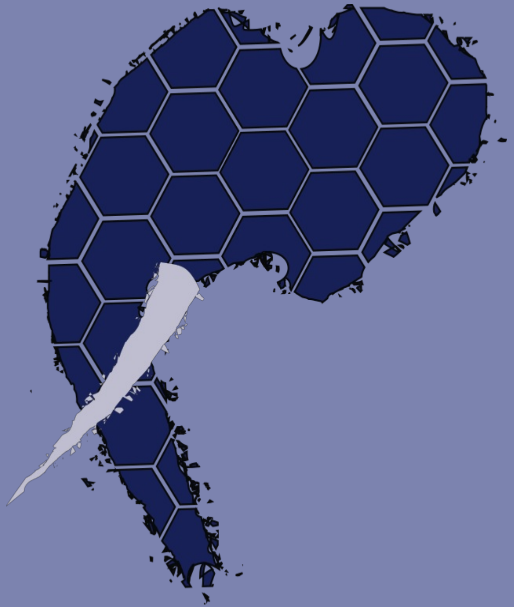
DATA OF FUTURE PAST

POSTGRES AS DISTRIBUTED ONLINE PROCESSING ANALYTICS ENGINE



by Gavin McQuillan / @gmcquillan

SETTING

Data Engineering at Urban Airship, a mobile messaging company:

- Counting lots of things as fast as possible
- HBase to the rescue
- Home grown dimensional storage called datacube

POSTGRES AS DISTRIBUTED ONLINE PROCESSING ANALYTICS ENGINE

1. Problem Statement
2. Distributed Postgres
3. Probabalistic Datastructures
4. Benchmarking Solutions, Unloaded/Loaded.

THE PROBLEM

- Data consistency
- New dimensions multiply writes
- Double counting
- Changing schema is hard
- Consistent backups?

EXPLORING SOLUTIONS

Postgres is pretty nice to work with.

Makes adhoc analytics simple.

Well known replication and backup story

PROBLEMS WITH POSTGRES

Not particularly good at scaling writes horizontally

Operationally complex

EXISTING SOLUTIONS

- Postgres-xc/xl
- Slony
- Redshift
- Pg_shard
- PLProxy

PLPROXY

- Simple API
- Battle tested
- Flexible
- Easy upgrade paths, no lock-in

APPROACH

Two phase commit

Commutative, Idempotent data

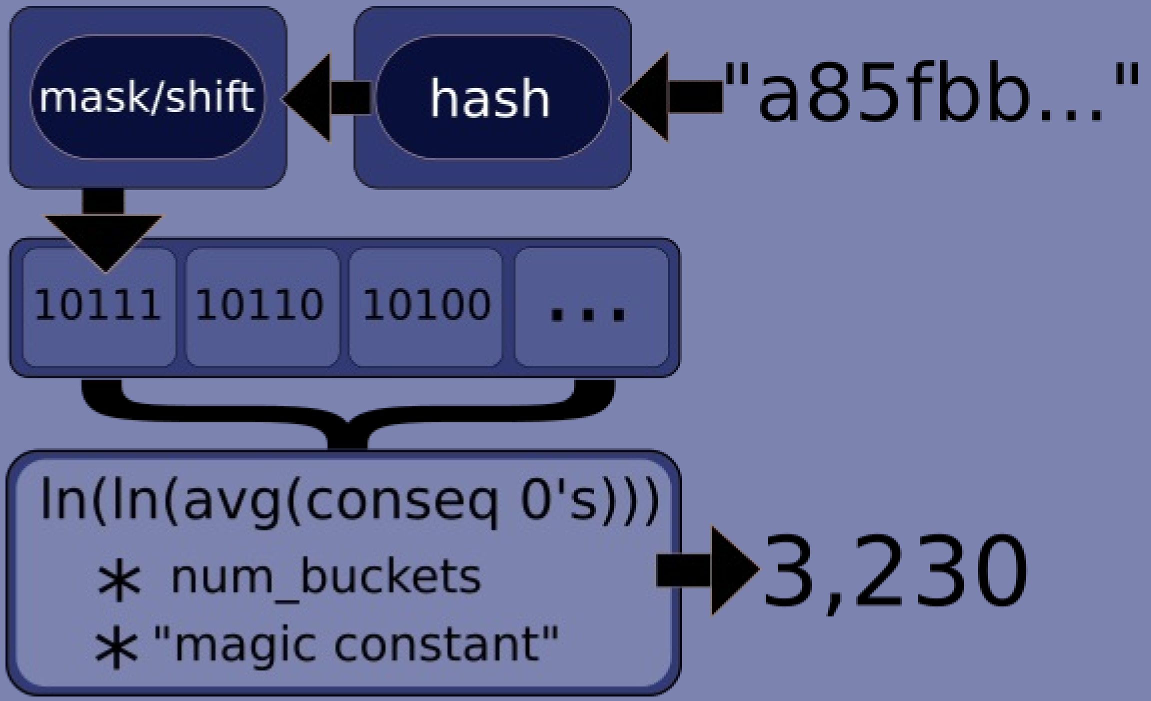
IDEMPOTENT WRITES WITH HYPERLOGLOG

Postgres-hll extension

Commutative, idempotent

Fast, approximate, cardinality

BRIEFLY, HOW HYPERLOGLOG WORKS



PLPROXY: SETTING UP FOREIGN DATA WRAPPERS IN SQL.

CLUSTER CONFIG

Partition defs, cluster version, connection config elided

Partition mapping is as follows:

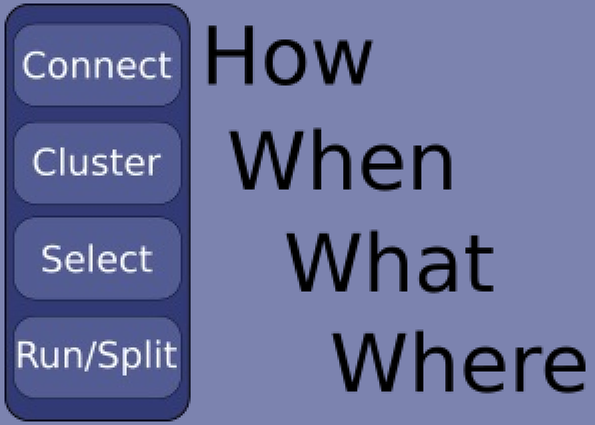
PARTITION MAPPING

```sql
CREATE FOREIGN DATA WRAPPER plproxy;

CREATE SERVER testcounts FOREIGN DATA WRAPPER plproxy
OPTIONS (connection_lifetime '1800',
         p0 'dbname=part00 host=10.130.1.38',
         p1 'dbname=part01 host=10.130.1.39' );

-- This mapping is accessible to all local users
CREATE USER MAPPING FOR PUBLIC SERVER testcounts;
```

PROXY FUNCTIONS

```
CREATE OR REPLACE FUNCTION upsert_count(
        in_id text, in_date date, in_hour smallint,
        in_event_id text, in_category text
) RETURNS TABLE (updates int)
        LANGUAGE plproxy
        AS $$
        CLUSTER 'testcounts';
        SPLIT ALL;
        RUN ON hashtext(in_event_id);
$$;
```

PL SYNTAX EXPLAINED



Connect How

Cluster When

Select What

Run/Split Where

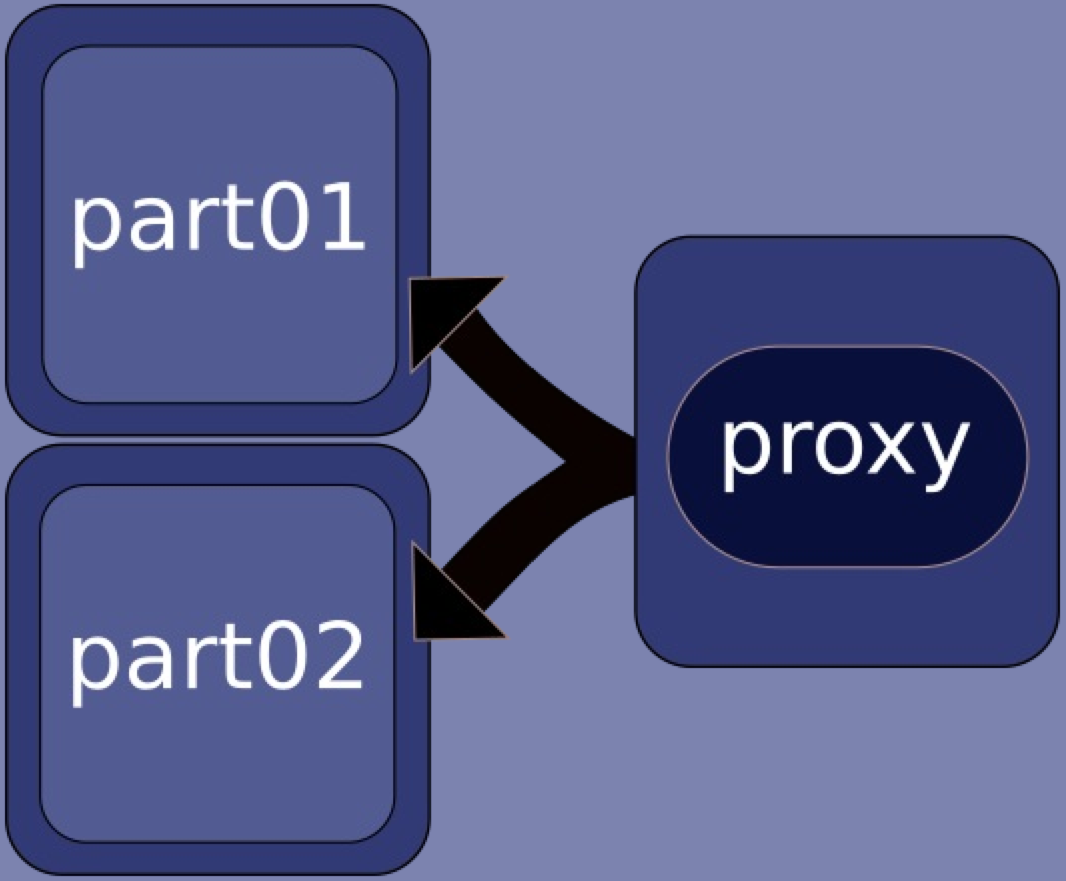
EXPERIMENTAL DESIGN

PHYSICAL LAYOUT

Three Dell R610s with:

- 2 8-core Xeon CPUs
- 6 SSDs in a RAID 10 configuration (~300GB usable)
- write-back cache enabled on the I/O controller
- 48GB of ECC RAM.
- Bonded Ethernet interfaces

SIMPLE TOPOLOGY



SETTING UP THE SHARDS

EXAMPLE TABLE

```sql
CREATE TABLE test_counts
(
        id CHAR(22),
        date DATE,
        hour SMALLINT,
        event_ids hll,
        category TEXT
);
```

SINGLE INSERT/UPDATE

```
CREATE OR REPLACE FUNCTION upsert_test_count(...) RETURNS int
BEGIN
    UPDATE test_counts set event_ids=hll_add(
                        event_ids, hll_hash_text(in_event_id))
        WHERE ...
    IF FOUND THEN RETURN 0; END IF;
    BEGIN
        INSERT INTO test_counts(event_ids, ...)
          VALUES (hll_empty(), ...);
        Update test_counts SET event_ids=hll_add(
                        event_ids, hll_hash_text(in_event_id))
        WHERE ...
    END;
    RETURN 1;
END;
```

Argument types other than hll field elided

SINGLE WRITE

```
select upsert_test_count(
        'some-identifier-string'::text,
        '2015-05-16'::date,
        '22'::smallint,
        'cabef32d-bc21-4a34-993d-3e7d606df9c6'::text,
        'Catagory1'::text
);
```

TUNING

Optimum index configuration (3/4 dimensions indexed)

The fillfactor tells Postgres to pre-allocate 90% of the index space empty, copy data less.

Standard best practices for workMem, and other memory settings

STILL TOO SLOW

~2,000 events/sec

A transaction per tuple just won't work long-term

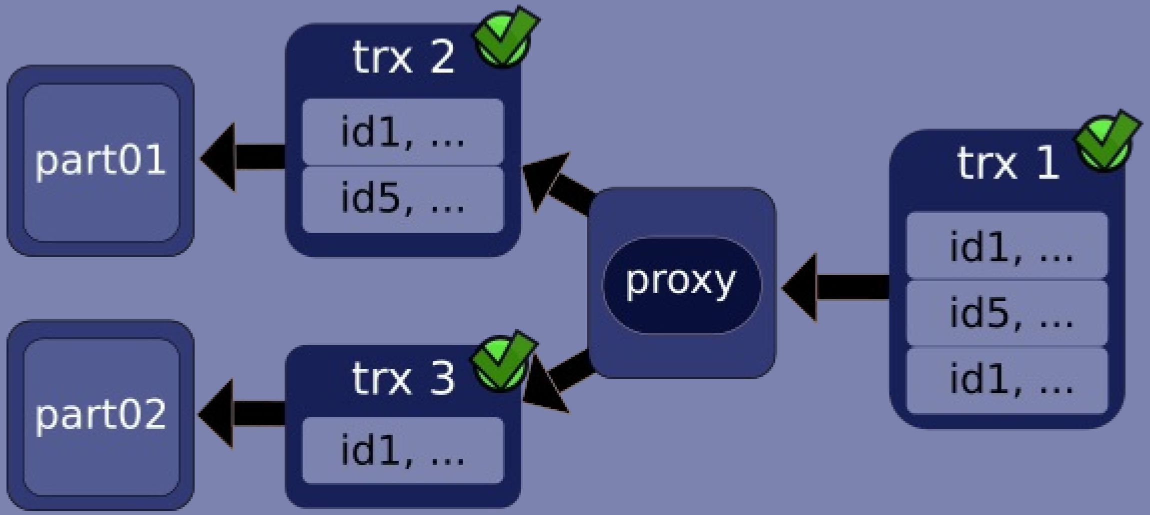
BATCHING

```
CREATE OR REPLACE FUNCTION upsert_test_count(
    in_ids text[], in_dates date[], in_hours smallint[],
        in_event_ids text[], in_cats text[]
) RETURNS TABLE (update int)
BEGIN

RETURN QUERY SELECT upsert_push_hll(
        c.in_ids, c.in_date, c.in_hour, c.in_event_id, c.in_cats
) FROM unnest(
        in_ids, in_dates, in_hours, in_event_ids, in_cats
) as c (in_id, in_date, in_hour, in_event_id, in_cats);
END;
$$;
```

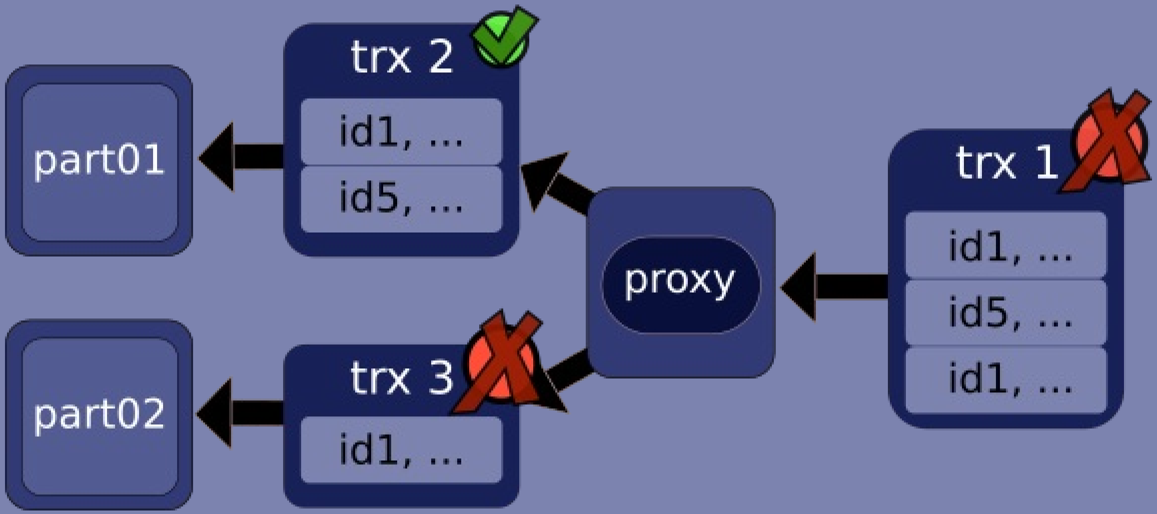
BATCH WRITE QUERY

```
select upsert_test_count(
        '{aaaaaaaaaaaaaaaaaaaaaa, ..., ...}'::text[],
        '{2015-05-15,2015-05-16,2015-05-16}'::date[],
        '{20,21,23}'::smallint[],
        '{cabef32d-bc21-4a34-993d-3e7d606df9b1, ..., ...}'::text[],
        '{Category1,Catagory2,Category1}'::text[]
);
```

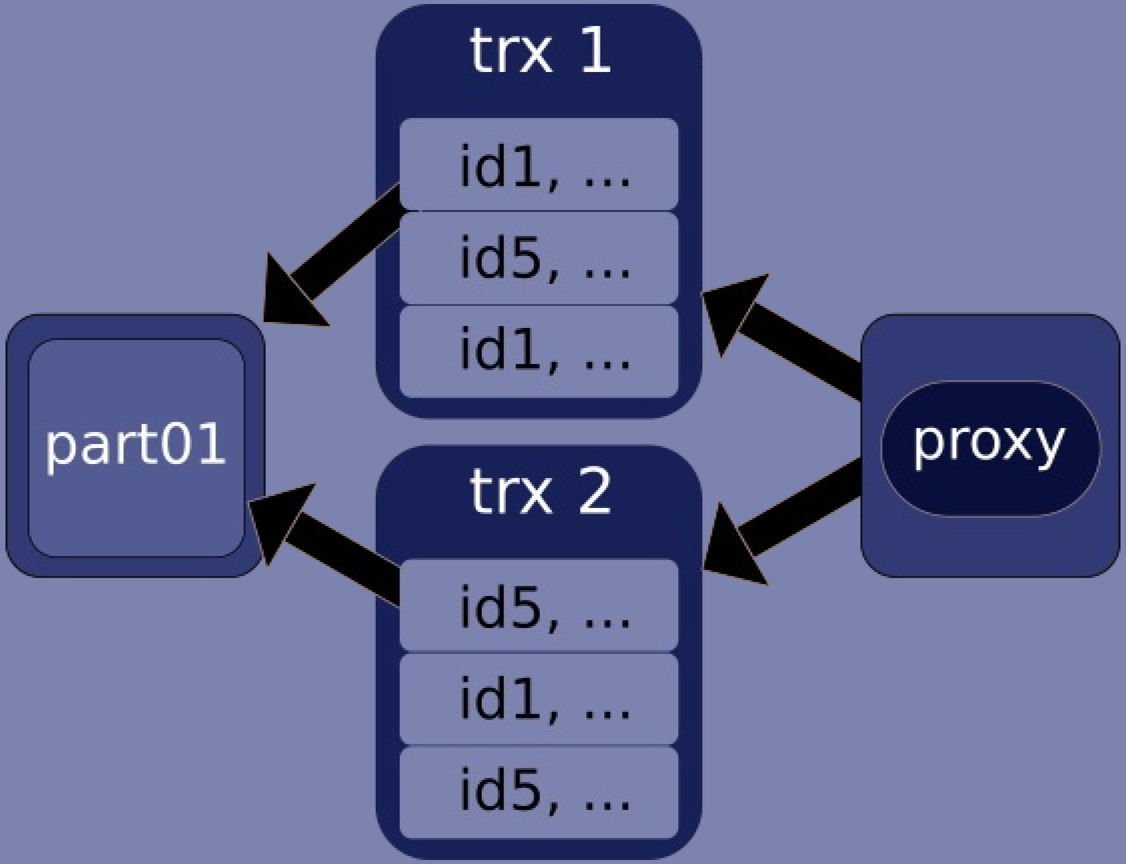
ANATOMY OF A PLPROXY TRANSACTION



WHEN THINGS GO WRONG



DEADLOCK DETECTED!



DEADLOCK SOLUTIONS

- Sort tuples before submitting them
- Single writer pattern

Our functions make sorting difficult, so single writer

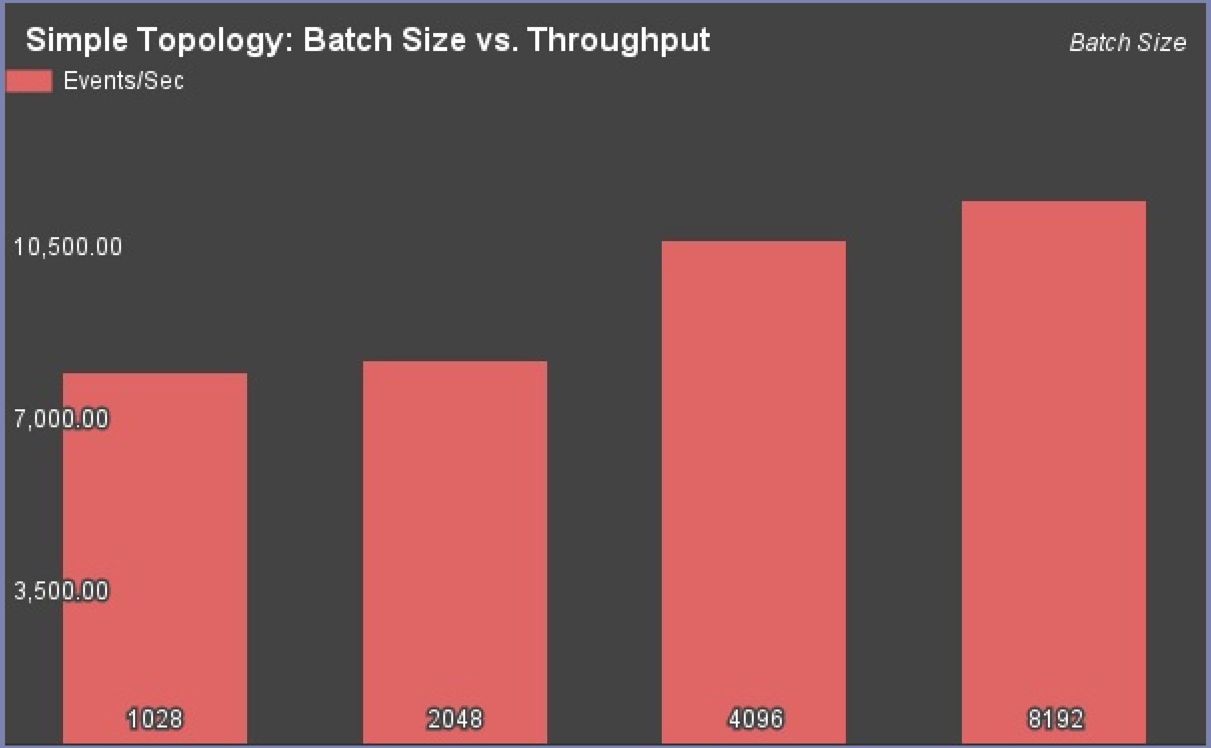
SIMPLE TOPOLOGY

Peaks out with tuning, indexes, and batching at **11k events/sec**

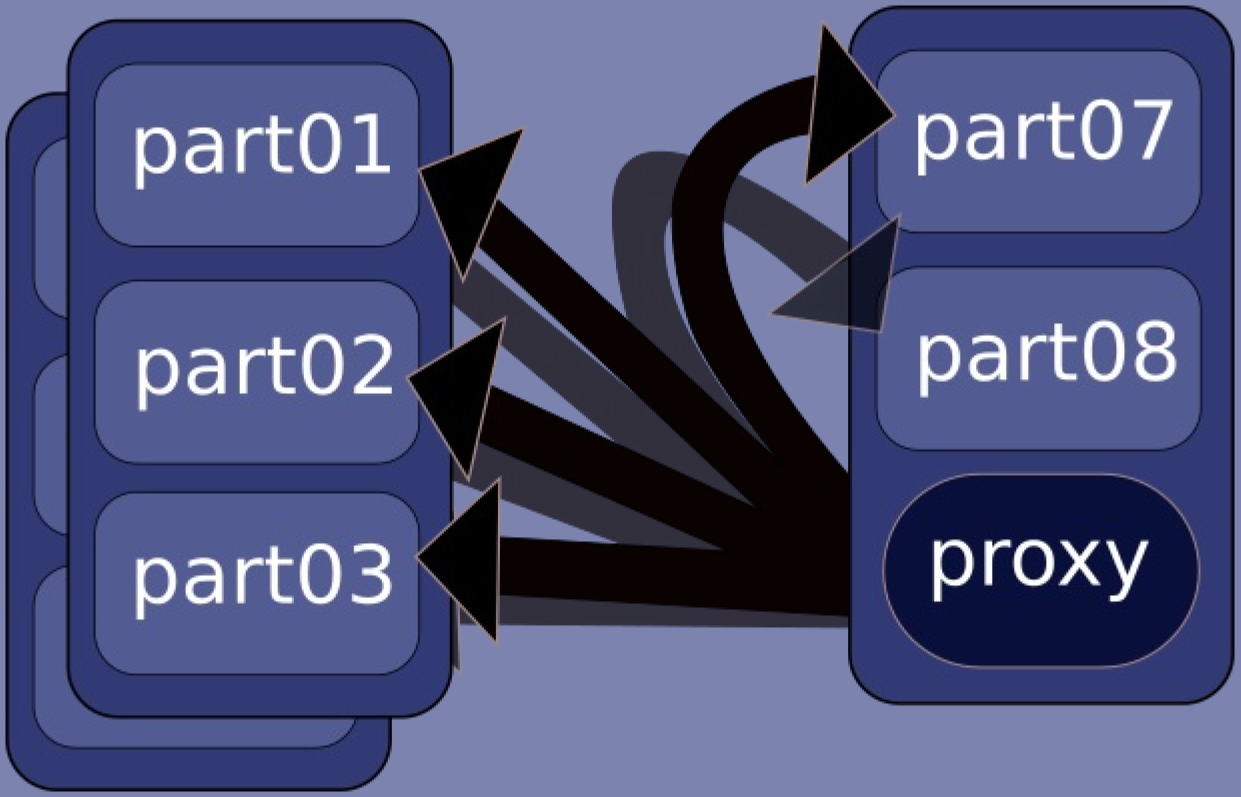
Next step is to increase parallelism

BENCHMARK RESULTS

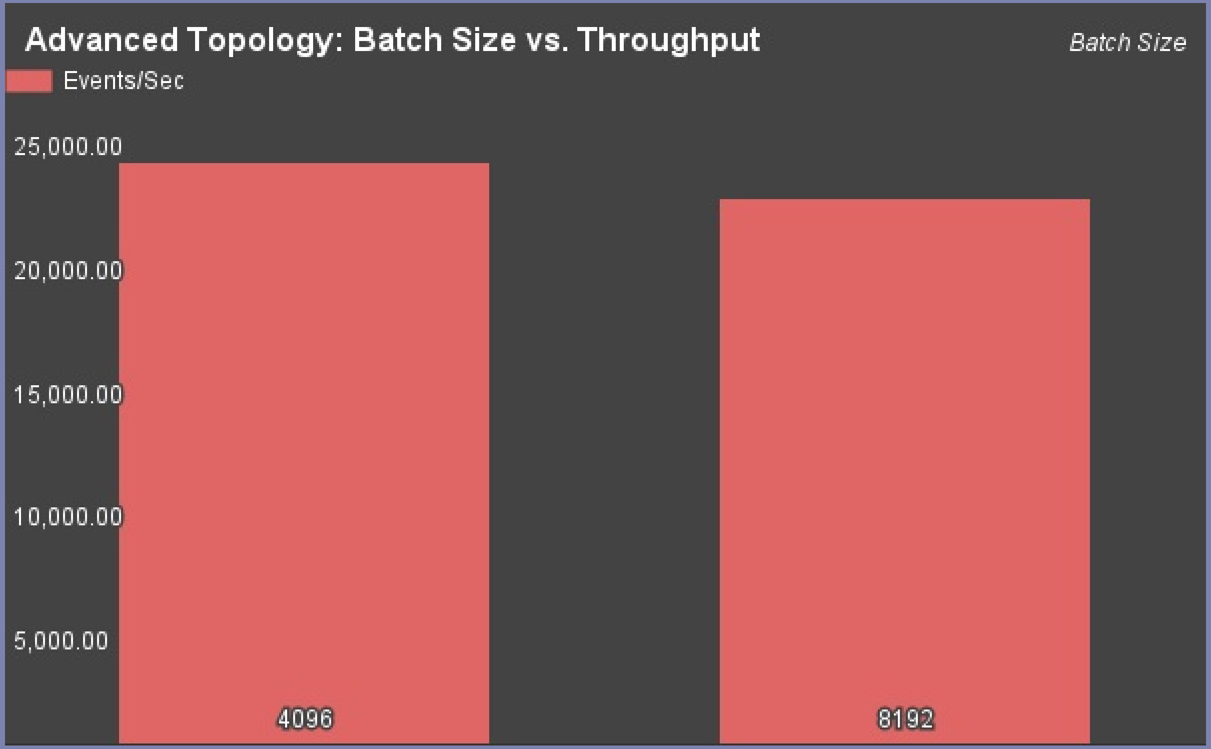
SIMPLE TOPOLOGY THROUGHPUT (200K)



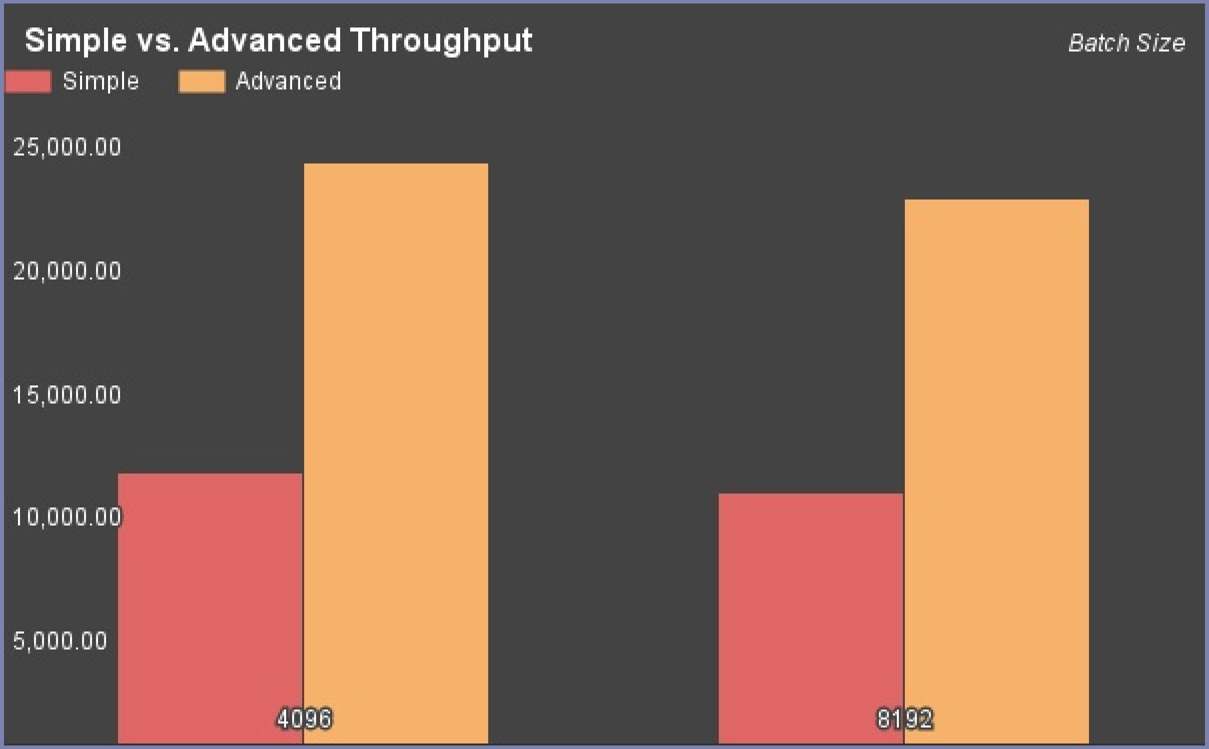
ADVANCED TOPOLOGY



ADVANCED TOPOLOGY THROUGHPUT (2MM)



DIRECT COMPARISON (2MM)



BENCHMARKS ON A LOADED CLUSTER

TYPES OF LOAD

1. Data load: number of rows, size on disk
2. Concurrent requests

SETTING UP A LOADED SYSTEM

Cluster State	Cluster Size (MB)	Index Size(MB)	Number of Rows	Number of IDs
Before	63,864	21,824	333,757,839	307,520
After	80,096	27,088	412,900,728	357,520

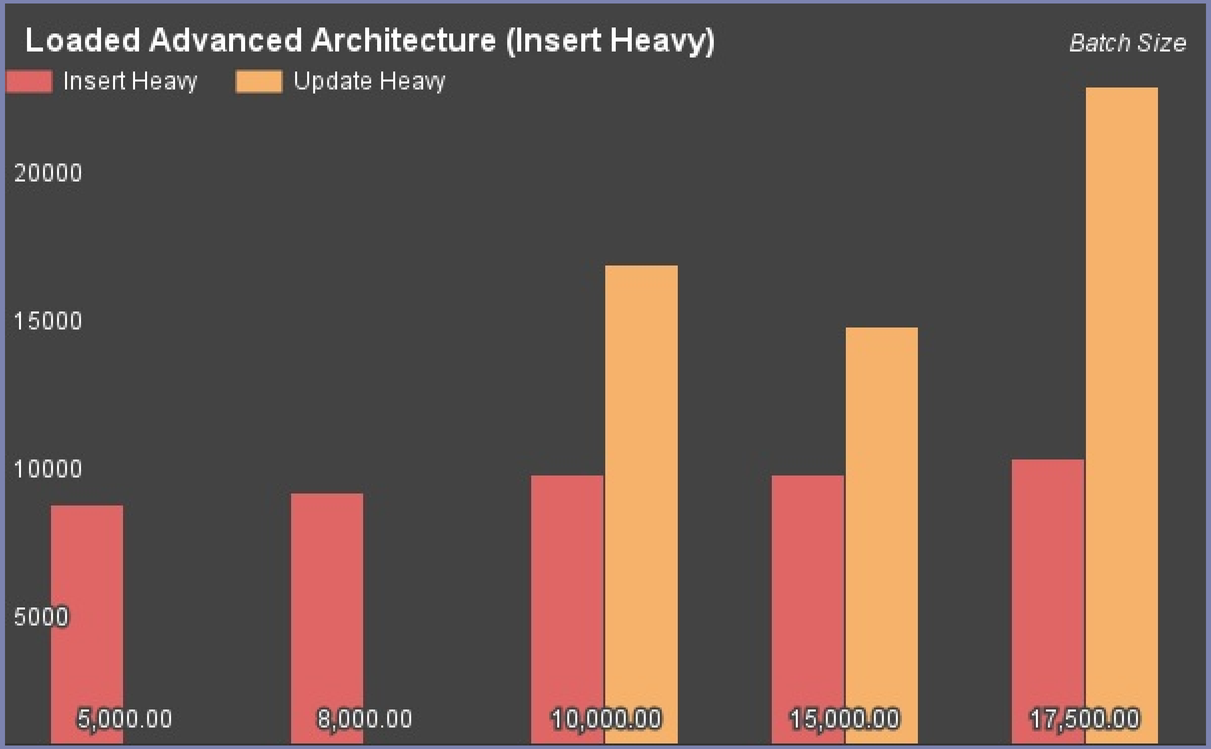
SETTING UP CONCURRENT REQUESTS

Pre-generate insert query batches into .sql files

Run 10 concurrently in a screen session

Not 100% representative of real-world behavior

LOADED RESULTS



READ QUERY (ADHOC)

```
SELECT id, date, hour, hll_cardinality(event_ids)
FROM dynamic_query(
        'SELECT * from test_counts
        WHERE date >= (now() - interval ''7 days'')
           AND id = ''M2E0MDdlNzYtY2Y4NC00Nz'''
) AS (
    id char(22),date date,hour smallint,event_ids hll, cat text)
ORDER BY
    date desc,
    hour desc
LIMIT 10;
```

READ QUERY RESULTS

```
id                  |    date    | hour | hll_cardinality
------------------------+------------+------+-----------------
 M2E0MDdlNzYtY2Y4NC00Nz | 2015-06-10 |   18 |               6
 M2E0MDdlNzYtY2Y4NC00Nz | 2015-06-10 |   13 |               6
 M2E0MDdlNzYtY2Y4NC00Nz | 2015-06-10 |   13 |               6
 M2E0MDdlNzYtY2Y4NC00Nz | 2015-06-10 |    6 |               6
 M2E0MDdlNzYtY2Y4NC00Nz | 2015-06-10 |   21 |               5
```

WRAP UP: POSTGRES FOR DISTRIBUTED OLAP

- Postgres can scale horizontally.
- Write throughput ~= Hbase system.
- New features are a few lines of SQL
- We **retain** queryability and DDLs
- Operational concerns only get worse :(

REMAINING WORK

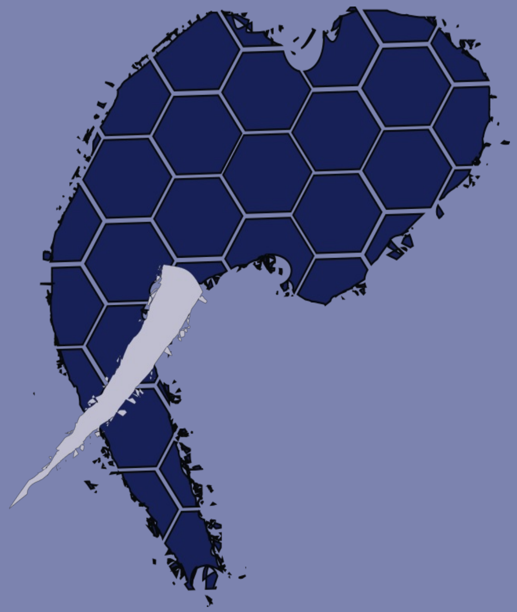
FUTURE FEATURES

- Cross table joins
- Automated failovers(shards)
- Automated, efficient backups
- Tools to help migrate data, add partitions
- Integrating PGBouncer

WORK IS ONGOING

Ansible automation for setting up a test cluster

github.com/gmcquillan/pg_plural



THANK YOU

REFERENCES

- PLProxy Syntax Reference
- PLProxy FAQ
- Martin Kleppmann on Transactions [VIDEO]
- depesz.com
- Urbanski Presentation at pgconf.ru [PDF]
- Deadlocks in Postgresql
- HyperLogLog: the analysis of near-optimal cardinality estimation algorithm - Flajolet [PDF]