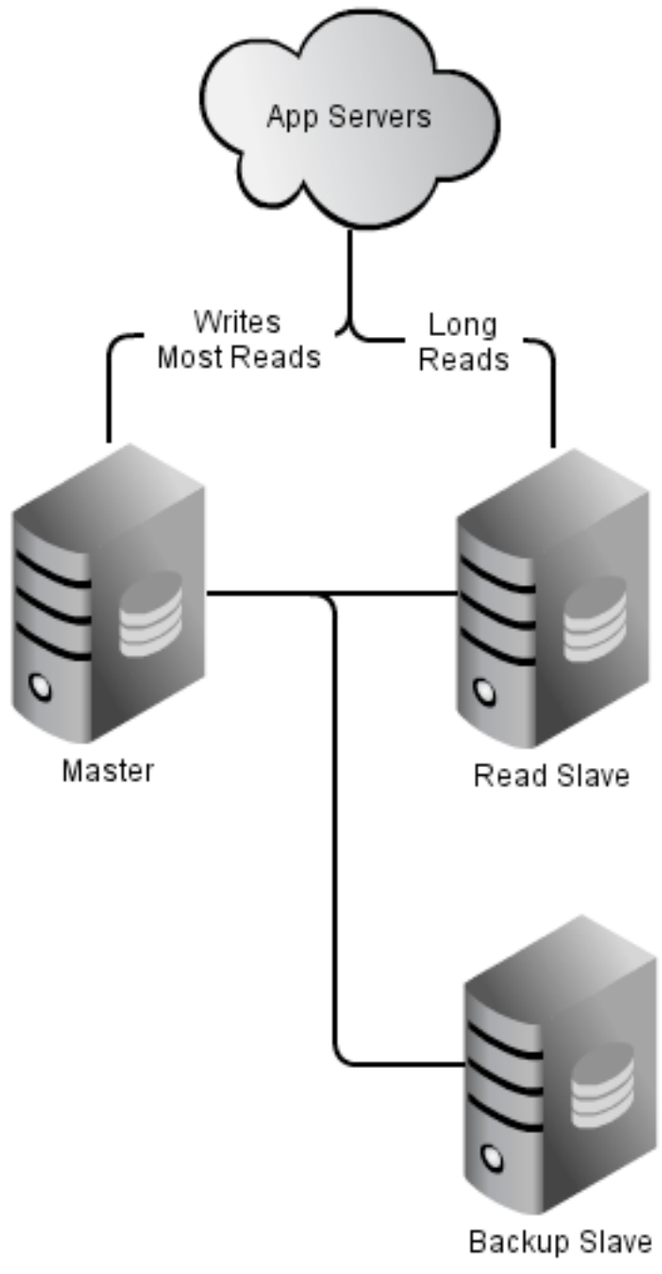
- Streaming asynchronous replication
  - Streams an oplog; no conflict resolution
- Master accepts writes, can read from slaves
  - Master + Slaves or...
  - Replica Sets (auto-election & failover)
- Non-idempotent operations like $inc/$push/etc are changed to their idempotent form:

```
{devices: 1,560} → {$inc: {devices: 1}} ⇒ {devices: 1,561}
```


App Servers
Writes
Most Reads
Long
Reads
Master
Slave

Locked In

- MongoDB only has Global (per-server) locks
  - Early versions (~1.2) locked server on every query
  - Later (~1.4) separate read & write locks
    - Present (>=1.6) updates use read lock while seeking
    - Future (1.9?) Per collection locks
- Moral: ***Do not run long queries on your master***


App Servers
Writes
Most Reads
Long
Reads
Master
Read Slave
Backup Slave

Double (Up)Dating

- Cause: **`update(..., {$push: {`*`big object`*`}})`**
- Effect:
  - Big object exceeds document padding
  - Document is moved to end of data
  - Update comes along and ***re-updates all documents***

Flip/Flop for the Win

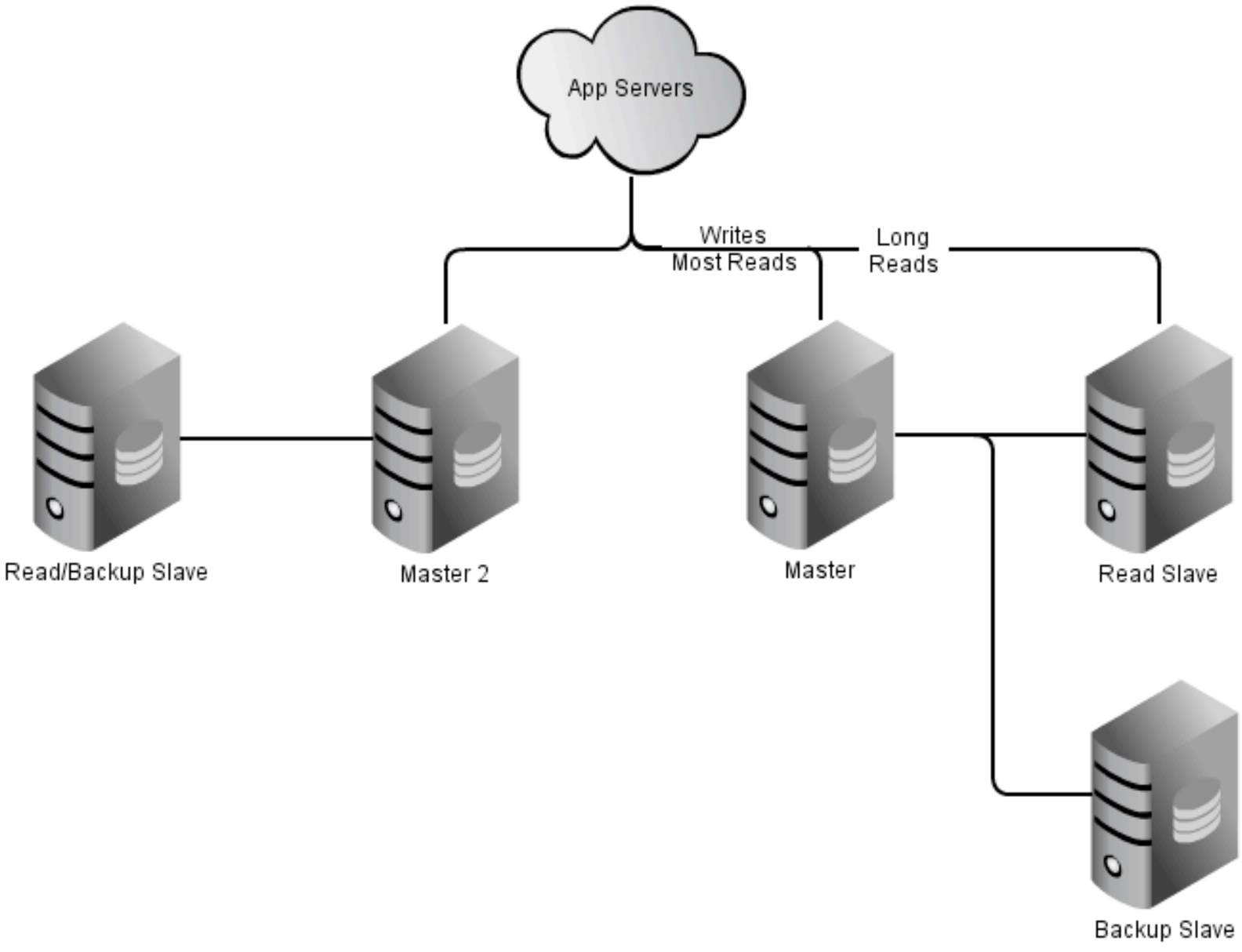
- Data files get sparse as documents are moved
- Indexes could get sparse (getting better & better)
  - Sparse indexes new in 1.8 (have to recreate old indexes)
- The Solution: **Take slave offline, resync, failover**
  - Requires downtime without Replica Sets
- Future (1.9) - In-place compaction

When adding RAM isn't enough

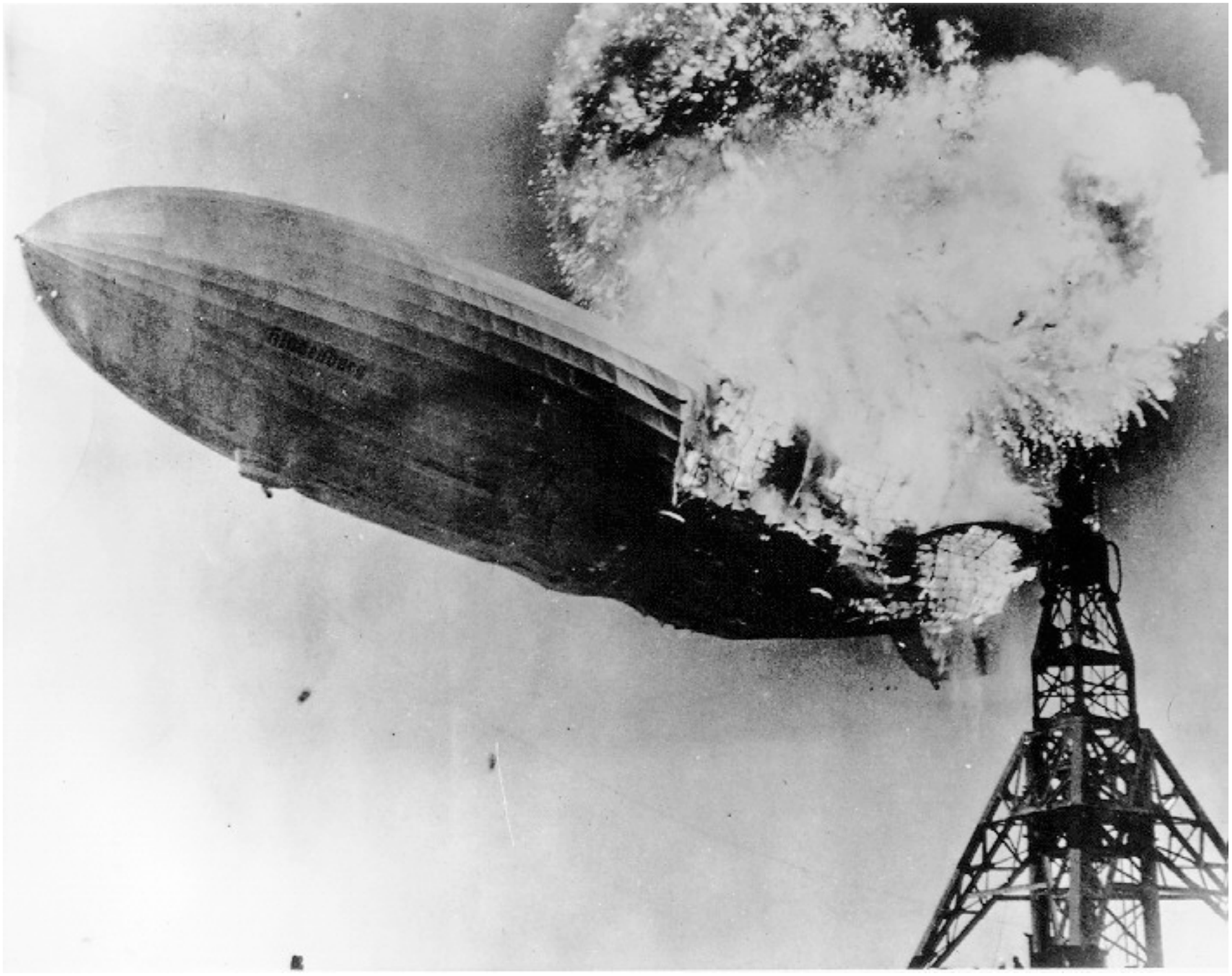
- More RAM doesn't alleviate pain of full server locks
- You have to write to disk someday
- Solution: Partition the cluster
  - Good old days: manually shard your data
  - Brave new world (1.6): **auto-sharding**

Auto-Sharding

- We don’t use it
  - Couldn’t get it to work in any of the 1.5 dev releases
  - Couldn’t pay me enough to use 1.6.0, maybe safe post 1.8.1
- Auto-shards based on a shard-key per collection
  - Order preserving partitioner
  - Querying anything other than the shard-key spams the cluster
  - Each shard should be a Replica Set
  - Run 1 mongos per app server


App Servers
Writes
Most Reads
Long
Reads
Read/Backup Slave
Master 2
Master
Read Slave
Backup Slave

Disaster Recovery



EBS Goes on Strike

- EBS volumes grind to a halt (or at least 1 per RAID)
- Restore from backup in a different Availability Zone!
- 4 hours later the restore is still building indexes
- Wait for EBS to come back or use stale backup data?
- In the end: EBS came back before restore finished.

mongorestore --indexesLast

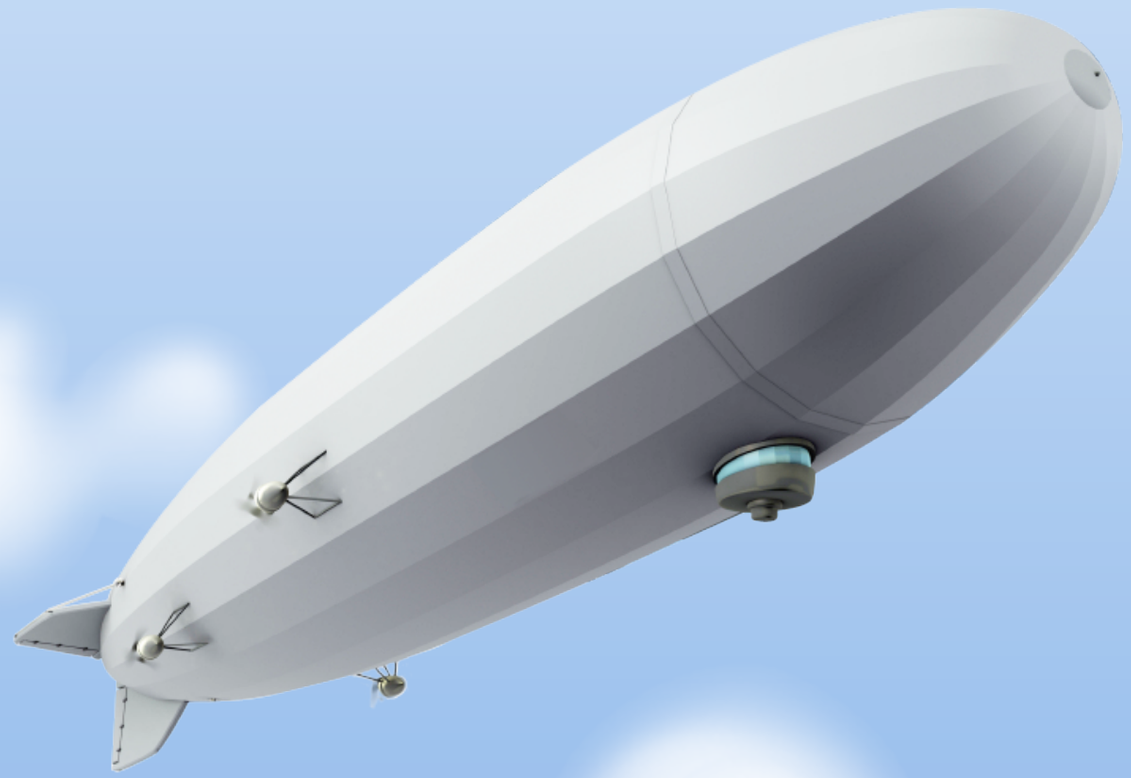
- Always use it
- Should be On by default
- Lock database and copy files (snapshot) is best

Goodbye MongoDB

- Moved bulk of data to manually partitioned PostgreSQL
  - 120 GB of MongoDB data became 70 GB in PostgreSQL
- Migration difficult due to undisciplined MongoDB code

Moral

- Test MongoDB for your use case
  - If you can't get auto-sharding working, probably run
  - That goes double for Replica Sets
- Choose your schema well (especially with sharding)
- Use short key names + a data abstraction layer
  - {created_time: new DateTime()} ⇒ 27 bytes
  - {created: new DateTime()} ⇒ 22 bytes
  - {ct: new DateTime()} ⇒ 17 bytes



Questions?

Content, text & diagrams, public domain (steal away)

Slide template copyright Urban Airship (sorry)