PostgreSQL at 20TB and Beyond

Analytics at a Massive Scale

Chris Travers

Adjust GmbH

April 19, 2018

About Adjust

Adjust is a market leader in mobile advertisement attribution. We basically act as a referee that attributes a given installation to a particular advertisement. We focus on fairness, fraud-prevention and other ways of ensuring that advertisers pay for the services they receiving fairly and with accountability.

- Our Environment
    - About Adjust
    - Traffic Statistics
    - Analytics Environment
- Challenges and Solutions
    - Staffing
    - Throughput
    - Autovacuum
    - Data Modeling
    - Backups and Operations
- Conclusions

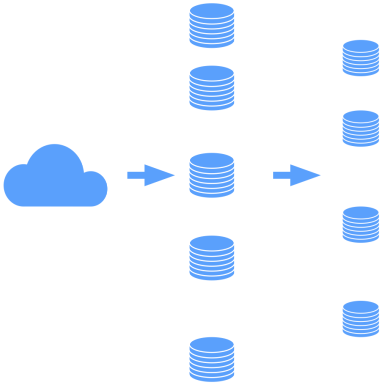
Basic Facts About Us

- We are a PostgreSQL/Kafka/Go/C shop (mostly)
- Around 200 employees worldwide
- Link advertisements to installs or purchases
- Delivering near-real-time analytics to software vendors
- Our Data is “Big Data”

Just How Big?

- Over 100k requests per second baseline
- Over 2 trillion data points tracked in 2017
- Over 500 TB of data to analyze
- Very high data velocity

General Architecture



- Requests come from Internet
- Written to backends
- Materialized to analytics shards
- Shown in dashboard

Common Elements of Infrastructure

- Bare metal
- Stripped down Gentoo
- Lots of A/B performance testing
- Approx 50% more throughput than stock Linux systems
- Standard PostgreSQL + extensions

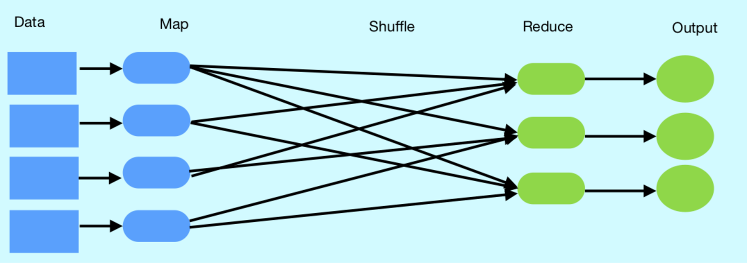
Backend Servers

- Original point of entry
- Data distributed by load balancer
- No data-dependent routing.
- Data distributed more or less randomly
- Around 20TB per backend server gets stored.
- More than 20 backend servers

Materializer

- Aggregates new events
- Copies the aggregations to the shards
- Runs every few minutes
- New data only

Materializer and MapReduce



Our materializer aggregates data from many servers and transfers it to many servers. It functions sort of like a mapreduce with the added complication that it is a many server to many server transformation.

Analytics Shards

- Each about 2TB each
- 16 shards currently, may grow
- Our own custom analytics software for managing and querying
- Custom sharding/locating software
- Paired for redundancy

Staffing: Challenges

- No Junior Database Folks
- Demanding environment
- Very little room for error
- Need people who are deeply grounded in both theory and practice

Staffing: Solutions

- Look for people with enough relevant experience they can learn
- Be picky with what we are willing to teach new folks
- Look for self-learners with enough knowledge to participate
- Expect people to grow into the role
- We also use code challenges

Throughput challenges

- Most data is new
- It is a lot of data
- Lots of btrees with lots of random inserts
- Ideally, every wrote inserted once and updated once
- Long-term retention

Throughput Solutions

Backend Servers

- Each point of entry server has its own database
- Transactional processing is separate.
- Careful attention to alignment issues
- We write our own data types in C to help
- Tables partitioned by event time

Throughput Solutions

Analytics Shards

- Pre-aggregated data for client-facing metrics
- Sharded at roughly 2TB per shard
- 16 shards currently
- Custom sharding framework optimized to reduce network usage
- Goal is to have dashboards load fast.
- We know where data is on these shards.

Throughput Solutions

Materializer

- Two phases
- First phase runs on original entry servers
- Aggregates and copies data to analytics shards
- Second phase runs on analytics shards
- further aggregates and copies.

Materializer: A Special Problem

- Works great when you only have one data center
- Foreign data wrapper bulk writes are very slow across data centers
- This is a known issue with the Postgres FDW
- This is a blocking issue.

Materializer: Solution

- C extension using COPY
- Acts as libpq client
- Wrote a global transaction manager
- Throughput restored.

Introducing Autovacuum

- Queries have to provide consistent snapshots
- All updates in PostgreSQL are copy-on-write
- In our case, we write once and then update once.
- Have to clean up old data at some point
- By default, 50 rows plus 20% of table being “dead” triggers autovacuum

Autovacuum problems

- For small tables, great but we have tables with 200M rows
- 20% of 200M rows is 40 million dead tuples....
- Autovacuum does nothing and then undertakes a heavy task....
- performance suffers and tables bloat.

Autovacuum Solutions

- Change to 150k rows plus 0%
- Tuning requires a lot of hand-holding
- Roll out change to servers gradually to avoid overloading system.

Why it Matters

- Under heavy load, painful to change
- Want to avoid rewriting tables
- Want to minimize disk usage
- Want to maximize alignment to pages
- Lots of little details really matter

Custom 1-byte Enums

- Country
- Language
- OS Name
- Device Type

All on PGXN!

IStore

Like HStore but for Integers

- Like HStore but for integers
- Supports addition, etc, between values of same key
- Useful for time series and other modelling problems
- Supports GIN indexing among others

The Elephant in the Room

How do we aggregate that much data?

- Basically incremental Map Reduce
- Map and first phase aggregation on backends
- Reduce and second phase aggregation on shards
- Further reduction and aggregation possible on demand

Operations Tools

- Sqitch
- Rex
- Our own custom tools

Backups

- Home grown system
- Logical per-table backup
- Runs as a Rex task
- We can also do physical backups (but...)

Ongoing Distributed Challenges

- Major Upgrades
- Storage Space
- Multi-datacenter challenges
- Making it all fast

Overview

This environment is all about careful attention to detail and being willing to write C code when needed. Space savings, better alignment, and other seemingly small gains add up over tens of billions of rows.

Major Points of Interest

- We are using PostgreSQL as a big data platform.
- We expect this architecture to scale very far.
- Provides near-real-time analytics on user actions.

PostgreSQL makes all this Possible

In buiding our 400TB analytics environment we have yet to outgrow PostgreSQL. In fact, this is one of the few pieces of our infrastructure we are perfectly confident in scaling.