There And Back Again

Databases At Uber

Evan Klitzke

October 4, 2016

Outline

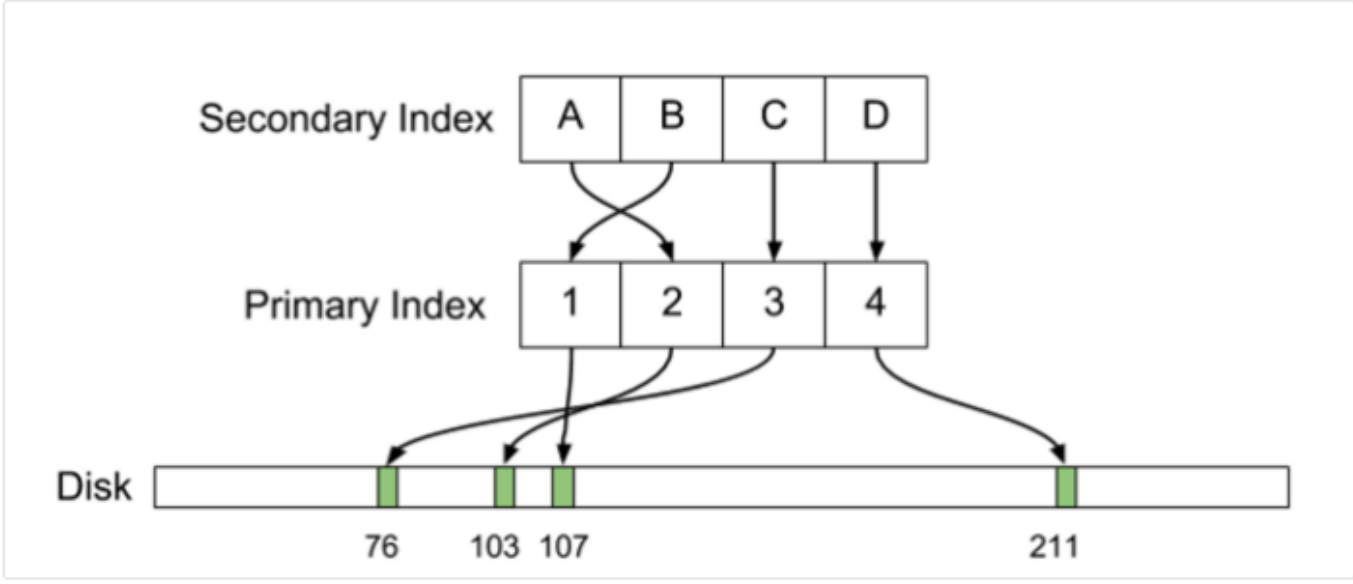
Uber Engineering Blog Post

ARCHITECTURE

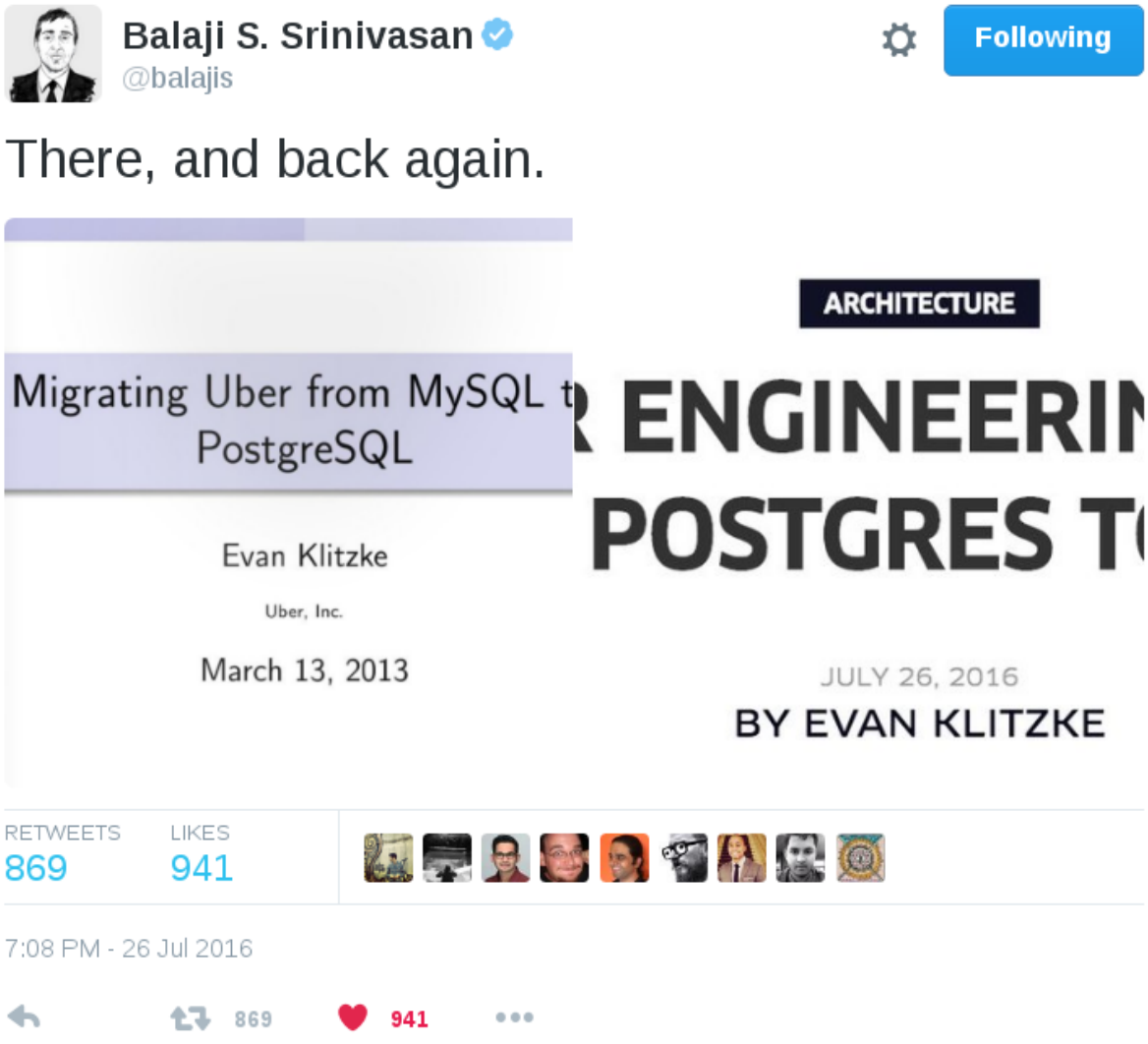
WHY UBER ENGINEERING SWITCHED FROM POSTGRES TO MYSQL

JULY 26, 2016

BY EVAN KLITZKE



Thanks Balaji



Outline

Company Background

When I joined in September, 2012:

- Company was about two years old
- About 30 engineers
- No DBA, only one person with an ops background
- I happened to be the person with more of a database background than anyone else, but I wasn't an DBA

Database Background

When I joined in September, 2012:

- Monolithic MySQL cluster
- One MySQL master, three or four slaves
- Examples of stored data: trips, user data, driver info, city data, fare structures, geofence information, toll data, etc.
- Database was ~ 500 GB in size
- Bare metal hardware in a colocation facility; 15K SAS drives, lots of memory/CPU

Why We Wanted To Switch

Reasons:

- Technical
- Non-technical

PostGIS

Uber is an incredibly geo-centric company, and we had lots of geo data in the database:

- Trip points
- Geofence definitions
- Toll/tunnel definitions
- Etc.

There were also lots of ideas on new features that would heavily use geospatial information in the app, and it was thought that PostGIS would help.

Online Schema Changes

We were making a **lot** of schema changes.

For better or worse, there was a strong culture of shipping code quickly, sometimes without proper design. This led to a rapidly evolving schema and rapidly evolving index requirements.

Hacker News Hype Cycle

MySQL is the database everyone loves to hate:

- Various poor default settings (esp. in older releases)
- Sordid history of replication bugs
- Somehow gets associated with PHP?
- Late to the game on “sexy” features like geospatial, JSON types, table inheritance, etc.
- Oracle is “evil”

People Haven’t Heard Of Postgres?

forinti 3 hours ago [-]

In my experience, most people who used MySQL simply hadn't heard of PostgreSQL (which isn't hard to install either and probably performed well enough for everybody's requirements - at least everybody I talked to).

I used to ask, because I found it baffling that someone would choose an RDBMS that wouldn't even check the size of a column.

MySQL Eats And Corrupts Data By Design

viraptor 62 days ago [-]

Mysql eats and corrupts data by design. For a company responding to real time events in physical world, that can be a big issue. I know they're trying to improve their defaults lately, but a lot of weird behaviour remains. And you don't have to be an expert DBA to know that choosing a technology known for silent data corruption is risky.

My Favorite HN Comment Ever

DiabloD3 62 days ago [-]

For the kind of stuff Uber stores, they may actually be doing it wrong (given what that Postgre mailing list post says) because that is one hell of an ugly use case for *any* DB.

I would have tried solving it by loading a dual E5v4 server full of 3TB and a slew of SSDs for L2ARC+ZIL under ZFS: more SSDs > bigger SSDs because the absolute worst case SSD performance that *any and all* SSDs suffer from is random *reads* (not writes) can slow to 200MB/sec even on those insanely fast "enterprise" PCI-E SSDs that do 2-4GB/sec continuous reads.

Given that, there are only four dbs worth using: Postgre, Oracle, DB2, and MS SQL. Unless you're doing something that is truly not suited for SQL (or SQL is insanely overkill), or you're Google and have a database in the hundreds or thousands of TB and literally have to write your own database (they went from almost inventing mapreduce, to going full circle back to full scale distributed RDBMS SQL with F1, which I wish they'd open source), any other database is just going to be a pain in the ass, buggy, and full of gotchas and undiscovered corner cases simply because it doesn't have over a decade of development and millions of users behind it.

Also, I have not included MySQL for obvious reasons: fun for toy SQL DBs where sqlite isn't a good fit, but not for enterprise use by any means. Uber switching to MySQL over Postgre is rather scary, I wouldn't want to be a Uber investor right now.

Hindsight Is 20/20

In retrospect, a lot of the motivation for moving to Postgres was fueled by this kind of hype.

No one at Uber actually had experience running Postgres at scale and we didn't do realistic load testing before switching.

Why I’m Telling You This

When you’re moving quickly it’s easy and tempting to take technical shortcuts and rely on hearsay. It happens to the best of us.

It’s incumbent on all of us, as engineers, to be objective, skeptical, and fact-driven.

Don’t believe the hype.

Outline

Connection Scalability

From `wiki.postgresql.org`, “Number Of Database Connections”:

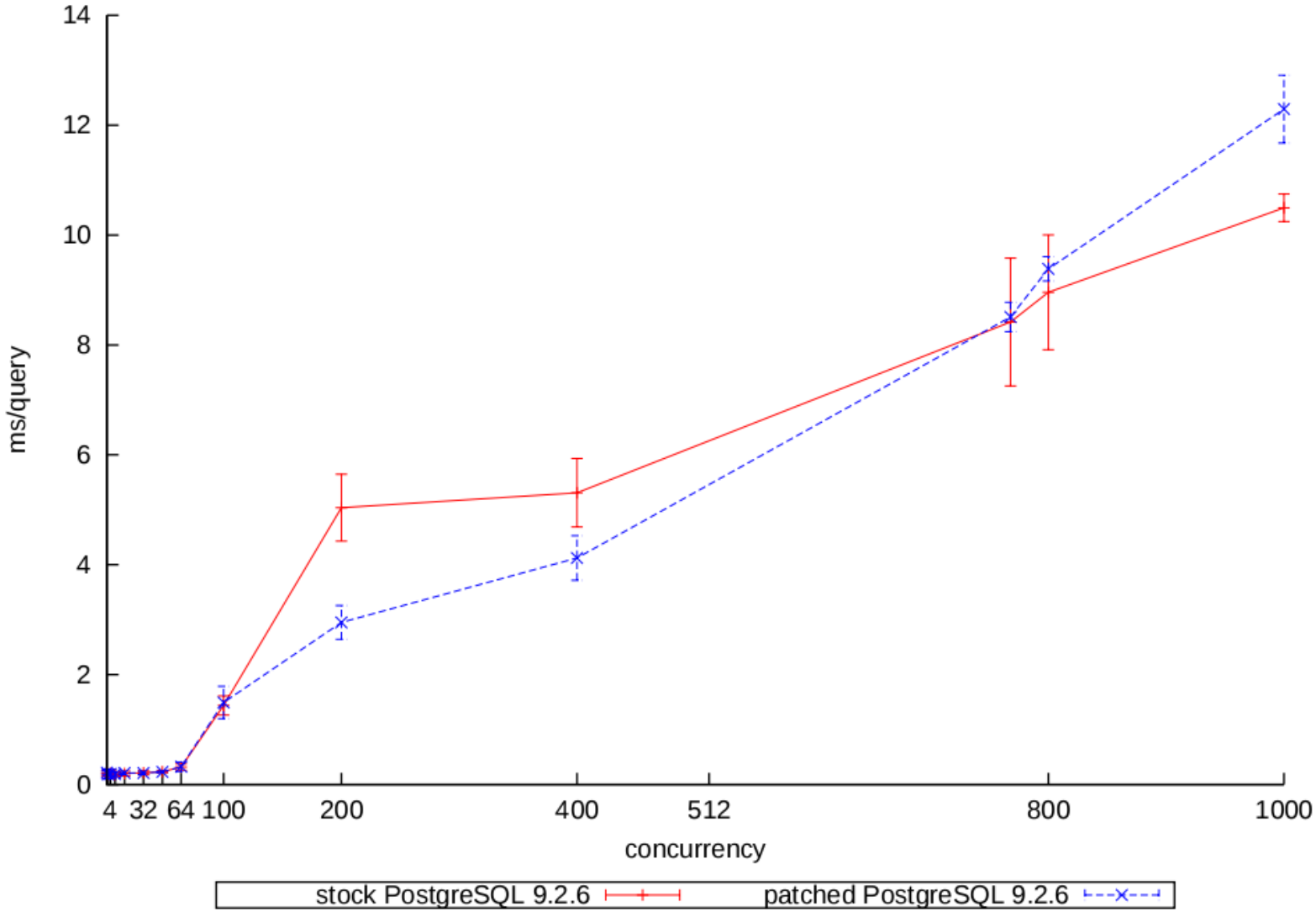
“A formula which has held up pretty well across a lot of benchmarks for years is that for optimal throughput the number of active connections should be somewhere near ((core_count * 2) + effective_spindle_count). Core count should not include HT threads, even if hyperthreading is enabled.”

Connection Scalability

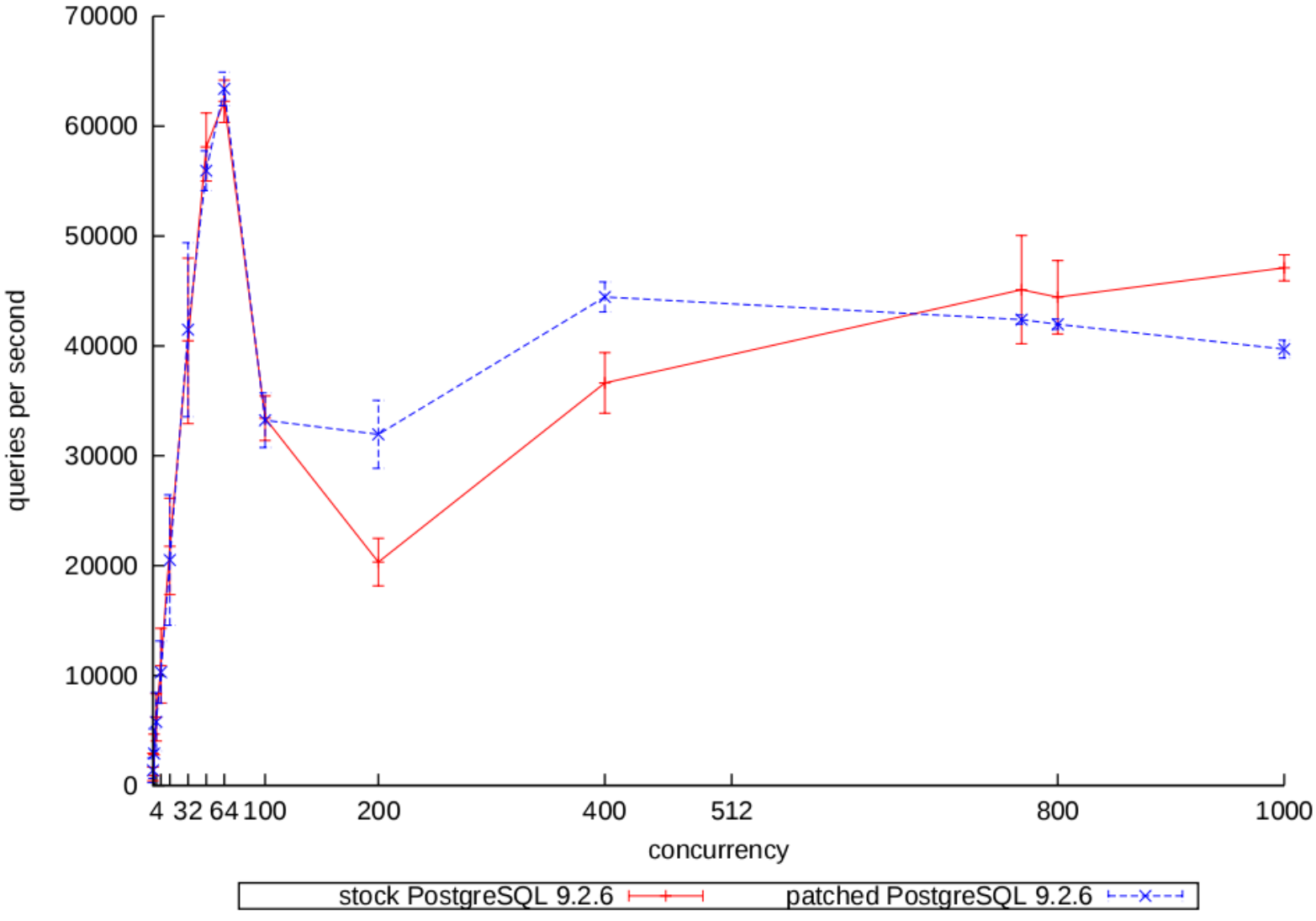
From `wiki.postgresql.org`, “Tuning Your PostgreSQL Server”:

“Generally, PostgreSQL on good hardware can support a few hundred connections.”

Milliseconds/Query

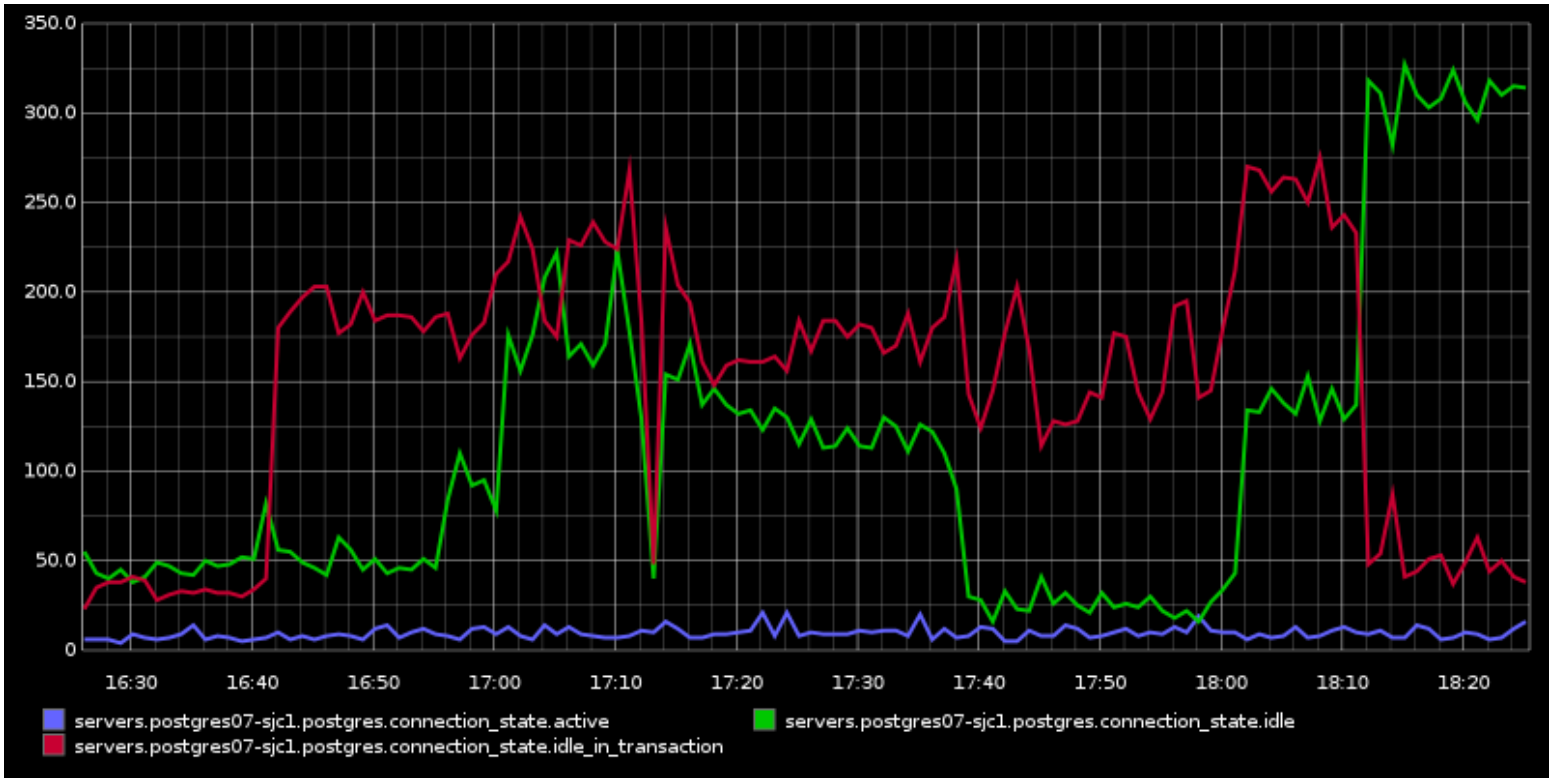


QPS


70000
60000
50000
40000
30000
20000
10000
0
queries per second
4
32
64
100
200
400
512
800
1000
concurrency
stock PostgreSQL 9.2.6
patched PostgreSQL 9.2.6

A Real Uber Outage

This graph is from an outage in November, 2014:



Outline

How Postgres Replication Works

Postgres maintains a write-ahead log (WAL) for crash recovery purposes. The WAL contents are very low-level, representing actual on-disk changes.

Slaves stream WAL files from the master and apply them as if they’re in crash-recovery.

How Postgres Row Updates Work

Row “tuples” in Postgres are immutable.[1]

A tuple is updated by creating a new copy at a new location. Later the autovacuumer will reclaim old tuples.

[1] Except for “heap-only tuples” (HOT)

How Postgres Indexes Work

Index entries have a reference to the `ctid` of the row they index.

Pro: Index lookups are very efficient!

Con: All indexes must be updated when rows are updated, because the new row tuples will have new ctids.

An Example

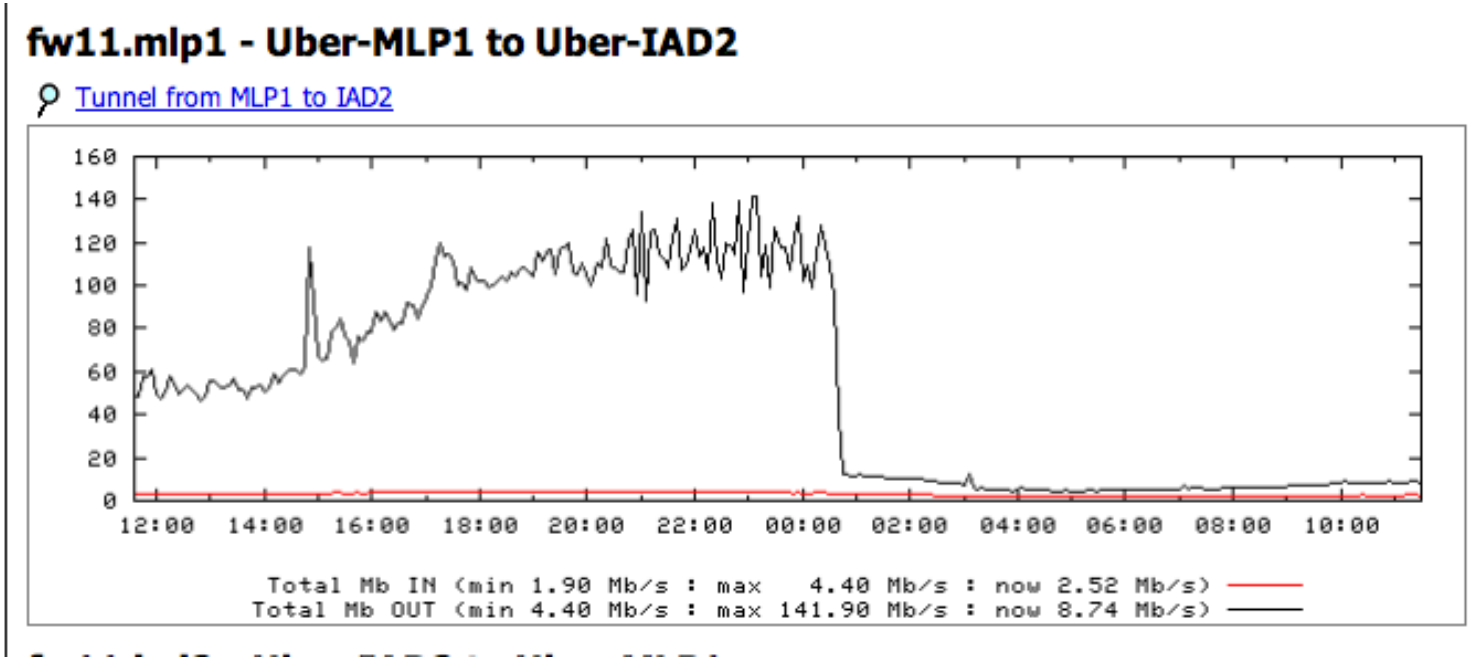
Suppose you have a `users` table with ten columns, five of which are indexed.

If you change the `email` for a cell you might expect the following updates: one to create a new tuple, and one to update the `email` index.

In fact you'll need: one write to create a new tuple and updates for **each** index, since the new tuple has a new `ctid`. All of these physical changes are propagated into the WAL.

Replication Bandwidth

Guess what happened here (February, 2014):



WARM

Postgres developers are working on this issue :-D

Patch: Write Amplification Reduction Method (WARM)

From:	Pavan Deolasee <pavan(dot)deolasee(at)gmail(dot)com>
To:	pgsql-hackers <pgsql-hackers(at)postgresql(dot)org>
Subject:	Patch: Write Amplification Reduction Method (WARM)
Date:	2016-08-31 16:45:33
Message-ID:	CABOikdMNy6yowA+wTGK9RVd8iw+CzqHeQSGpW7Yka_4RSZ_LOQ@mail.gmail.com (view raw)

```
Hi All,

As previously discussed [1], WARM is a technique to reduce write
amplification when an indexed column of a table is updated. HOT fails to
handle such updates and ends up inserting a new index entry in all indexes
of the table, irrespective of whether the index key has changed or not for
a specific index. The problem was highlighted by Uber's blog post [2], but
it was a well known problem and affects many workloads.
```

Outline

Postgres Upgrades

How do you upgrade Postgres 9.2 to something newer?

Buffer Pools vs. Kernel Page Cache

Reasons MySQL's "buffer pool" design is great:

- More efficient
- Not tied to on-disk format
- Fewer context switches
- Allows more efficient buffered writes
- Cool statistics, e.g. hit rate

Outline

Databases at Uber today

We still have some legacy Postgres 9.2 databases. There is no active effort to convert these.

New database clusters are:

- Regular MySQL if sharding and bidirectional replication are not needed
- “Schemaless” MySQL if sharding or bidirectional replication are required
- Cassandra for certain other sharded use cases

PostGIS

PostGIS is definitely cool, but it has a **lot** of caveats and is too slow for large-scale OLTP use cases.

For nearly all of the cases where we thought we wanted PostGIS, we ended up using alternative solutions that were customized for the exact use case and faster.

Schema Changes

Schema changes are a solved problem:

- Percona has `pt-online-schema-change`
- GitHub has `gh-ost`

They're both great, pick one and use it.

Contact

Let me know if you think I'm wrong, if you have feedback, or just want an opinionated database friend:

Twitter: @eklitzke
Email: evan@eklitzke.org

This slide deck is online at:
https://github.com/eklitzke/percona-live-2016