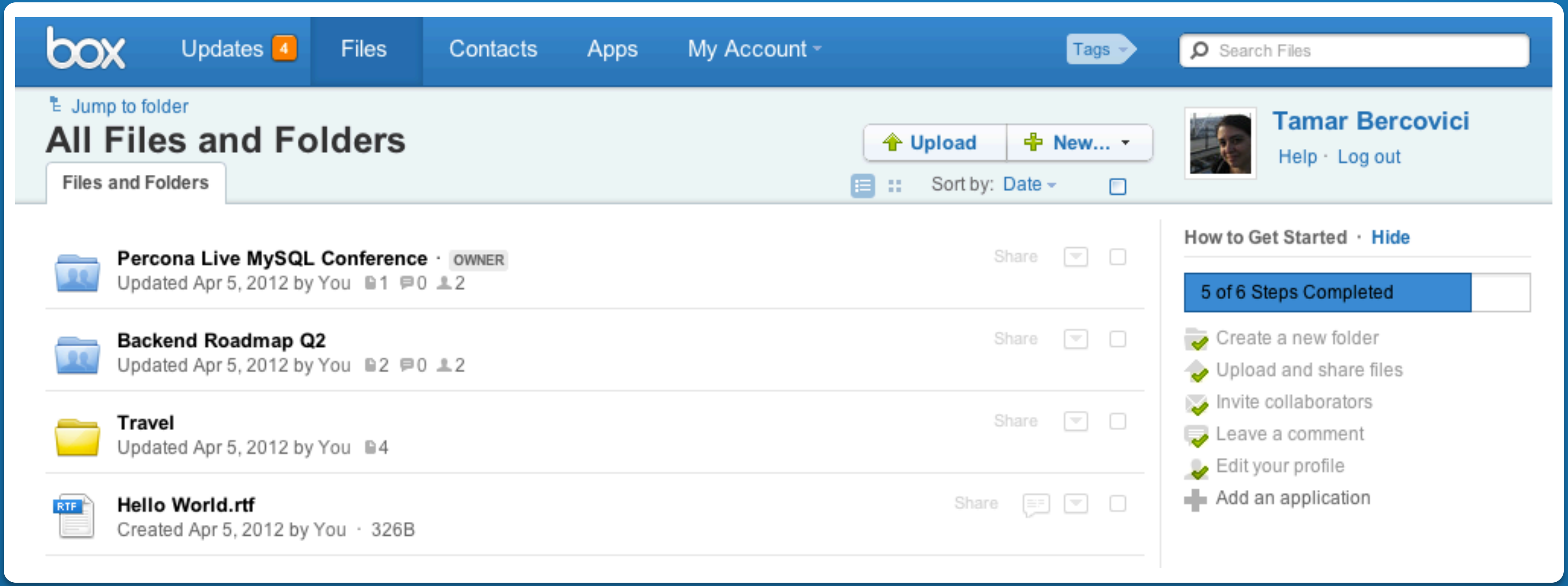
One to Many: The Story of Sharding at Box

Tamar Bercovici
Florian Jourda

What is Box?

- Cloud storage and collaboration for enterprises



All Files and Folders

Percona Live MySQL Conference · OWNER
Updated Apr 5, 2012 by You 1 0 2

Backend Roadmap Q2
Updated Apr 5, 2012 by You 2 0 2

Travel
Updated Apr 5, 2012 by You 4

Hello World.rtf
Created Apr 5, 2012 by You · 326B

Where we started off

- 800K line codebase mainly in PHP
- Full application DB on one MySQL 5.5 instance with cross data-center M-M replication
- Approximately 1.7B queries a day
- Limited reading from slaves; most reads and all writes go to primary master
- Millions of users
- Folder table tens of millions of rows
- File table hundreds of millions of rows
- Aggressive growth expected

Designing a Sharded Architecture

Pain Points and Constraints

Pain Points:

1. Key tables were getting too big to manage
2. Both read and write QPS were growing fast
3. All DB interaction was centralized – SPOF

Constraints:

1. Get to a position where we are more resilient to scale ASAP
2. Transition to the new architecture with no downtime

Goals

1. Transition to an initial sharded architecture that addresses our immediate pain points
   - Introduce as little complexity as possible
   - Make sure the architecture is extensible
2. Make code changes that are backwards compatible and incremental
   - How do we do this?!
3. Roll out with a small number of shards so that we can ramp-up on monitoring
   - Support for gradually adding shards

Unique Box Challenges

1. Any two users can collaborate
   – Users may have access to content on multiple shards

   → **Support for cross-shard queries**

2. Users can move content into a collaborator's folder

   → **Support for online moving of content between shards**

3. Users can continually create more content
   – Shards continue to grow, even if no new accounts are mapped to them

   → **Support for splitting shards**

Sharding Strategy

What to partition?

- Partition file and folder
- User table not partitioned (yet) to reduce complexity

How to partition?

- Partition by user
- Co-locate users belonging to the same enterprise

How to locate content?

- Maintain a Mapping DB that maps IDs to shards
  - Allows for moving data between shards
  - Allows for splitting of shards

Mapping Database

- Maintains a mapping of object ids to shard ids
- MySQL DB, separate from the application database
- We considered other options for the mapping database, but opted for MySQL because it was straightforward to rollout and fit our needs
- High reads / low writes (mainly inserts)
- Very lightweight tables
- All reads are PRIMARY INDEX lookups

Object IDs

Goals:

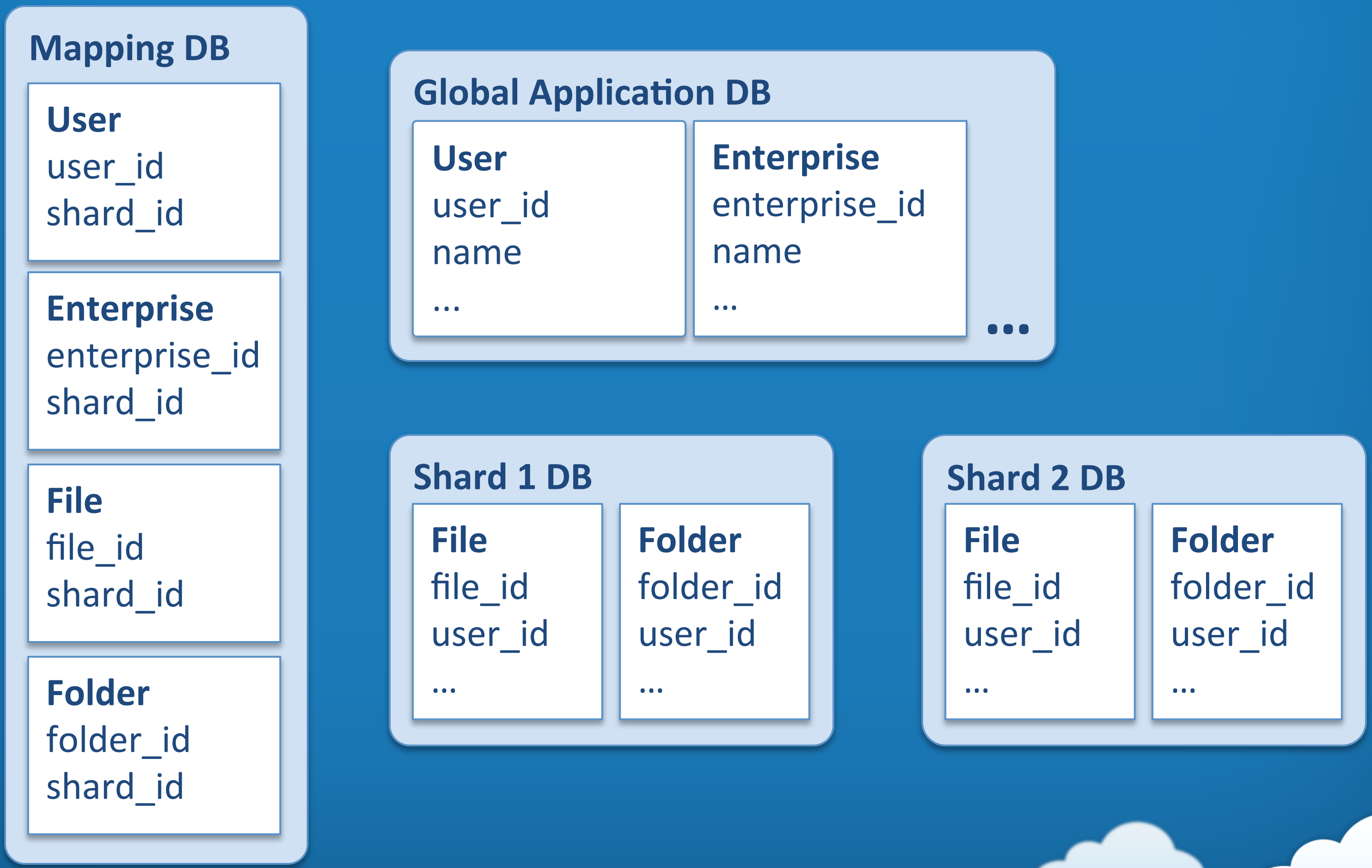
- IDs should be **unique** across the system
  - We need to move / aggregate data across shards
- IDs should be **constant** for an object's lifetime
  - So that reporting and tracking doesn't break
- ID scheme needs to be **backwards compatible**
  - We can't change IDs of existing objects

ID Generation

- Use the mapping database to distribute IDs using MySQL auto-increment
- Every new object (file, folder…) is first added to the mapping, and then to the shard
- Backfill the mapping with existing content

  → New IDs do not overlap with existing IDs
- This ended up being another advantage of using MySQL for our mapping database

Schema Layout



Mapping DB

User
user_id
shard_id

Enterprise
enterprise_id
shard_id

File
file_id
shard_id

Folder
folder_id
shard_id

Global Application DB

User
user_id
name
…

Enterprise
enterprise_id
name
…

…

Shard 1 DB

File
file_id
user_id
…

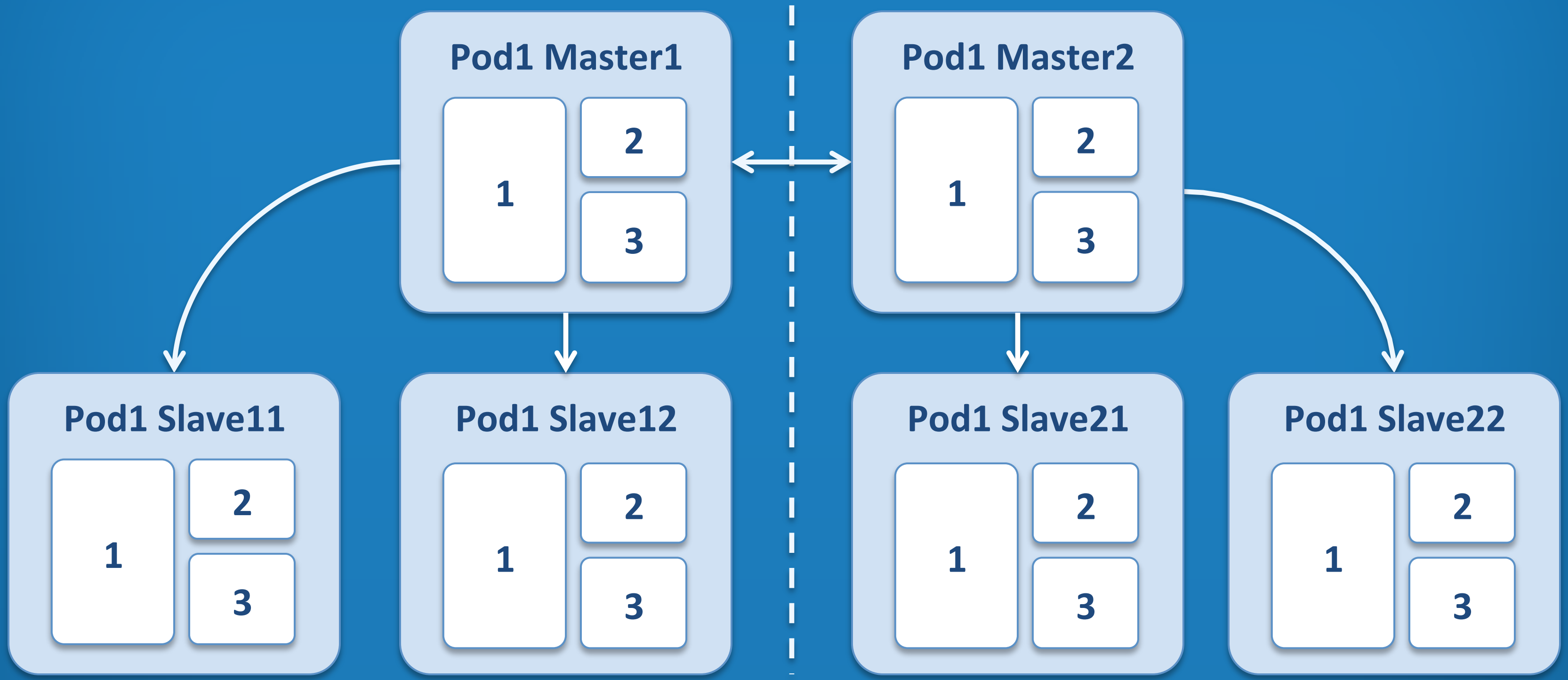
Folder
folder_id
user_id
…

Shard 2 DB

File
file_id
user_id
…

Folder
folder_id
user_id
…

Physical Layout of Shard Pod



- Multiple shards per MySQL instance on a server
- M-M replication across datacenters
- M-S replication within datacenters

Transitioning the Application to Work with Shards

Determining the Shard

- Sharded classes define **mapping keys**
  - Used to look up the shard id in the mapping
  - e.g., for file the keys are file_id, folder_id and user_id
- Queries are analyzed by the ORM layer to find a reference to a mapping key
  - Very easy for ORM queries such as `$folder->children()`
  - More complicated queries require minimal parsing
  - The framework supports passing “hints”
- Shard ids are locally cached within a request

Querying the Shards

- To execute a SELECT / UPDATE / DELETE query:

1. Analyze query and find a mapping key
2. If shard ids not already in cache, query the mapping
3. Execute the query on the all the referenced shards
4. Combine results if necessary and return

- To execute an INSERT query:

1. Allocate shard for new content
2. Insert into mapping db and obtain ID
3. Insert into shard db

Moving Data Between Shards

- Sometimes objects need to move between shards
  - e.g., ownership changes for files and folders
- Whenever an object is modified, the ORM layer checks if the value of a mapping key is changing
  - e.g., the user_id of a file is set to a new value
- If it is, the ORM layer checks if the new value is mapped to a new shard, in which case it kicks off an online cross-shard move process

Online Cross-Shard Move

1. Begin transaction on mapping
2. Begin transaction on source shard
3. Begin transaction on destination shard
4. SELECT … FOR UPDATE rows on source shard
5. Update rows in mapping to destination shard
6. Insert new rows to destination shard
7. Delete content on source shard
8. Commit transaction on destination shard
9. Commit transaction on mapping
10. Commit transaction on source shard

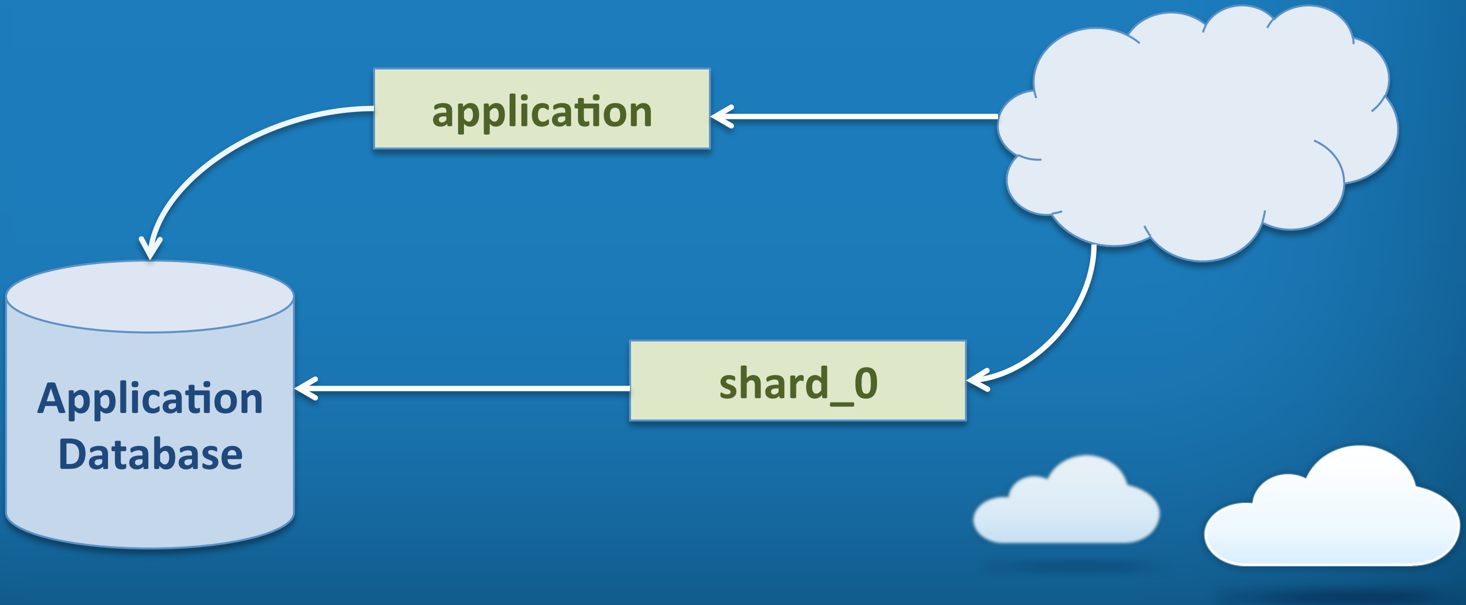
🟦 Mapping

🟩 Source Shard

🟪 Destination Shard

Incremental Roll-out Strategy

- To be able to incrementally roll-out app changes we use **logical sharding**
- We begin by defining the application database as a virtual or logical shard – shard_0



Ramping-up the Mapping DB

- Modify insert code to create mapping entries
  - Insert row into Mapping DB with shard_id = 0
  - Obtain auto-increment id from mapping row
  - Insert row with new id into Application DB / shard_0
- When switching to the new code, avoid id collisions
  - Bump Mapping DB auto-increment table values to be larger than those in Application DB
  - Set Mapping DB auto_increment_offset to be opposite of the one on the Master Application DB (like in M-M)
- Backfill Mapping DB with shard_id = 0 for all rows

Ramping-up Querying Shards

- Log all queries to the app-db that need to be sharded
  - Monitor logs for non-migrated code paths
- Gradually transition code paths to determine the shard_id using the mapping database
  - New code paths go to shard_0; old code paths go to app-db
  - Either way, they will find the data they are looking for
- If the code is unable to determine the shard_id, log an error and default to querying shard_0
  - Monitor logs for code paths to fix
- Throughout the process – functionality is not broken or altered

Automated Testing

- PHPUnit tests were key in our development process
  - Easy testing of narrow code paths on shards, even if the surrounding code is not yet shard-aware
  - When testing interaction with multiple dbs, automatic set up the initial state saves an enormous amount of effort
  - Strong PHPUnit test coverage gave us confidence to tweak our framework code during the course of development
- Existing functional and unit tests were great for verifying we hadn't broken anything
- Added framework support to create content on different shards for focused testing

Physical Shards

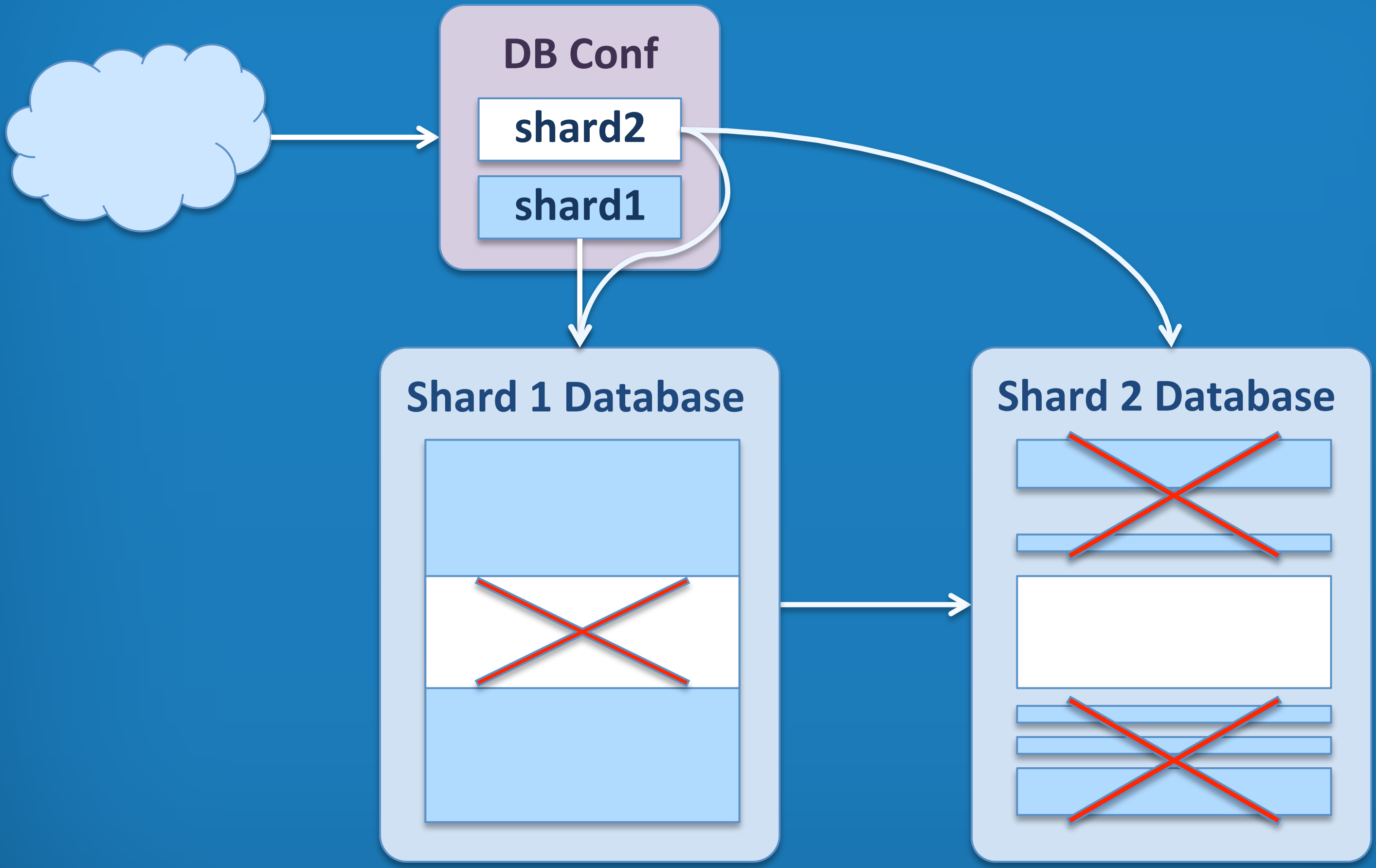
Query Monitoring

- A complex database landscape makes strong automated tools crucial
- Our DB-OPS team built an amazing tool to process and visualize slow query logs across multiple dbs
  - Track bad queries that we introduced
  - Track queries that had shifted from slaves to the master
- When a "bad" query is found, we can easily drill down to the cause: each query has a comment appended with the backtrace, db name and host
- Check out github.com/box/Anemometer

Splitting Shards

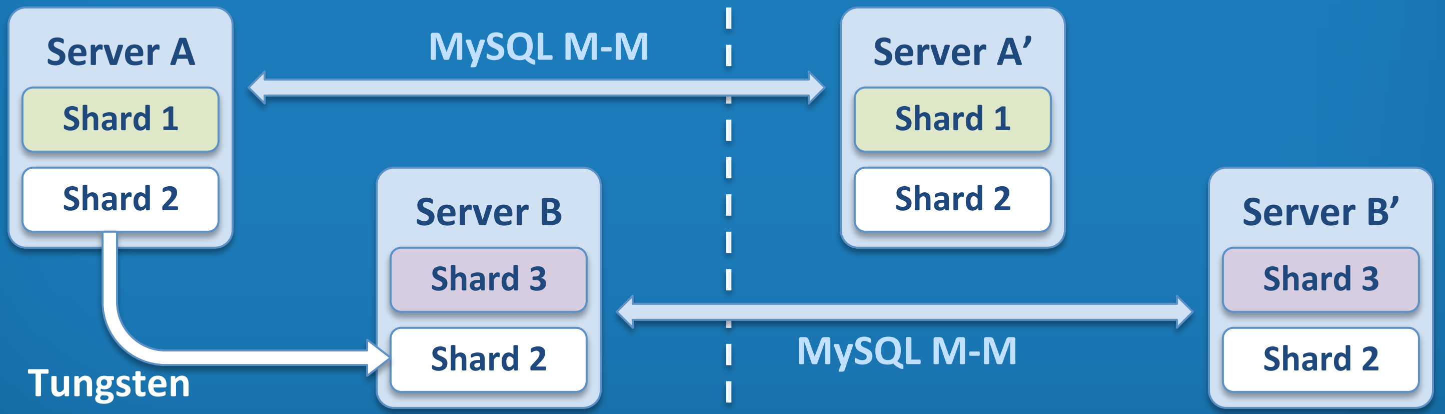
- Here too, we leverage logical sharding
  1. In the DB conf, add a logical shard_id for the new shard and point it at the physical location of the source shard
  2. Gradually update mapping rows for the rows we want to migrate to point at the new shard_id
  3. Copy a consistent snapshot of the rows to migrate to the new database
  4. Set up replication from the source db to the new db
  5. Once replication has caught up, modify the conf entry for the new shard to point at the new database
  6. Once everything stabilizes, kill replication and delete the dead data on both shards

Splitting Shards



Tungsten Replicator

- Our shard split strategy requires a separate replication stream for each shard database
- We use the open-source Tungsten Replicator
  - Supports one replication stream per DB, as opposed to MySQL 5.5 replication which is instance-based
  - Support for db-based replication streams coming in MySQL 5.6



Use Cases for Shard Split

- Gradually rolling out initial set of shards
- Splitting "hot" shards

These were in our reqs

- Moving whole shards between servers for load rebalancing
- Moving rows between two live shards
  - Use a logical shard id different from both source and destination shards
- Sharding additional tables into existing shards
  - Feature flip app code to "turn on" sharding for new table, one shard at a time

Finally Rolling it Out…

Our Biggest Bug

- After months of work and validation, we were finally ready to test on ourselves and move our own accounts from shard_0 to shard_1…
- After marking the mapping rows to 1 we started seeing duplicates of all the folders in our accounts
- Because of collaborations, the folder query needed to execute on both shard_0 and shard_1 – but they pointed at the same db, so the query returned double results
- 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 shard_1 traffic to the new db
- And again…. Duplicate folders!
- This time, shard_0 and shard_1 were different, but the shard_1 rows that had been replicated from shard_0 were still there – these were the duplicates

Solution

- We extended the ORM layer to automatically generate versions of queries so that the query version executing on each shard filtered for results from that shard only
- We filtered based the mapping_key and mapping_ids that had been used to determine the shard to execute on
  - e.g., to a query executing on shard1 we added an additional WHERE clause …

    AND {mapping_key} IN ({mapping_ids_on_shard1})

Takeaways:

- Test all the stages of your roll-out process – we focused only on fully shard_0 and fully shard_1
- Dogfood-ing is better than breaking live ☺

Summary

Where we are today

- First live shard in production and more on the way
- 50% of all application queries go to sharded tables
- Hundreds of millions of queries a day to the mapping with 95% of those being reads
- Folder table more than 3x since we started
- File table in the billions of rows

Going Forward

- Finish removing file and folder tables from Global DB
- Graceful failure modes and failure isolation
- Monitoring / Isolated failover
- Plans to shard additional tables underway…
- Aggressive growth? Bring it on!

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:** Designing both code and physical transitions to be incremental will help you minimize bugs and maximize site stability
- **Plan to be extensible:** You are going to get to phase two; might as well plan ahead…

Thank You!

Tamar Bercovici
tamar@box.com
@TamarBercovici

Florian Jourda
florian@box.com