The Story of Sharding at Box

Tamar Bercovici
@TamarBercovici

Scaling Web Architecture

- Some components of web architecture are easy to scale horizontally
- For example, front end machines
- Horizontally scaling your database is much more involved…

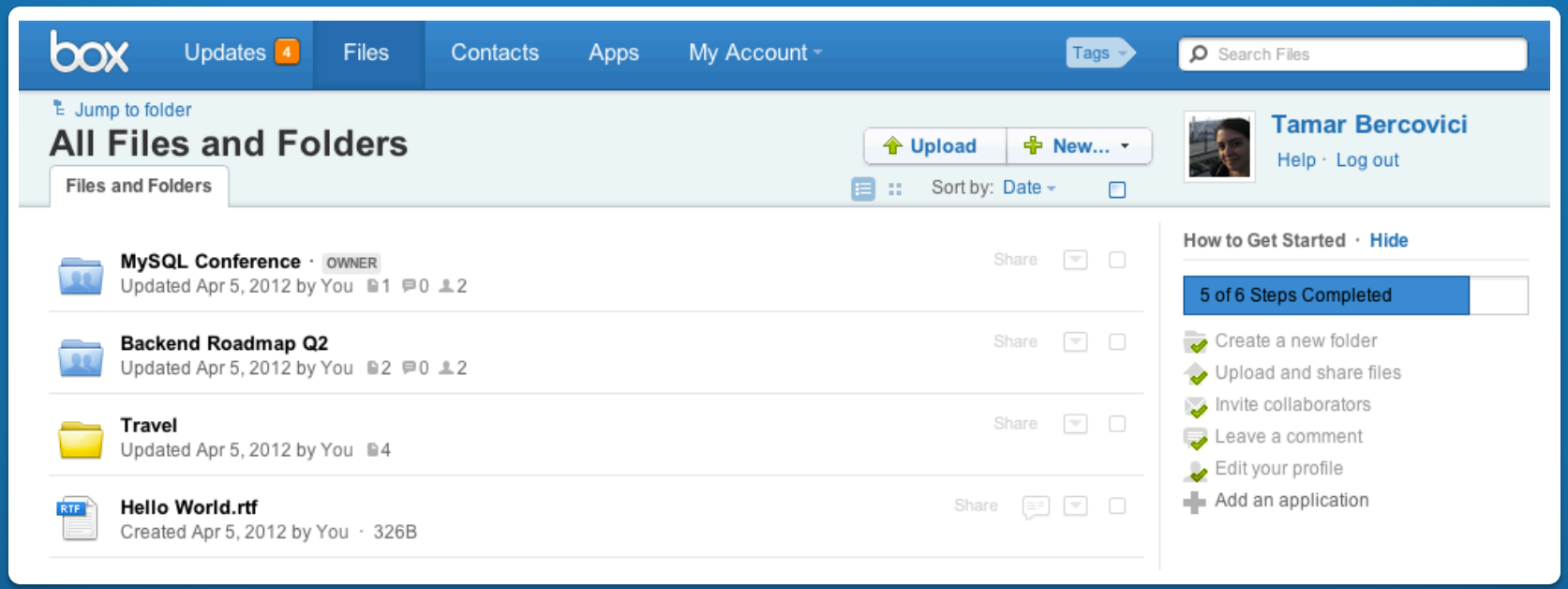
  Do you split traffic? Data?... Both?

  How?!

- At Box, we decided to shard our database – this is our story…

What is Box?

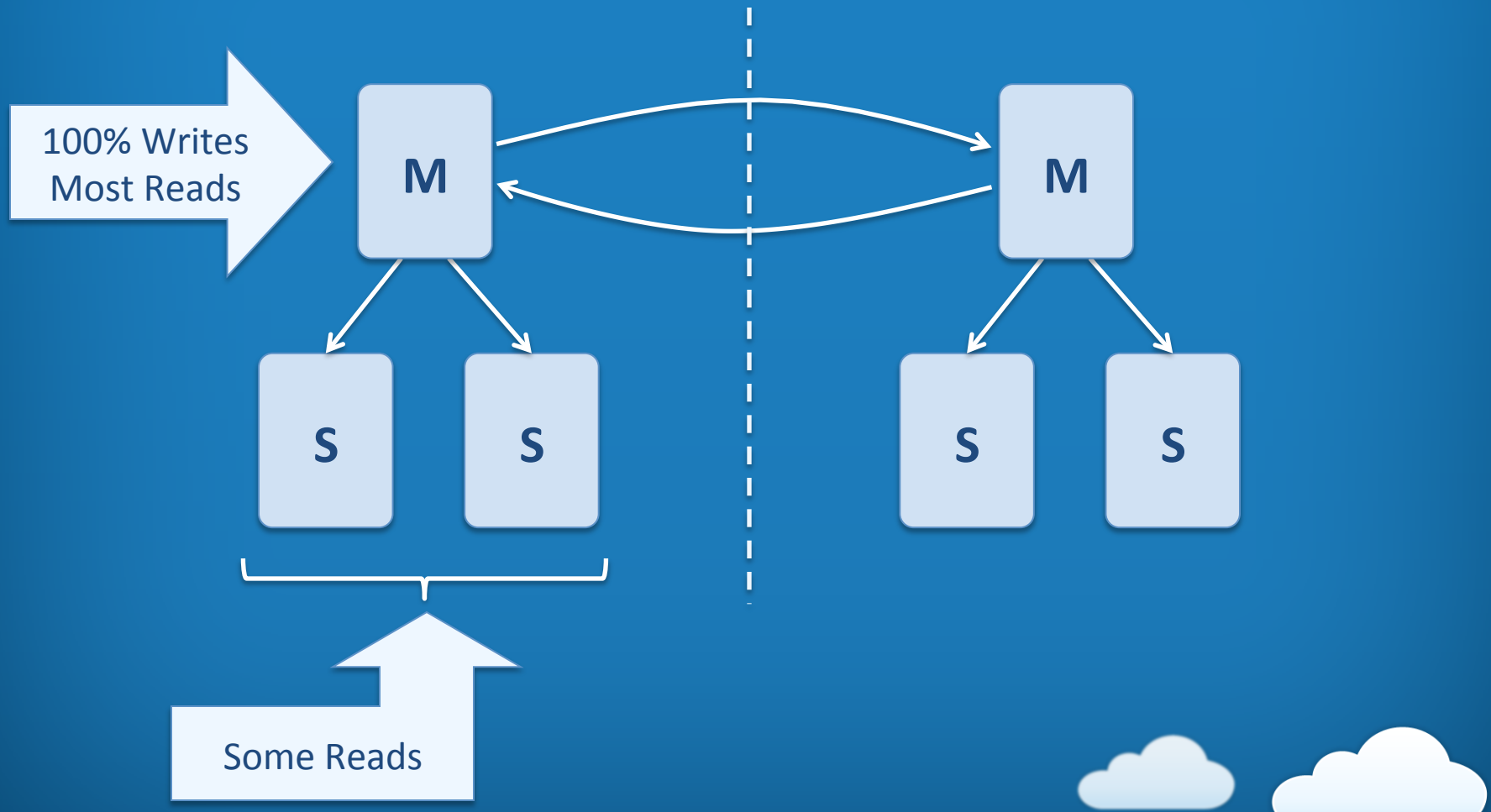
The simplest way for enterprises to share and access data from anywhere



Where we started off

- 1 million line codebase mainly in PHP
- Entire database on one MySQL 5.5 server
- Over 1.7B queries a day
- Millions of users
- Tens of millions of folders
- Hundreds of millions of files
- Aggressive growth expected

Initial Physical Configuration



Designing a Sharded Architecture

Pain Points and Constraints

Pain Points:

1. Key tables (file, folder) getting too big to manage
2. Read and write QPS growing fast

Constraints:

1. Scale ASAP
2. No downtime

Goals

Address Pain Points:

1. Horizontally scale file and folder tables

Within Constraints:

2. Do the minimum
3. Introduce as little complexity as possible
4. Make code changes incrementally
5. Make physical changes incrementally

Sharding Strategy

1. What to partition?
   - File and folder
   - Other tables, e.g. user table, not sharded (yet…)
2. How to partition?
   - Most queries fetch content from a user's account
   - Partitioning by user minimizes # database requests
3. How to locate content?
   - ID hash approach: simple but rigid
   - Mapping/Lookup DB: full flexibility

Choosing a Lookup Strategy

- Users tend to collaborate within their enterprise
  - → **Support for co-location of users in an enterprise**
- Users can move content into a collaborator's folder
  - → **Support for online moving of content between shards**
- Users can continually create more content
  - → **Support for splitting shards**
- Needed flexibility → went with mapping approach

Mapping Database

- Mapping of object ids to shard ids
- MySQL DB, separate from the application database
  - High reads / low writes (mainly inserts)
  - Very lightweight tables
  - All reads are PRIMARY INDEX lookups
- Why MySQL?
  - Simplify development, deployment and maintenance
  - Quickest safest way to meet our needs
  - Re-evaluate when it becomes a bottleneck

ID Generation

Goal: IDs should be unique, constant and backwards compatible

- IDs distributed by Mapping DB using auto-increment
- Every new object (file, folder…) is first added to the mapping, and then to the shard
- Mapping backfilled with existing IDs

→ *Another advantage of using MySQL for our mapping database*

Determining the Shard

- Sharded classes define **mapping keys**
  - Used to look up shard id in mapping db
  - e.g., for file: file_id, parent_folder_id and user_id
- ORM analyzes queries to find mapping keys
  - Easy for ORM queries such as `$folder->children()`
  - Complex queries require minimal parsing
  - Framework supports passing “hints”

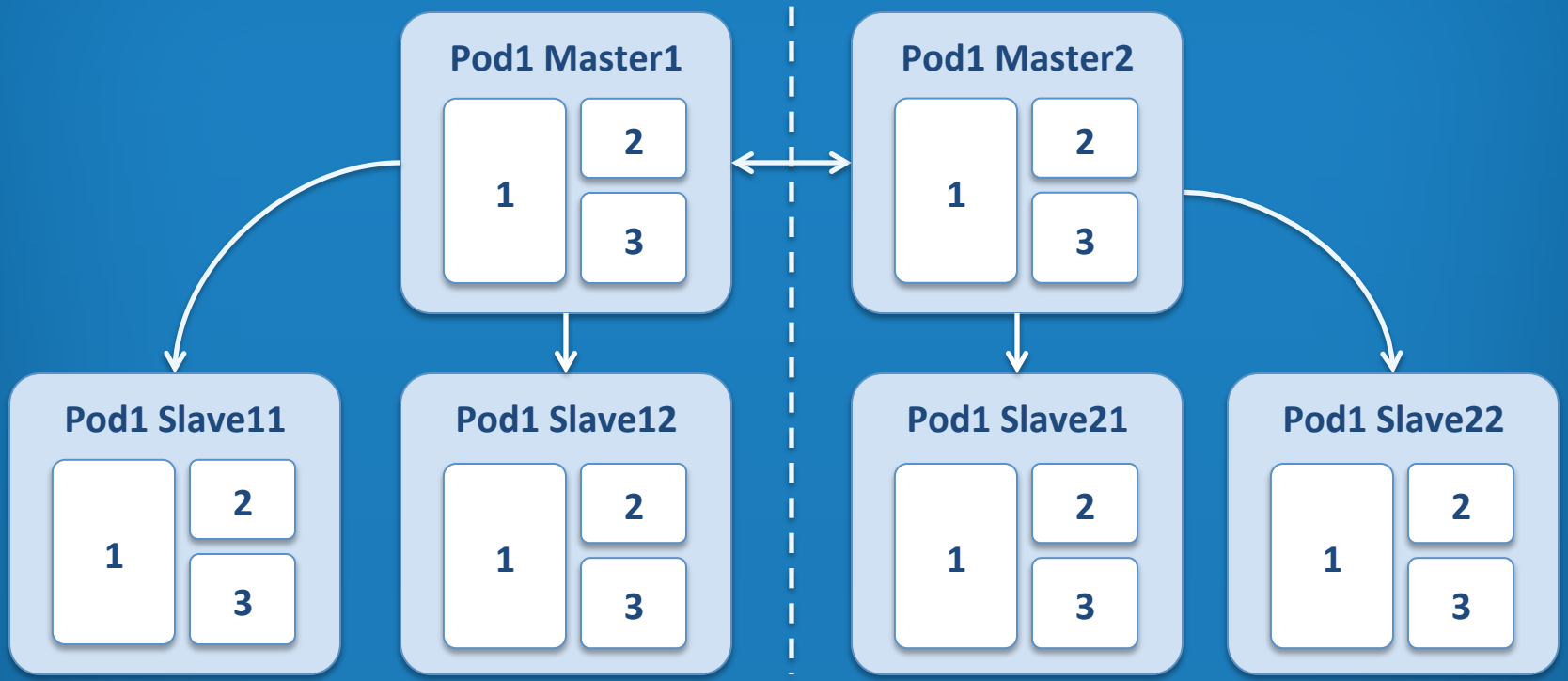
Querying the Shards

- To execute a SELECT / UPDATE / DELETE query:
  1. Analyze query and find a mapping key and ids
  2. Query the mapping to find the set of shard ids
  3. Execute the query on the all the referenced shards
  4. Combine results if necessary and return
- To execute an INSERT query:
  1. Choose shard for new content
  2. Insert into mapping db and obtain ID
  3. Insert into shard db

Online Cross-Shard Moves

- Sometimes objects need to move between shards
  - e.g., ownership changes for files and folders
- In `$object->save()` the ORM executes the following:
  - Check if mapping key being modified
  - If yes, check if new value mapped to different shard
  - If yes, move data to new shard and update mapping

Physical Layout of Shard Pod



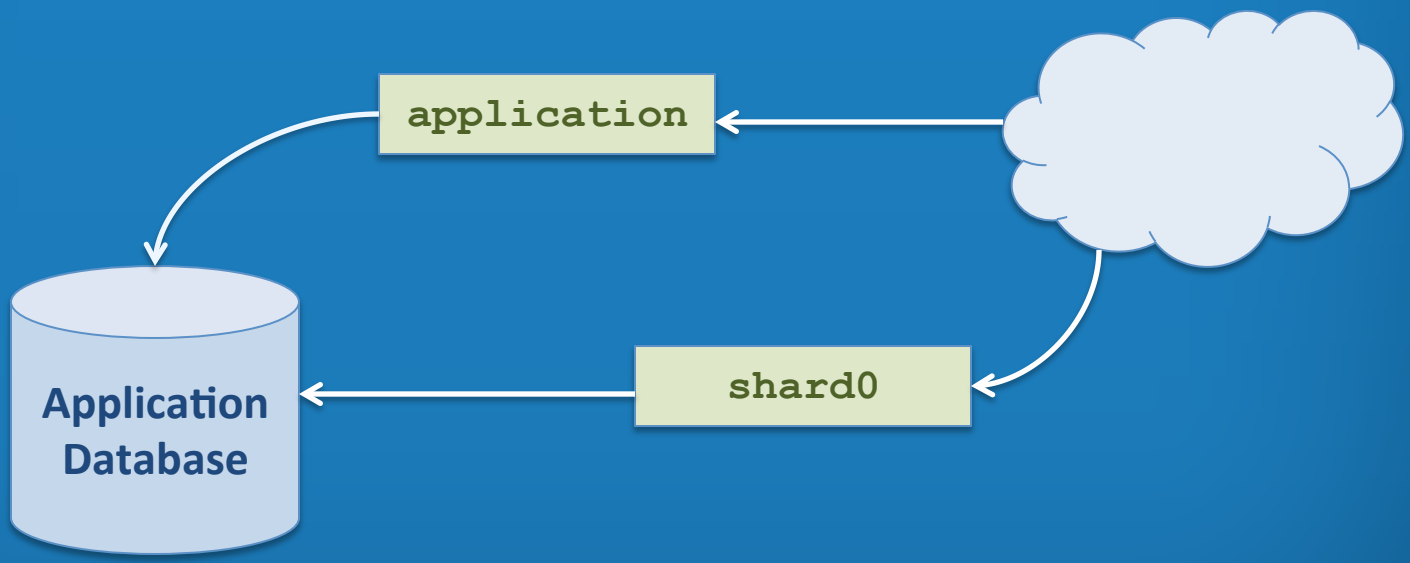
- Multiple shard databases per MySQL instance on a server
  - → Gives flexibility to co-locate “hot” and “cold” shards
- Optimizes for table size bottleneck

Transitioning the Code

… Without Breaking the Site

Incremental Roll-out Strategy

- To be able to incrementally roll-out app changes we use **logical sharding**



Ramping-up the Mapping DB

- Modify insert code to create mapping entries with **`shard_id = 0`**
- When switching to the new code, avoid id collisions
  - Bump Map DB auto-increment to be larger than App DB's
  - Set Map DB offset to opposite of App DB's (like in M-M)
- Backfill mapping with **`shard_id = 0`**
- The mapping is now consistent with the data layout: all objects are mapped and located on **`shard0`**

Ramping-up Querying Shards

- Log all queries and monitor for non-migrated code
- Gradually transition code to determine shard using the mapping database
  - Both new and old code hits the same physical database
  - Log errors and default to querying **`shard0`**
  - Monitor logs for code paths to fix
- Throughout the process – website functionality is not broken or altered

Automated Testing

- PHPUnit tests were **key** in our development process
  - Testing new code paths in isolation
  - Automatic set up of multiple databases
  - Confidence to tweak our low-level framework code
- Existing tests configured to run on single shard
  - Keep them deterministic
- Built support for creating content on multiple shards for focused testing

Query Monitoring

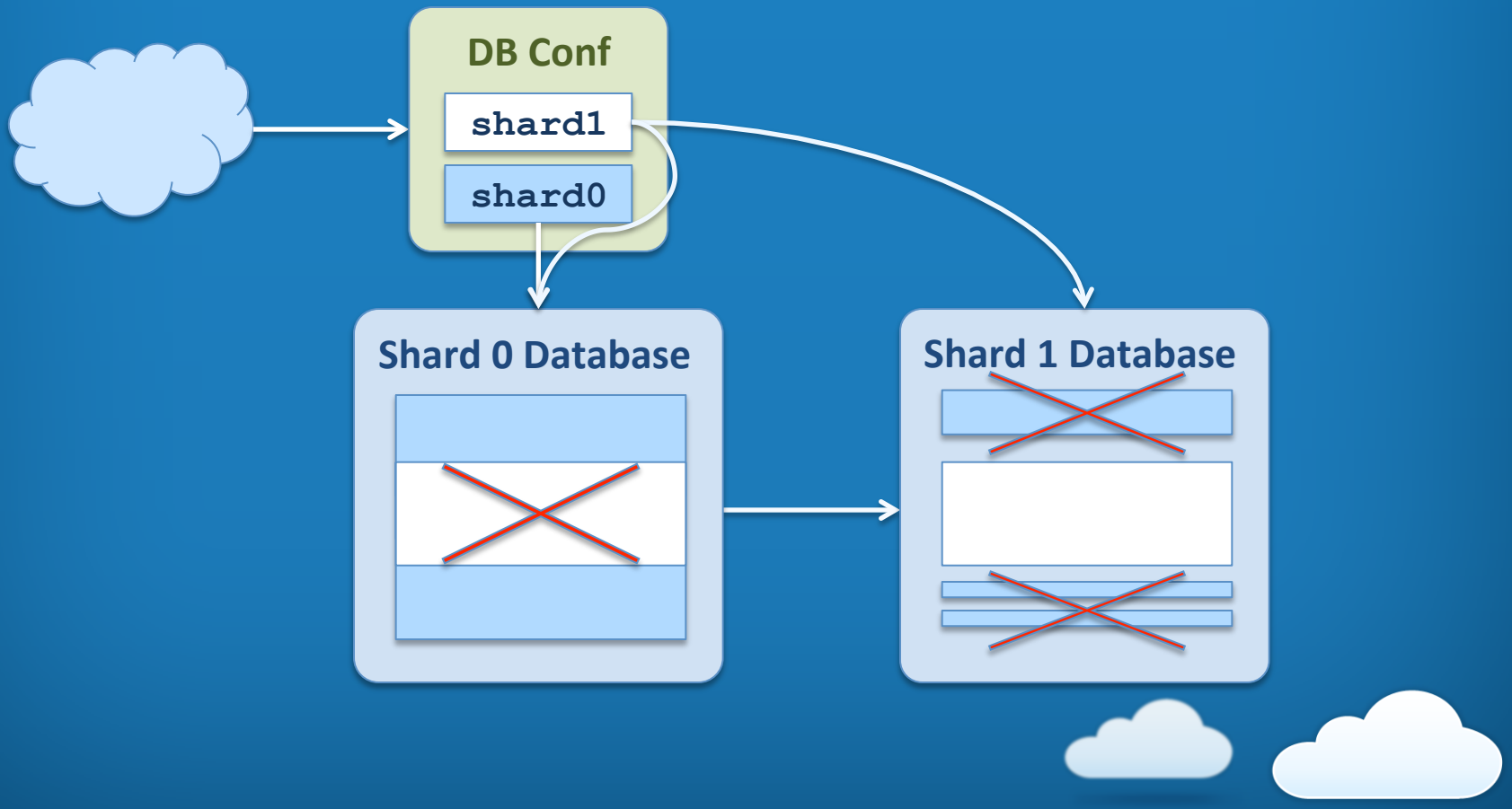
- Our DB-OPS team built an amazing tool to process and visualize slow query logs across multiple dbs
- Check out github.com/box/Anemometer
- Each query that our code generates has a comment appended with the backtrace, db name and host for easy drill-down

Transitioning to Physical Shards

… Without Breaking the Site

Migrating the Data

- Here too, we leveraged logical sharding



Use Cases for Shard Split

- Gradually rolling out shards
- Splitting “hot” shards
- Merging “cold” shards
- Moving rows between live shards

} These were in our reqs

- Using one well-tested process is awesome
- Built-in verification steps in our automated migration scripts saved us on several occasions…

Finally Rolling it Out…

Our Biggest Bug

- After months of work and validation, we were finally ready to move our own accounts to **`shard1`**…
- After marking the mapping rows to 1 we started seeing duplicates of all folders
- Because of collaborations, the query was executed on both **`shard0`** and **`shard1`** – but they pointed at the same db, so double results were returned
- We quickly pushed out a change to de-dup db handles before executing queries

Our Biggest Bug Take 2

- We attempted the migration again, and this time, phase 1 passed cleanly
- We set up replication, and then pushed the conf change to switch **`shard1`** traffic to the new db
- And again…. Duplicate folders!
- This time, **`shard0`** and **`shard1`** were different, but the **`shard1`** rows that had been replicated from **`shard0`** were still there – these were the duplicates

Solution

- Extended ORM to add filtering clause to queries so only "real" results returned
  - e.g., for a query executing on **`shard1`**, add
    - … **`AND {mapping_key} IN ({mapping_ids_on_shard1})`**

Takeaways:

- Test all stages of your roll-out! We focused only on fully **`shard0`** and fully **`shard1`**
- Dogfood-ing is better than breaking live ☺

Summary

Where we are today

- File and folder table partitioned across four database servers with just over 30 shards
- 60% of queries offloaded from main database
- Hundreds of millions of queries a day to the mapping database with 95% being reads
- Folder table more than 3x since we started
- File table in the billions of rows
- Sharding of additional tables underway

Lessons Learned

- **Invest in design:** You are going to have surprises; a solid design will hold up and save you a rewrite
- **Set clear goals:** You are going to be tempted; clear goals will help you push back on nice-to-haves
- **Plan to be incremental:** Incremental changes help you minimize bugs and maximize stability
- **Plan to be extensible:** Code is never "done"; might as well plan ahead…
- **Don't solve bottlenecks you don't have:** Guessing future bottlenecks is impossible, and encourages building unnecessary complexity into your systems
- **Cooperate:** Dev + OPS == success!!

Thank you!

tamar@box.com @TamarBercovici