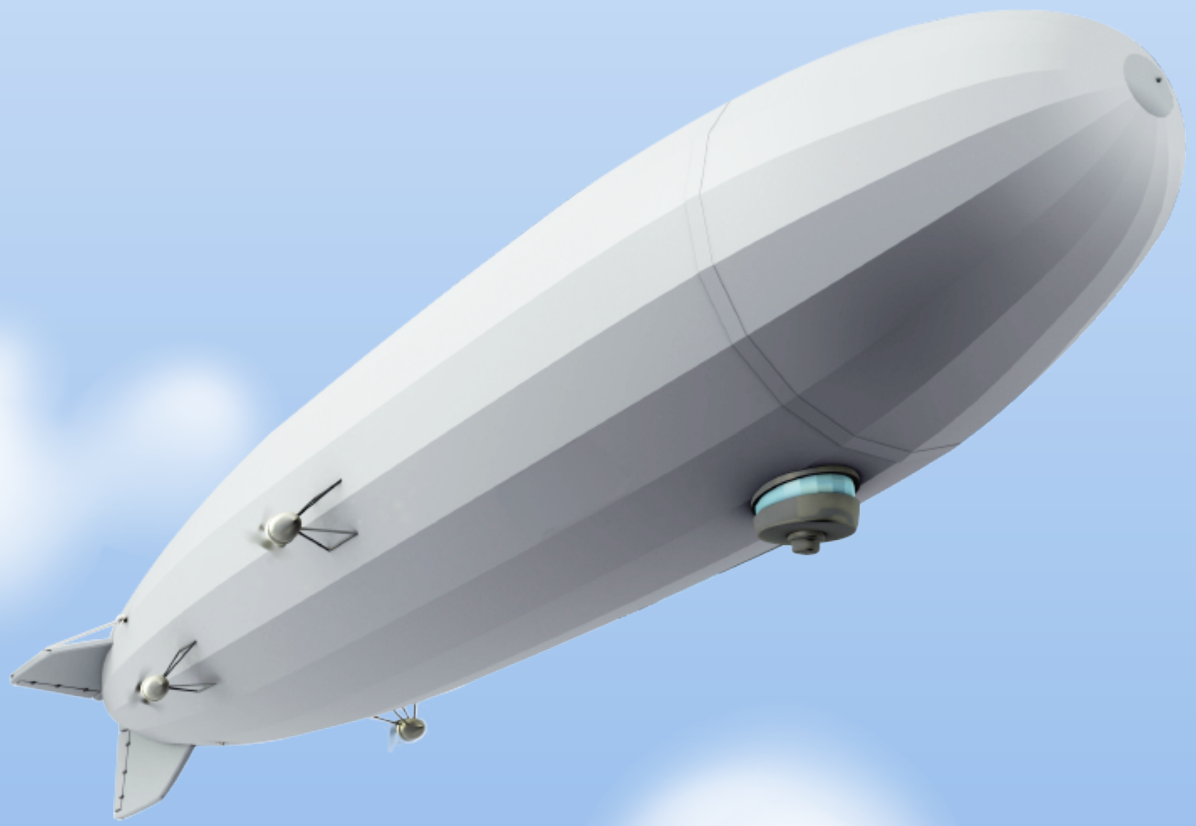


Scaling with MongoDB

by Michael Schurter 2011

@schmichael

What is MongoDB? Community

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

What is MongoDB? Architecture

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

Queries are on the collection level (no joins)
Indexes are per collection
Documents have a unique ID
Atomicity is at the document level

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

Must escape keys and insure values aren’t objects

What is MongoDB? Operations

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

Multi-master slaves
mongo shell is javascript
mongostat is *amazing*

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

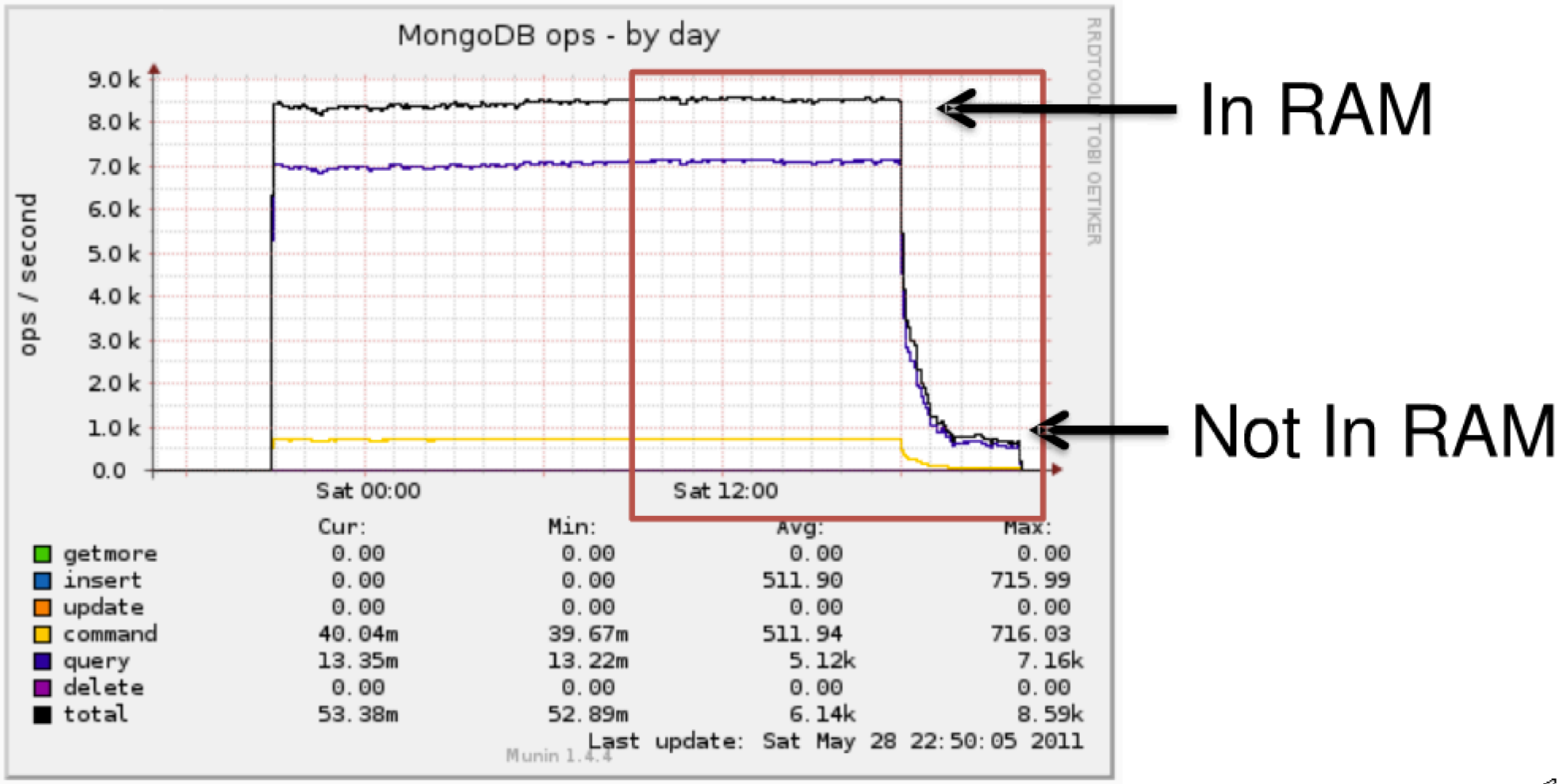
Unknown performance hit for durability
Probably worse than systems that ship WALs/journals as part of replication
safe=False is literally fire and forget

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

Oodles of marketing
Anecdote: Cassandra vs. Memcache

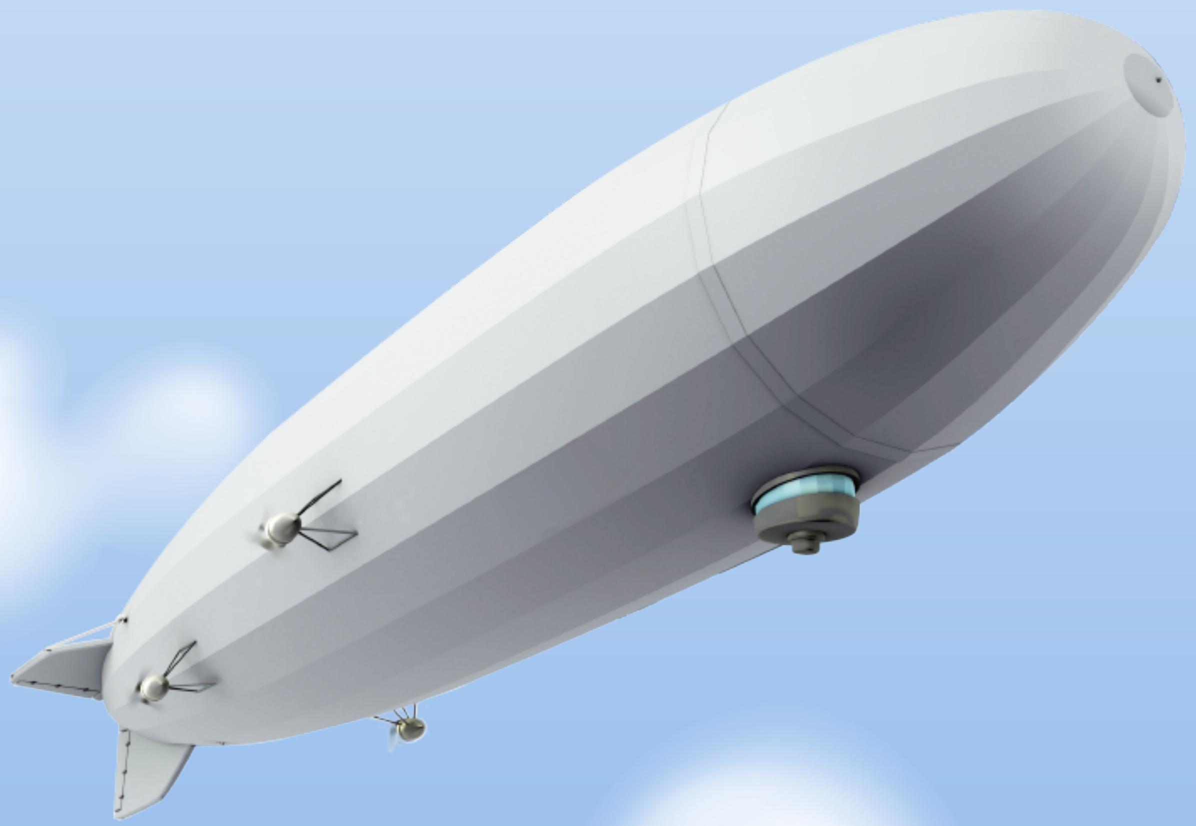
Ops per second



	Cur:	Min:	Avg:	Max:
🟩 getmore	0.00	0.00	0.00	0.00
🟦 insert	0.00	0.00	511.90	715.99
🟧 update	0.00	0.00	0.00	0.00
🟨 command	40.04m	39.67m	511.94	716.03
🟦 query	13.35m	13.22m	5.12k	7.16k
🟪 delete	0.00	0.00	0.00	0.00
⬛ total	53.38m	52.89m	6.14k	8.59k

Last update: Sat May 28 22:50:05 2011

Random internet picture of a read heavy system
Ours was write heavy and ran into something else first... but I'm getting ahead of myself



MOAR RAM

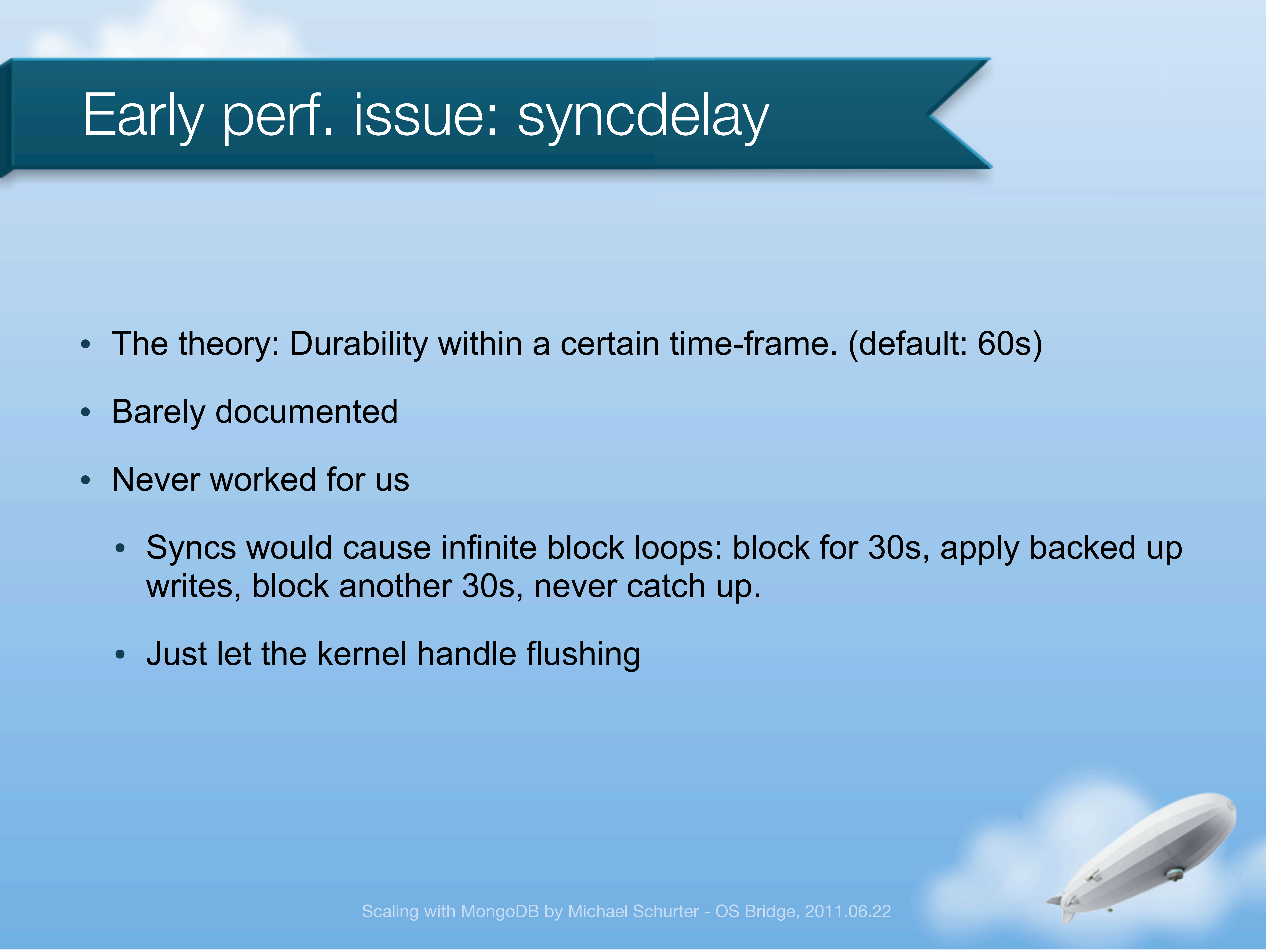
Rinse; repeat.

RAM is probably your single most important resource
SSDs would be great
Sharding essentially buys you the ability to add RAM forever
lol voltdb

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

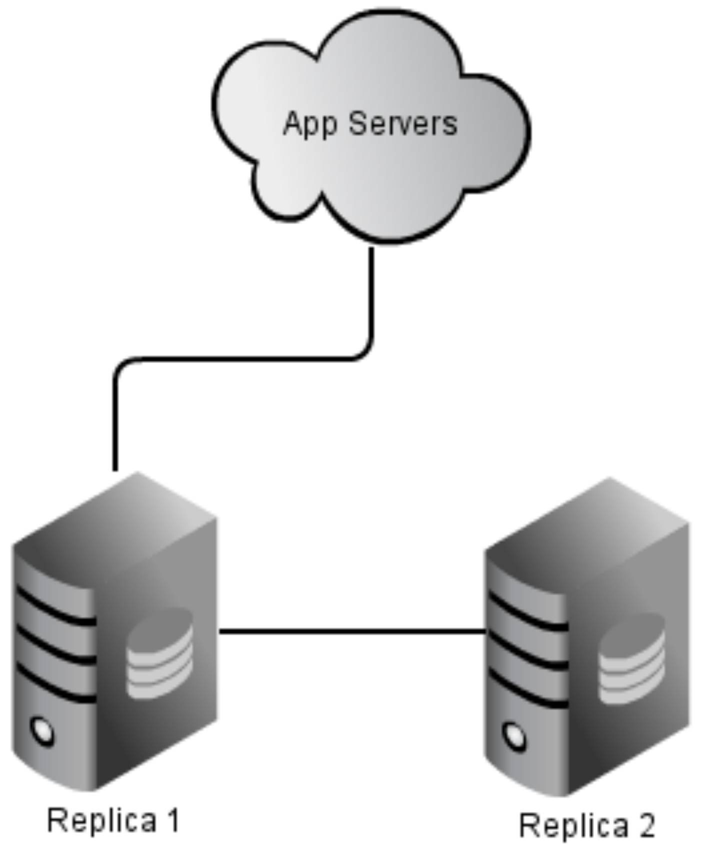
- User permissions and profiles were relatively complex
- Initially stored in a BDB-backed CouchDB-inspired Python service.
  - Worked fine, a bit slow, major NIH-Syndrome
- MongoDB (1.2?) to the rescue!
  - Perfect fit for a small read-heavy, write-light dataset



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

syncdelay useless with journaling? No hints in docs
There are proc tunables for how the kernel handles dirty pages



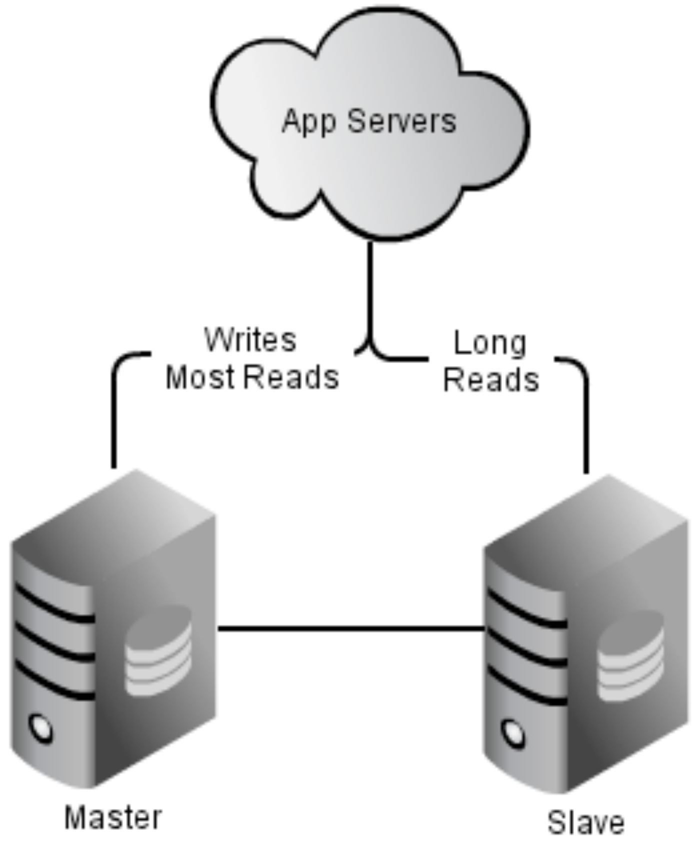
Initial MongoDB setup at Urban Airship

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}
```

But we were using replica pairs which were deprecated before replica sets were even officially released
Had a driver + replica sets issue where writes with safe=False went to slave (safe=False means we never saw an error)

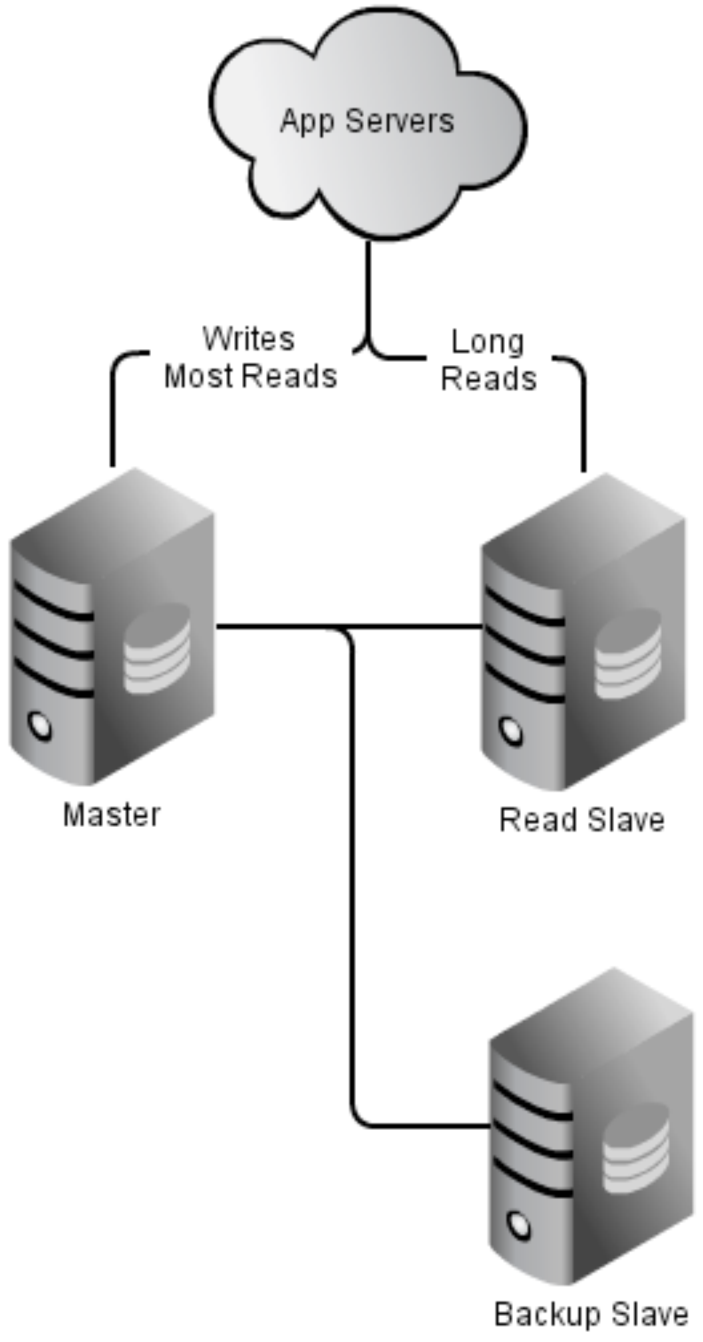


After the replica set bug we went to a simpler setup

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***

Similar to dynamic programming languages
Keep indexes in memory to keep read locks short
Have fast disks – NOT EBS (See Gavin's talk http://bit.ly/j6pR21)
https://jira.mongodb.org/browse/SERVER-1240



3rd generation mongodb setup - separate long reads from backups

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***

Changing your schema can avoid this
... how do you change schemaless data’s schema? **not easily**
Isn’t guaranteed to hit all documents, so you can’t just $pop once

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

Docs are padded to avoid moving; but when they are moved, gaps can grow
Poor data locality in disk / in memory
Schemaless means lots of space wasted repeating field names / structure
In-place - *not* online. Queries & replication stopped before compaction

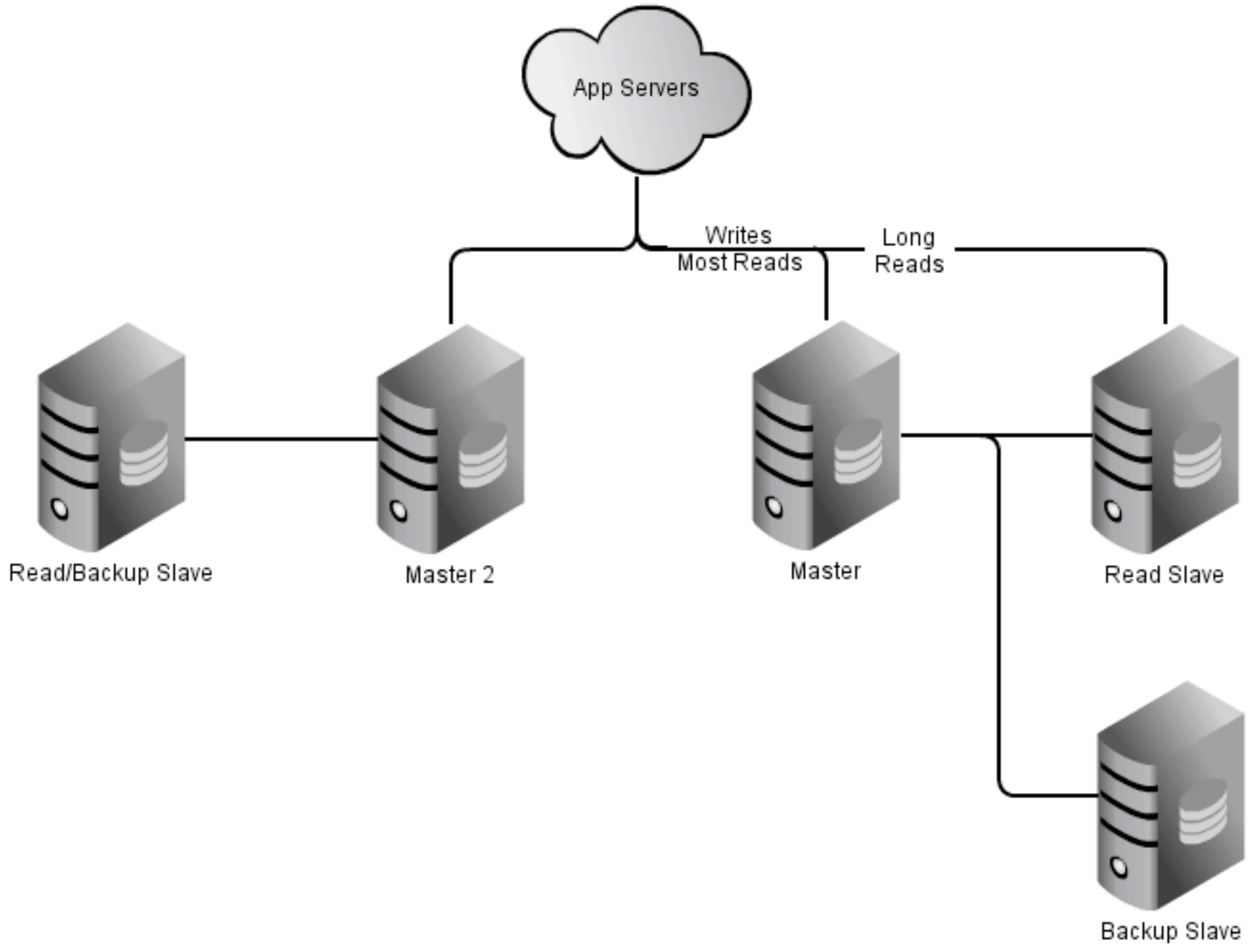
When adding RAM isn't enough

- 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

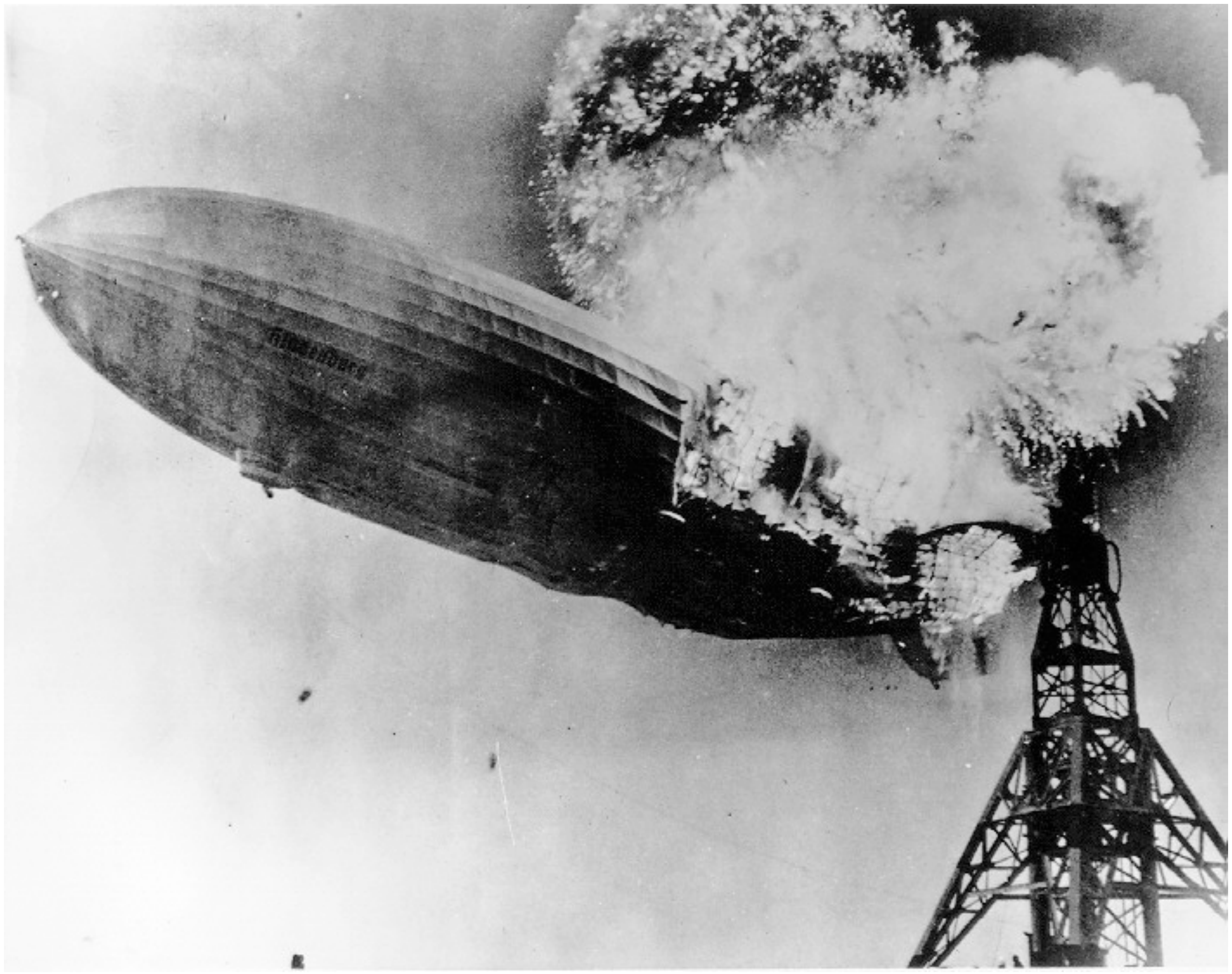
- We don’t use it
  - Couldn’t get it to work in any of the 1.5 dev releases
  - Couldn’t pay me enough to use 1.6.0, maybe safe post 1.8.1
- Auto-shards based on a shard-key per collection
  - Order preserving partitioner
  - Querying anything other than the shard-key spams the cluster
  - Each shard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

Early adopters seemed to all have 10gen support & “special builds”
Took down 4sq because adding shards & rebalancing incurs expenses in any distributed system (most? maybe there are clever ways to mitigate...)
Much larger ops burden
Make the right decisions initially (not totally schemaless)



4th (and final) generation
Shard by unrelated datasets
Don’t use an underpowered backup slave
Monitor replication delay closely - in a disaster nothing else matters if your replication delay is *14 days*

Disaster Recovery



So what happens when things go *really* wrong?

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.

When your HDD locks, MongoDB locks

mongorestore --indexesLast

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

Waiting for indexes to build takes approximately as long as it takes AWS engineers to
EBS issues.

Goodbye MongoDB

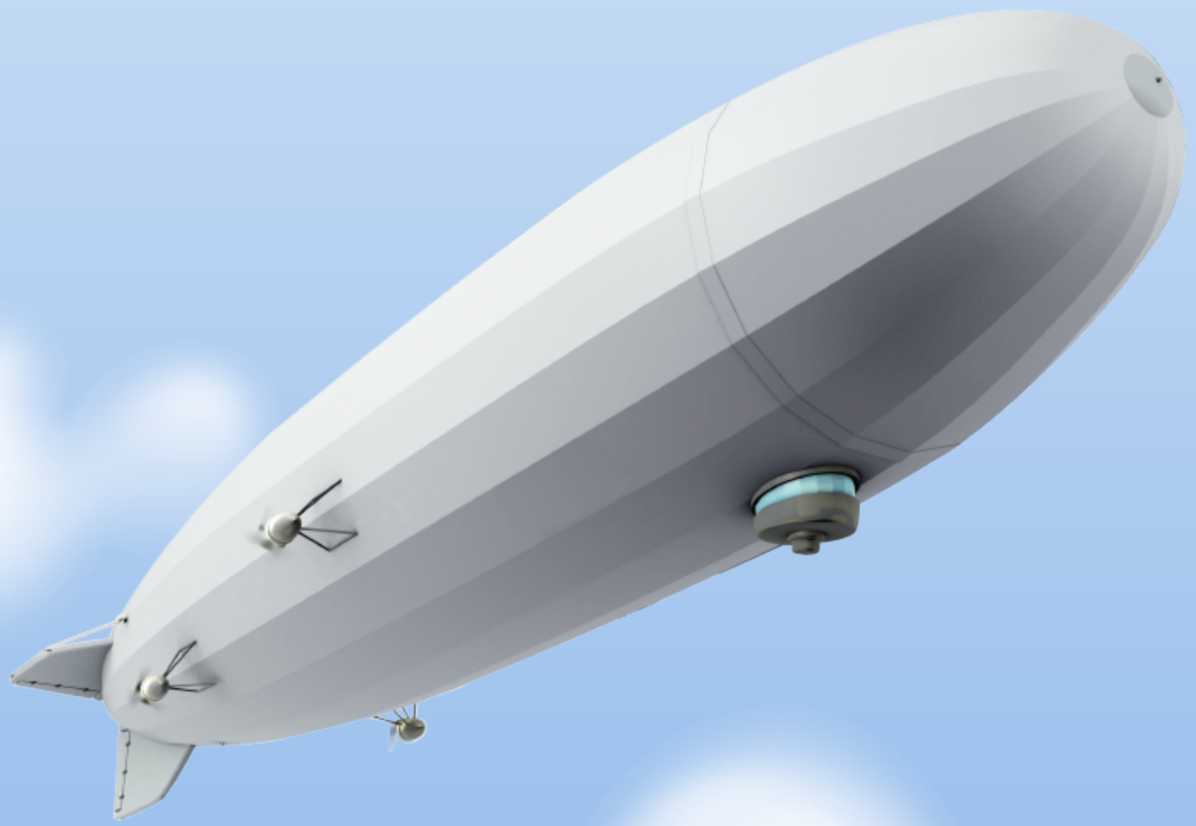
- Moved bulk of data to manually partitioned PostgreSQL
  - 120 GB of MongoDB data became 70 GB in PostgreSQL
- Migration difficult due to undisciplined MongoDB code

PostgreSQL 9.0’s streaming replication is nice
Slightly better failover story than MongoDB Master/Slave
Lack of data abstraction layer let us get sloppy – long migration

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

With Auto-sharding *and* Replica Sets, MongoDB’s scaling and durability stories are just too scary to trust.
You’ll thank me when your data files are 1/3 smaller.



Questions?

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

Slide template copyright Urban Airship (sorry)