Taking the first SIP: An overview of Klaviyoâ€™s Segmentation Improvement Project, or how we learned to stop worrying and love OLAP databases

Author: Josh Bradt

Claps: 172

Date: Jul 10

This post was co-authored by

Josh Bradt
and
Nicholas Hoffmann
.

One of the most interesting parts of working on the engineering team at Klaviyo is watching how huge systems respond to increasing scale as the company grows. We try to build systems that will scale up to meet increasing demand and incoming data rates for at least the next 3â€“5 years. As the end of a systemâ€™s useful life approaches, though, itâ€™s fascinating to watch what we thought was our best work gradually start to fall behind.

Over the last year, we took on a big project to replace Klaviyoâ€™s segmentation engine, which we called the Segmentation Improvement Project (SIP). Segmentation lets our customers create dynamically updating audiences from the profiles in their Klaviyo accounts based on extremely flexible user-defined rules. This system has to respond to billions of changes to data points across the Klaviyo platform each day to keep millions of segments up to date in real time. The existing system was starting to reach the limits of what we could do to scale it further, so over the last year, we designed and implemented a new, more scalable replacement segmentation engine. In this blog post, weâ€™ll talk about what segmentation is, how the old system worked, and the design of our new and improved segmentation engine.

What is a segment?

Though the Klaviyo app is flexible and is used for many different use cases, our typical user is a direct-to-customer ecommerce business. These businesses integrate Klaviyo with their online storefront provider to track things like which products their customers are viewing, customersâ€™ purchase histories, and properties about their customers. Businesses can use this data to send well-targeted, personalized email and SMS communications to their customers with the goal of sending people marketing messages theyâ€™re actually interested in seeing.

At the core of this system is the profile, which represents a person who can be communicated with. Profiles have both identifying properties (like an email address or phone number) and other attributes (like a mailing address, or whatever custom properties a business needs).

Segments are dynamic groupings of profiles based on their attributes. Customers define a set of rules that they want to use to include profiles in a segment. The segmentation engine then compares all the profiles in the customerâ€™s account to the segment definition and records which profiles belong in the segment. Segments are also automatically updated as new data enters Klaviyo. For example, if the definition says that the profile must have looked at jeans on the web site in the last ten days, and someone just did that, they will be immediately included in the segment.

Segment definitions are composed of a set of criteria that determine the segmentâ€™s membership. There are many different types of criteria, and they draw data from across the Klaviyo application, making segments a very flexible tool. Some example criteria include:

- People who have placed an order at least once in the last 30 days
- People who have clicked a link in a specific email campaign
- People who live within 10 miles of the ZIP code 02110 in Boston, MA
- People who have consented to receive SMS messages
- People whose expected date of next order, based on their activity history, is in the next two weeks

These criteria can be combined with AND and OR to make arbitrarily complex logic.

The original segmentation engine

At a basic level, the approach of our original segmentation engine was to pull data from the various underlying data stores powering the Klaviyo app, check the data against the segment definition, and record which profiles qualified for the segment.

Segmentation operates in three different modes:

1. When a new segment is created, the system needs to calculate the initial set of memberships from all profiles in an account. This processing mode is also used if the definition of an existing segment is updated. We call this **manual segmentation** since itâ€™s triggered on-demand by the Klaviyo customer.
2. Memberships in existing segments are kept up to date by a real-time processing mode as events and other data about each profile are streamed into Klaviyo. Customers rely on their segments being up to date and ready to use for things like email campaign sends and segment-triggered flows, so itâ€™s critical that **real-time segmentation** keeps to a tight SLO.
3. Finally, a **nightly relative time check** is an extension of the real-time processing mode that accounts for membership changes that should happen due to the passage of time. As an example, if there is a criterion that checks if zero orders were placed in the last 30 days, there is no â€œnon-eventâ€� that happens on day 31 that tells the system to update any relevant memberships. The relative time check will evaluate that criterion periodically and make sure everything is kept up to date.

The original segmentation engine worked by directly querying the many OLTP databases that store the data needed to calculate who belonged in a segment and then combining this data in Python code. The latest incarnation of this was a modular system with standard interfaces for fetching and caching data from the source of truth data stores, formatting it for use with the segmentation engine, and evaluating segment criteria against it. This model was flexible and easy to extend since the query and evaluation logic needed for each type of segment criterion was nicely encapsulated in one place.

The catch was that each backing datastore needed to be able to tolerate the segmentation engineâ€™s scale. While some data was read from previously computed aggregates stored in columnar stores like Cassandra, in many cases we were effectively performing analytical queries against OLTP databases â€” typically Amazon Aurora MySQL clusters â€” and thatâ€™s to say nothing of the sheer volume of data we were sending over the wire. As a result, the segmentation system had knocked over its fair share of data stores over the years, earning it the internal nickname â€œthe ray gun of death.â€� It was particularly effective at degrading the Cassandra nodes storing pre-aggregated event data, which was unfortunate since event-based criteria are the second most commonly used criterion type, being evaluated roughly 100 million times per minute on a typical day.

> The GIF that appears when you type â€œactual footage of real-time segmentationâ€� in Klaviyoâ€™s Slack (Source: GIPHY)

These Cassandra clusters with pre-aggregated counters for events were one key optimization in the old system. Others included careful batching of work to allow Cassandra to read continuous ranges of data from disk, tuning queries to rely on indexes in MySQL databases, some ingenious task spawning tricks that relied on knowing implementation details of our profile schema to accelerate how fast we could create new segments, and an elaborate Redis-based system for deduplicating real-time segment update triggers within sliding time windows.

By the start of 2022, though, it had become clear that we were running out of tricks. The segmentation system was successfully handling more data than ever, with 5.19 billion real-time segment updates being triggered on Black Friday alone in 2021, but we were having to run an unsustainable amount of hardware to support the workload. With hundreds of nodes in total, the Cassandra clusters that held our event counters in particular were getting prohibitively expensive with each round of scaling out. As Klaviyo added larger and larger customers, we also wanted to make sure that they got the same speedy performance that everyone had come to expect from the segmentation engine. That realization led us to consider a deeper overhaul of the segmentation engine and think about what an alternative might look like.

Requirements for the new system

The team had worked with the segmentation engine for several years at this point, so we had a good idea of where the gaps and shortcomings were. In addition to the constant desire to be faster, we had built up a backlog of feature requests and improvements that were difficult or impossible to build in the old system. We also knew that we wanted the new system to cost significantly less to operate than the old one. Ideally it would be easier to operate as well since our Cassandra clusters were often at the heart of some of our longest and most challenging incidents.

Using all of that as a starting point, we worked with our product managers to come up with a set of primary requirements for the project. After a bit of back and forth, we chose an arbitrary performance target that would be a great experience for customers, and which we suspected was within the realm of technical possibility with good engineering. Choosing a target brought focus to our discussions and allowed the project to move forward. A lot of our early design work then went into validating this target as something that was feasible before we went on to build the rest of the system.

By the end of Q1 of 2022, we had a set of primary requirements that set a clear destination for our desired end state. We also had several high-level design options for engineering to pursue further. The requirements we ended with were:

1. Create a segment in less than 5 minutes at p99 for an account with 60M profiles
2. Be able to keep up to 18M segments up to date in real time
3. Support all existing functionality from the current segmentation engine
4. Support known future feature development
5. Operating costs no more than 50% of the cost of the existing system

Overview of the new system

After completing our research and testing prototypes, we concluded that the only way to hit our performance target was to turn the segmentation model on its head: instead of bringing the data to the queries, weâ€™d bring the queries to the data. The right tool for this job was an OLAP database.

Weâ€™d actually tossed around the idea of running segmentation on a giant OLAP database for years, but weâ€™d dismissed the idea before since it would require duplicating all of the data we base segments on and keeping that duplicate copy in sync with the source of truth, which is a substantial task. However, after a few years of rapid growth, the engineering team had already invested in defining robust domain boundaries around each Klaviyo product area with all updates being funneled through service classes. This earlier work â€” along with the increased capacity of the team â€” now made this previously formidable project something we could actually consider.

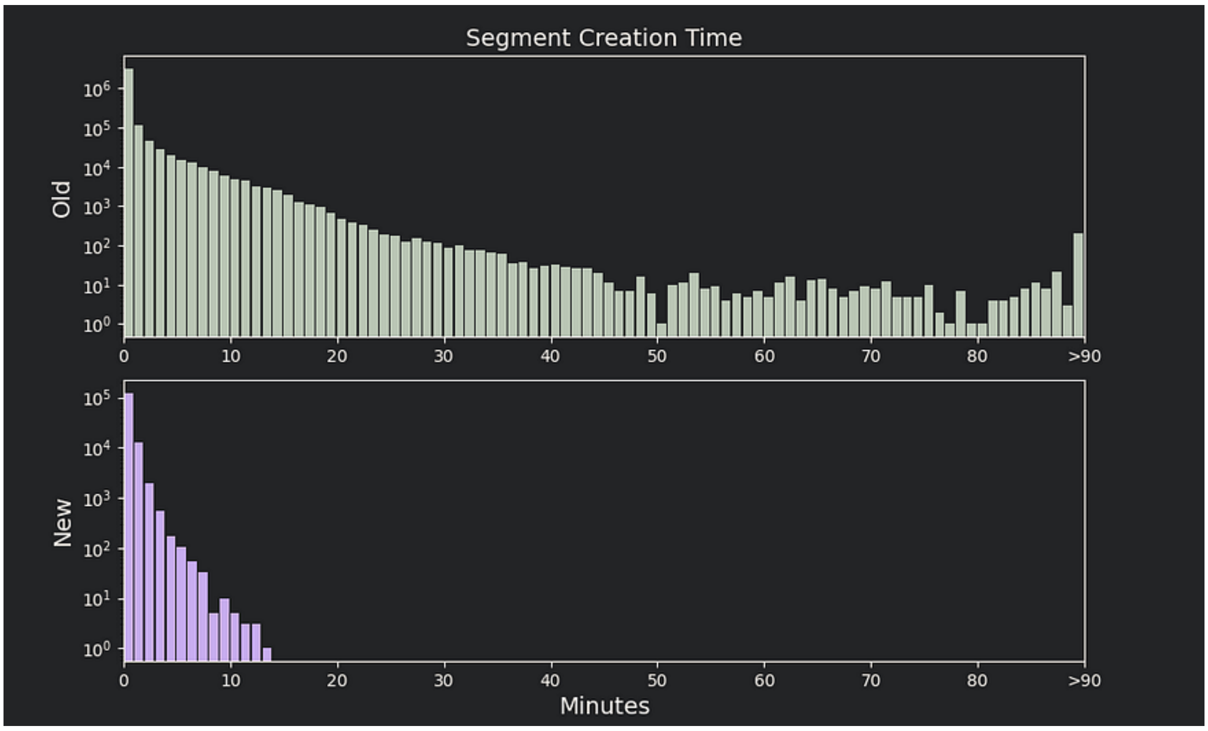
We also had the benefit of running an OLAP database in another part of Klaviyo. Another team that we work closely with was using ClickHouse to power the latest iteration of our account-level event and reporting functionality. Choosing technologies that other teams at Klaviyo had already used successfully was a big part of balancing risk and innovation for this project. We wanted to use the best tools available and stay aligned with the broader technical direction of the company, but we didnâ€™t want to get too far out ahead of everyone else. We did spend a few more â€œinnovation tokensâ€� by running the new segmentation engine on Kubernetes and using Pythonâ€™s asyncio framework, both of which were newer to the team, but here again we were able to follow a path blazed by others. We compensated for this relative novelty in other areas by choosing â€œboringâ€� technologies like Python and Django that we were experts in to help manage risk.

The new segmentation engine has a 192-node ClickHouse cluster at its core. We considered a few alternatives for the database, but we landed on ClickHouse because, as mentioned earlier, we already had in-house experience using it for event analytics, and it also supported all of the features we wanted for our long-term roadmap.

The ClickHouse cluster is kept up to date by custom change capture pipelines that funnel in 1.5 billion profile trait updates per day and nearly 2 billion events per day from the rest of the Klaviyo app. Every time something about a profile changes â€” including things like their contact details, custom properties about them, their subscription status, or predicted attributes about them from our data science team â€” a change notification message is sent into a topic in our internal messaging service. Change notification handlers respond to those messages by reading the current profile trait values from the source-of-truth databases we used to query in the old segmentation system, serializing the result into a format close to what weâ€™ll insert into ClickHouse, and sending the result into a second messaging service topic. Finally, these trait change messages are grouped by ClickHouse shard and inserted into ClickHouse in large batches.

With all of the data colocated on our one giant ClickHouse cluster, the next task was to reimplement all of our segmentation logic in SQL. This was quite complicated since a core benefit of the segmentation engine is its flexibility: we allow our customers to combine as many criteria as they need to get the job done. Additionally, we needed to be able to process many segments at once in large batches since ClickHouse works best with a low rate of more complex queries. These requirements meant that we couldnâ€™t dynamically build SQL queries in code based on the customerâ€™s segment definition (to say nothing of the security risks that would entail), so we created a way of encoding segments as sets of filtering rules that we could store in the ClickHouse cluster and join to our data at runtime. The query that supports this weighs in at over 1300 lines of heavily optimized SQL. The result of this query is the list of segment members, which is written to a table in ClickHouse to be read out asynchronously by another process that syncs those back to the main Klaviyo app.

The full system took us a little over a year to design, implement, tune, and release, and the results were worth it. We met all of our project requirements. Most end users will have noticed no difference (which is good!), but for users working with large segments, the speed improvements are striking:



Histogram of segment creation time, exponential scale. Even segments with 20+ criteria for accounts with tens of millions of profiles can be created in under fifteen minutes during peak system load, which was not the case with the old system.

Conclusion

The Segmentation Improvement Project is the largest project that the Segmentation Team has undertaken to date and is among the largest projects that any team at Klaviyo has ever done. The project was an opportunity for the whole team to do some of the best and most engaging engineering work of our careers so far. This summary blog post presents the project as a straightforward process of requirements, design, build â€" but in fact, the project was filled with fascinating challenges. We learned important lessons around technical design, leveraging the best parts of the technologies we chose, and managing such a large project. Weâ€™re planning on discussing all of those topics in future posts, and weâ€™ll link them here as they are published.