Migrating to Vitess at (Slack) Scale

Michael Demmer
Percona Live Europe 2017
Welcome!
This is a (brief) story of how Slack's databases work today, why we're migrating to Vitess, and some lessons we've learned along the way.
Michael Demmer
Senior Staff Engineer
Slack Infrastructure

- ~1 year at Slack, former startup junkie
- PhD in CS from UC Berkeley
- Long time interest in distributed systems
- (Very) new to databases
Our Mission: To make people’s working lives simpler, more pleasant, and more productive.
slack

- 9+ million weekly active users
- 4+ million simultaneously connected
- Average 10+ hours/weekday connected
- $200M+ in annual recurring revenue
- 800+ employees across 7 offices
- Customers include: Autodesk, Capital One, Dow Jones, EA, eBay, IBM, TicketMaster, Comcast
How Slack Works

(Focusing on the MySQL parts)
The Components

- Linux
- Apache
- MySQL
- PHP / Hack
- Real Time Messaging
- Caching
The Components

- Linux
- Apache
- MySQL
- PHP / Hack
- Real Time Messaging
- Caching
MySQL Numbers

Primary storage system for the Slack service
(File uploads in AWS S3)

~1400 database hosts

~600,000 QPS at peak

~30 billion queries / day
MySQL Details

- MySQL 5.6 (Percona Distribution)
- Run on AWS EC2 instances, no containers
- SSD-based instance storage (no EBS)
- Single region, multiple Availability Zones
- PHP webapp connects directly to databases
Master / Master

- Each is a *writable* master AND a replication slave of the other
- Fully async, statement-based replication, without GTIDs
- Yes, this is a bit odd... BUT it yields Availability >> Consistency
- App *prefers* one "side" using `team_id % 2`, switches on failure
- Mitigate conflicts by using upsert, globally unique IDs, etc
Sharding Today

Workspace (aka "team") assigned to a shard at signup

App finds team:shard mapping in mains db

Globally Unique IDs via a dedicated service
Added Complexity

Enterprise Grid: Federate multiple workspaces into an org using N + 1 shards

Shared Channels: Keep multiple shards in sync for each workspace
The Good Today

Highly available for both transient or permanent host failures
Highly reliable with low rate of conflicts in practice
Writes are as fast as a single node can accept
Horizontally scale by splitting "hot" shards
Can pin large teams to dedicated hosts
Simple, well understood, easy to administer and debug
Challenges
Hot Spots

Large customers or unexpected usage concentrates load on a single shard

Can't scale up past the capabilities of a single database host
Application Complexity

Need to know the right context to route a query

No easy way to shard by channel, user, file, etc
Inefficient Usage

Average load (~200 qps) much lower than capacity to handle spikes

Very uneven distribution of queries across hosts
Operator Interventions

Operators need to manually repair conflicts and replace failed hosts.

Busy shards are split using manual processes and custom scripts
So What To Do?
Next Gen Database Goals

- Shard by Anything!
- Easy Development Model
- Highly Available (but a bit more consistent)
- Efficient System Utilization
- Operable In Slack's Environment
# Possible Approaches

<table>
<thead>
<tr>
<th>Shard by X in PHP</th>
<th>NoSQL</th>
<th>NewSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ no new components</td>
<td>+ flexible sharding</td>
<td>+ flexible sharding</td>
</tr>
<tr>
<td>+ easiest migration</td>
<td>+ proven at scale</td>
<td>+ scale-out storage</td>
</tr>
<tr>
<td>- development effort</td>
<td>- major change to app</td>
<td>+ SQL compatibility!</td>
</tr>
<tr>
<td></td>
<td>- new operations burden</td>
<td>- some new ops burden</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- least well known</td>
</tr>
</tbody>
</table>
Vitess In One Slide

Credit: Sugu Sougoumarane <sougou@google.com>
Why Vitess?

- NewSQL approach provides the scaling flexibility we need without needing to rewrite the main application logic
- MySQL core maintains operator and developer know-how
- Proven at scale at YouTube and others
- Active developer community and approachable code base
Shard by Anything

- Applications issue queries as if there was one giant database, Vtgate routes to the right shard(s)
- "Vindex" configures most natural sharding key for each table
- Aggregations / joins pushed down to MySQL when possible
- Secondary lookup indexes (unique and non-unique)
- Still supports inefficient (but rare) patterns: Scatter / gather, cross-shard aggregations / joins
Easy Development Model

- Vitess (now) supports the mysql server protocol end to end
- App connects to any VtGate host to access all tables, specifying a different "database" for master or replica
- Most SQL queries are supported (with some caveats)
- Additional features: connection pooling, hot row protection, introspection, metrics
Highly Available (and more consistent)

- Vitess topology manager handles master / replica config
- Actual replication still performed by MySQL
- Changed to row-based, semi-sync replication using GTIDs
- Deployed Orchestrator to manage failover in seconds
Efficient System Usage

• Vitess components are performant and well tuned from production experience at YouTube
• Can split load vertically among different pools of shards
• Even distribution of fine grained shard keys spreads load to run hosts with higher average utilization
Operable in Slack's Environment

- MySQL is production hardened and well understood
- Leverage team know-how and tooling
- Replication still uses built-in mysql support
- New tools for topology management, shard splitting / merging
- Amenable to run in AWS without containers
Vitess Adoption: Approach and Experiences
Migration Approaches

Migrate individual features one by one

Run Vitess in front of existing DBs
Migration Approaches

Migrate individual features one by one ✓

- Only approach that enables resharding (for now)

Run Vitess in front of existing DBs 🔴

- Could make it work with custom sharding scheme in Vitess
- But we run master/master
- And doesn't help to avoid hot spots!
How to Migrate a Feature

• For each table to migrate:
  1. Analyze queries for common patterns
  2. Pick a keyspace (i.e. set of shards) and sharding key
  3. Double-write from the app and backfill the data
  4. Switch the app to use vitess

• But we also need to find and migrate all joined tables
  ... and queries that aren't supported or efficient any more
  ... and whether the old data model even makes sense!!
VtExplain

- vtexplain -- an offline analysis tool that shows what actually runs on each shard
- Vitess' query support is not yet (likely never be) 100% MySQL
- Choice of sharding key is crucial for efficiency
Migration Backfill

- Enable double-write in the app
- Backfill scan loop
  
  LOCK TABLES `<table>` READ
  SELECT * WHERE ... LIMIT `<batch>`
  INSERT IGNORE ...
  UNLOCK `<table>`
  SLEEP
  (Adjust batch size based on lock time)
- Then enable dark reads / writes and compare for a while
Current Status

🎉 Deployed in production for one feature (~1% of all queries)

- More migrations & new features that depend on Vitess sharding
- Ported or redeveloped existing processes for managing clusters
Current Status: Details

• ~2000 QPS, about 50/50 read vs write
• 4 shards, 3 replicas per shard, 8 vtgate hosts
• Ported most operations processes, but still automating many processes
• Decent performance overall with occasional hiccups that require investigation (seemingly due to infrastructure)
Performance

Millisecond latencies for connect/read/write

Vitess is more network bound, so things are slower

No significant performance issues with Vitess components (so far)
Vitess Deployment: Multi AZ
Vitess Deployment: Multi AZ

Diagram showing the deployment of web apps, Elastic Load Balancer, and vtgate nodes across different availability zones (us-east-1a, us-east-1b, us-east-1d, us-east-1e) with master and replica databases.
AZ-Aware VTGate Preference

us-east-1a
- web app
- vtgate
- replica

us-east-1b
- web app
- vtgate
- master

us-east-1d
- web app
- vtgate
- replica

us-east-1e
- web app
- vtgate
- replica
Sub-Cell (Future)

us-east-1a
- web app
- vtgate
- replica

us-east-1b
- web app
- vtgate
- master
- replica

us-east-1d
- web app
- vtgate
- replica

us-east-1e
- web app
- vtgate
- replica
High Level Takeaways
Because of Vitess, we had to:

switch to master / replica...
using semi-sync with gtid...
with orchestrator for failover...
and start reads from replicas...

But at the same time, we:

switched to row based replication...
on mysql 5.7 on new i3 EC2 hosts...
and an updated Ubuntu release...
using hhvm's async mysql driver...
First Query is the Hardest

- Migration exposed latent bugs in our app
- Each of the various changes caused some glitch or another
- Double read differences: Vitess or our existing system?
- Instrumented and tuned (a lot) to gain confidence
- Still learning as we go
Networking Matters

- Vitess is intrinsically more network dependent than our existing database architecture
- Performance depends (a lot) on network quality
- Complicated to track down and diagnose
- Able to work around some issues by kernel tuning, host placement, application routing to vtgate
Vitess: Build and "Buy"

- The core of Vitess is stable, performant, and robust
- Yet each new adoption finds missing or unexpected features around the edges
- Ecosystem is still small but growing as interest spreads beyond YouTube
- Active developer community: [github.com/youtube/vitess](http://github.com/youtube/vitess), [vitess.slack.com](http://vitess.slack.com)
"Vitess is magical but not magic"

🤔 Can't (yet) use familiar tools like phpmyadmin
❗️ Besides MySQL, there are a still lot of new moving parts
😢 No ability (yet) to change sharding key
🚫 Unsupported queries
⚠️ Efficiency requires stale reads from replica
📈 Gained consistency, but reduced availability and performance
🔍 Documentation!! -- many, many options to understand
Vitess At Slack: Thriving

Running in production after ~7 months of effort
Active contributor to developer community
Stable and performs as expected, but more to go
Leadership buy in as the future approach for Slack Databases

We have a long but exciting road ahead...

=*/ And we are hiring! */
Thank you!