Parse

Charity Majors
@mipsytipsy
Parse

Charity Majors
@mipsytipsy
- Mobile backend
- 500k+ apps
- AWS
- MongoDB, cassandra, mysql, redis
- ruby & rails => golang
why upgrade?

- new features
- better performance
- better support from the vendor
- avoid code rot, don’t get too far behind current
- all of your cool friends have upgraded
A Very Short List Of Terrible Things Database Upgrades Have Done To Me

- 35% perf reduction, 60% perf reduction
- data corruption (so many flavors)
- db files deleted on startup. DELETED!!
- indexing race conditions
- invalid indexes bug causes collections to be unwritable
- undocumented change in geoquery behavior
- default storage format has 60% more bloat
- backwards-incompatible mysql replication
- storage format changes
- all geo indexes block global lock until the first document found
- undocumented query syntax changes
- changed the definition of scan limits, doesn’t cache query plans that exceed scan limit
- unindexable writes suddenly refused
- internally-assigned data type changes
- secondaries crash instead of pausing replication
- query planner fails to cache plans when race phase interrupted
- query planner caches plans for least data not representative data
- accepted a bad op in the primary which bricked secondaries preventing quorum
data integrity

query performance

your sanity
The Minimal Set:

- run unit tests
- read the release notes
- assess your appetite for risk
the cowboy continuum

yee haw!

whoa there …
Risk assessment

- How mature is the db?
- How critical is the data?
- How mature is your company?
- Can you roll back? How hard will it be?
- How much does your workload push the boundaries of the db?
- Are other people doing similar workloads?
- How much changed between releases?
yolo

# apt-get upgrade

nothing can ever change
let’s use oracle
Redis
Cassandra
MySQL
MongoDB
MongoDB 2.6 risk assessment for Parse:

• How mature is the db? — **NOT**
• How critical is the data? — **TERRBLY**
• How mature is your company? — **FAIRLY**
• Can you roll back? How hard will it be? — **DEPENDS**
• How much does your workload push the boundaries of the db? — **EXTREMELY**
• Are other people doing similar workloads? — **LONNO**
• How much changed between releases? — **A LOT**
Paranoid Upgrades

EFFECTIVE 99.9% OF THE TIME?

I DON'T LIKE THOSE ODDS

Paranoid Upgrades
Real production traffic
Real production traffic

- YOUR query set
- YOUR data set
- with YOUR hardware
- and YOUR concurrency
Correctness
Base Performance
Outliers

... p.s. don’t forget the clients
Correctness

- unit tests
- tools to replay sample queries against two primaries (e.g. pt-upgrade)
- traffic splitter
- bulk traffic capture + replay
splitter

API node

splitter compares responses and analyzes differences, returning prod db response to api request

prod db

shadow db
Base Performance

- Snapshot data
- Capture ops
- Replay ops
- Reset, tweak, repeat
Replay tools for mongo (flashback)

- **Snapshot** — from start of record run. Then create an LVM snapshot for resetting
- **Record** — python tool to capture ops
- **Replay** — go tool to play back ops
- **Rewind snapshot, rinse, repeat**
Replay tools for mysql

- Apiary (old, deprecated)
- Percona Playback (new, shiny)
Replaying

- n concurrent workers pulling off a queue
- as fast as possible, or follow timestamps?
- evict working set between runs (LVM snapshot reset does this, or echo 3 >/proc/sys/vm/drop_caches)
- compare logs for errors
- break down by op type and percentile
## Outliers

<table>
<thead>
<tr>
<th></th>
<th>2.4.10 P99</th>
<th>2.4.10 MAX</th>
<th>2.6.3 P99</th>
<th>2.6.3 MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>18.45ms</td>
<td>20953ms</td>
<td>19.21ms</td>
<td>60001ms</td>
</tr>
<tr>
<td>insert</td>
<td>23.5ms</td>
<td>6290ms</td>
<td>50.29ms</td>
<td>48837ms</td>
</tr>
<tr>
<td>update</td>
<td>21.87ms</td>
<td>3835ms</td>
<td>21.79ms</td>
<td>48776ms</td>
</tr>
<tr>
<td>FAM</td>
<td>21.99ms</td>
<td>6159ms</td>
<td>24.91ms</td>
<td>49254ms</td>
</tr>
</tbody>
</table>
Bug hunting time.

- `removeOp()` on Installation deviceid
  - [https://jira.mongodb.org/browse/SERVER-14311](https://jira.mongodb.org/browse/SERVER-14311)
- non-yielding full index scans
  - [https://jira.mongodb.org/browse/SERVER-15152](https://jira.mongodb.org/browse/SERVER-15152)
- intersection-based query plans cached over single index plans with occasional empty predicates
  - [https://jira.mongodb.org/browse/SERVER-14961](https://jira.mongodb.org/browse/SERVER-14961)
### Outliers — after

<table>
<thead>
<tr>
<th></th>
<th>2.4.10 P99</th>
<th>2.4.10 MAX</th>
<th>2.6.4 P99</th>
<th>2.6.4 MAX</th>
<th>2.6.5 P99</th>
<th>2.6.5 MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>18 ms</td>
<td>20,953 ms</td>
<td>19 ms</td>
<td>60,001 ms</td>
<td>10 ms</td>
<td>4,352 ms</td>
</tr>
<tr>
<td>insert</td>
<td>23 ms</td>
<td>6,290 ms</td>
<td>50 ms</td>
<td>48,837 ms</td>
<td>24 ms</td>
<td>2,225 ms</td>
</tr>
<tr>
<td>update</td>
<td>22 ms</td>
<td>3,835 ms</td>
<td>21 ms</td>
<td>48,776 ms</td>
<td>23 ms</td>
<td>4,535 ms</td>
</tr>
<tr>
<td>FAM</td>
<td>22 ms</td>
<td>6,159 ms</td>
<td>24 ms</td>
<td>49,254 ms</td>
<td>23 ms</td>
<td>4,353 ms</td>
</tr>
</tbody>
</table>
Confidence
“I upgraded and got 70% worse performance”

“I upgraded and 30% of my writes started getting rejected bc mongo started enforcing index key lengths”

“I upgraded and I’m getting corrupt data due to indexing race conditions”

“I upgraded and .01% of my apps started ordering slightly differently for certain find queries”

“I upgraded and one of my offline DW jobs had an incorrect implicit data type”

“I upgraded and had to adjust to a slightly different administrative workflow”
We’re not going for perfection here.

data, there will
Always Be Something Wrong
data integrity

query performance

your sanity
Resources

MongoDB:

- MongoDB flashback tools:
  - https://github.com/ParsePlatform/flashback
- Travis Redman’s slides on how we benchmarked 2.4 -> 2.6
  - www.slideshare.net/travisredman79/benchmarking-at-parse

Mysql:

- blog post on Linden Lab mysql upgrade:
- Apiary (deprecated):
  - https://bitbucket.org/lindenlab/apiary
- Percona toolkit:
  - http://www.percona.com/software/percona-toolkit
- Percona Playback:
  - http://www.percona.com/downloads/Percona-Playback/
Parse

Charity Majors
@mipsytipsy