Faster Averages in the Real World

Author: Paul Langton

Claps: 453

Date: May 4, 2021

Every month, Klaviyo releases Benchmarks for the previous month by aggregating critical business metrics from databases across the Klaviyo information sphere. In order to ensure Benchmarks are as accurate and up to date as possible, we take 5 days at the beginning of every month to query and audit the data before releasing Benchmarks to hundreds of thousands of customers. This March we noticed query time for our largest customers taking up most of our internal 5 day SLO, leaving us insufficient time to audit and release Benchmarks.

In this piece weâ€™ll discuss how we implemented a subsampling strategy to achieve a 34x speedup in processing time for our largest customers, from more than 4 days down to 2.85 hours.

Metrics at Profile granularity

The Benchmarks workload will look familiar at first glance to those acquainted with analytics workloads common on Business Intelligence teams. Data from disparate production sources is pulled into an ETL pipeline, transformed, and persisted to an intermediate, more easily-queried database for presentation. This process took about 4 days with the vast majority taken up by one particularly slow query, on which we will focus for the rest of the piece.

To keep a healthy air of mystery about us, weâ€™ll be speaking about the query in question in terms of beehives. Bees are a fun superorganism, not to mention they produce delicious bee goo. At Klaviyo, our customers have excellent taste and therefore care a lot about their engagement with the keepers of said superorganism. Benchmarks naturally provides the **average number of beehives per profile (ABP)** as a key metric to indicate how in touch a customer is with the beekeeping community.

Vocabulary: each Klaviyo **customer** has a number of **profiles** which represent individuals associated with their brand (Figure 1).

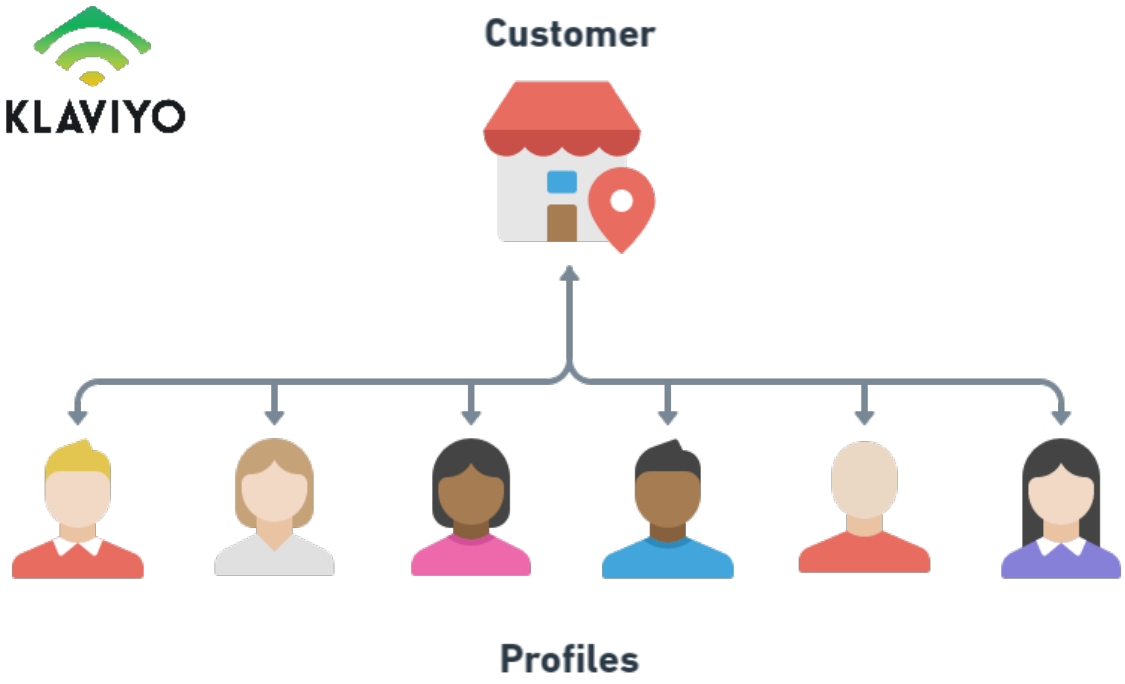

Figure 1: The basic data structure of a Klaviyo customer

At Klaviyo, we pre-aggregate metrics like beehive count at both the customer and profile level per hour, day, and month. Since Benchmarks presents aggregate metrics about a customerâ€™s monthly performance, most Benchmarks queries are able to take advantage of customer-level aggregates. The natural approach to compute ABP is to sum each month of beehive counts for a customer and divide by their number of profiles.

Unfortunately this approach does not work because the beehive count metric suffers from a serious outlier problem. Specific profiles, including test profiles, beekeeper fanatics, and beekeeping wholesalers skew the customer-level beehive count significantly because these profiles have an enormous amount of beehives that isnâ€™t representative of their regular profiles, resulting in an overly optimistic picture of how the profiles are engaging with beekeeping. This source of noise can be significant for some customers, skewing the ratio by up to 290%.

To remove this source of bias we need access to beehive counts **per profile**. To calculate this, we sum each profileâ€™s monthly beehive counts, then account for outlier profiles by inferring a reasonable **outlier threshold** and drop profiles with beehive counts above the threshold. The final ABP value for a customer is calculated as follows.

```
ABP = (sum(beehive_counts) â€” sum(outliers)) / (len(beehive_counts) â€” l
```

This March, we noticed the total ABP query runtime for some of our larger, older customers getting dangerously cozy with our 5-day SLO.

Plan of Attack

We knew the query needed to be faster, but didnâ€™t yet have any concrete requirements beyond â€œuphold our 5-day SLOâ€�. To gather requirements, we started with the facts.

- The time complexity of this query is horrendous. For `n = number of profiles, m = mean profile age`, ABP runs in roughly `O(nm)` time.
- For our largest customers ABP can take up to 4 days real time to complete.

The second point in conjunction with some background knowledge about our deploy systems gave us a good starting point for developing a requirement. Deployments at Klaviyo are by and large for changes to the Klaviyo Django monolith, which is run by almost all compute instances running application-level logic. It is important that every instance be running the latest (or at least a recent) version, so most deployments include all machines running the monolith, which includes our Benchmarks ETL workers. Deployments also interrupt running processes, which puts an implicit (but not previously encountered) upper bound on the amount of time any single query can take to complete.

This leads us to our first major requirement: we want our ETL workers to be part of monolith deploys, so our upper bound on query time should be related to the time between deploys. Deployment frequency varies, but on R&D off hours there are usually a few 6-hour windows which go without deploys. We are fine with these queries struggling during high-frequency deploy business hours, but ABP should be able to finish without having to wait for the weekend.

Requirement 1: For any customer, ABP must complete in less than 6 hours.

Satisfied with our fleshed out time requirement, we started considering possible approaches. Our first approach concerned speeding up ABP at the database level by adding a new metric aggregated per customer for outlier-adjusted beehive counts. With this data we could avoid querying every single customer profile to compute outliers, theoretically bringing our query time to `O(1)` in exchange for pre-aggregating a new metric for every customer. Thatâ€˜s great, but probably too heavy-handed an approach for our once-monthly Benchmarks query with a 5-day SLO.

Next, we turned to reducing the amount of data necessary for ABP. Armed with the statistical intuition that increasing sample size gives diminishing returns on the accuracy of a population average, our second approach was to implement an approximate solution at the query layer which used a subsample of profiles to estimate ABP. Any approximate approach begs for a requirement which specifies the range of acceptable inaccuracies, so we sought help from the Benchmarks UI to understand how customers use ABP. Since the purpose of ABP is to help customers make business decisions, an approximate ABP should not differ from the real ABP by enough to impact our customersâ€™ business decisions. ABP is presented in the Benchmarks UI to 2 decimal places of precision, which led us to our second requirement.Â[1]

Requirement 2: For any customer, an approximate ABP should not differ from the actual ABP by a delta more than 0.1.

Evaluating the Approach

With requirements defined and an approach in mind, we designed a numerical simulation to see if our approach would hold water on real data. First, we built a new ABP that could leverage profile sampling. Itâ€™s very simple but we include it below for reference.

```
def sampled_abp(customer, sample_size):
    sample = choose(sample_size , customer.profiles)
    beehive_counts = [alltime_beehive_count(profile) for profile in sample
    outlier_counts = outliers(beehive_counts)
    return (sum(beehive_counts) â€” sum(outliers)) / len(beehive_counts) â
```

We held sample size constant at 60000 and did not test customers with fewer than 60000 profiles, for whom no sampling is necessary because computation speed is acceptably fast. We chose some test customers stratified across a few potentially concerning groups:

- **Big**: â€œWhaleâ€�-type customers. Theyâ€™re valuable, usually older customers with lots of profiles. Concern: runtime performance
- **High Outlier Thresholds**: customers with high ABP outlier thresholds relative to their average. Concern: ensure the outlier detection model is robust to subsampling and does not misrepresent data when outliers are extreme or in a long, thin tail
- **Low Outlier Thresholds**: customers with low ABP outlier thresholds relative to their average. Concern: ensure the outlier detection model is robust to subsampling when there is a short, fat tail of outliers
- **Randoms**: A random sample of customers for the control group

We ran 1000 iterations of ABP subsampled at a constant 60000 profiles for each test customer and observed the following results (Figure 2).Â¹