چالش ها:

در این تسکی که تعریف شده بود یک سری چالش ها وجود داشت که در این داک به بررسی دقیق این چالش ها ، روش های حل آنها و روشی که ما برای آن انتخاب و پیاده سازی کردیم می پردازیم.

چالش اول:

چالش اول در رابطه با گزارش تعداد امتیاز ها و میانگین آنها بود که با زیاد شدن تعداد امتیاز ها برای یک مطلب ، محاسبه میانگین و تعداد امتیاز ها با کوئری دیتابیسی به ازای هر رکوئست ورودی برای دریافت مطالب هزینه زیادی داشت و پرفورمنس را بد می کرد. حالا در ابتدا به روش هایی که برای حل این مشکل وجود دارد می پردازیم.

راه‌حل‌ها:

۱. ایندکس‌گذاری پایگاه داده:

ایندکس‌گذاری روی فیلدهای مدل `Rating`، مانند `content_id`، می‌تواند سرعت پرس‌وجوها برای محاسبه میانگین و تعداد امتیازات را افزایش دهد. این روش برای مجموعه داده‌های کوچک مفید است، اما با افزایش تعداد امتیازات، ایندکس‌گذاری به تنهایی ممکن است برای جلوگیری از افت عملکرد کافی نباشد.

۲. استفاده از کش:

با ذخیره‌سازی میانگین محاسبه‌شده و تعداد امتیازات در یک کش با دسترسی سریع مانند Redis، می‌توانیم نیاز به پرس‌وجوی مکرر از پایگاه داده را به‌طور چشمگیری کاهش دهیم. این روش دسترسی سریع به این مقادیر را فراهم می‌کند، اما نیاز به مدیریت دقیق برای تضمین هم‌زمانی کش با پایگاه داده دارد.

۳. پردازش دسته‌ای با Celery:

- با استفاده از Celery، می‌توانیم وظایف پس‌زمینه‌ای را برای محاسبه و بررسی دوره‌ای داده‌های امتیازدهی برنامه‌ریزی کنیم. این روش تضمین می‌کند که هر گونه ناهماهنگی در کش به مرور زمان اصلاح می‌شود و تعادل بین عملکرد لحظه‌ای و نیاز به مدیریت داده‌های بزرگ و ناهنجاری‌های احتمالی را برقرار می‌کند.

۴. ذخیره‌سازی مقادیر محاسبه‌شده در مدل:

- ذخیره‌سازی میانگین امتیازات و تعداد آنها به‌طور مستقیم در مدل `Content`، دسترسی فوری به این مقادیر را فراهم می‌کند، اما به منطق پیچیده‌ای برای به‌روزرسانی این فیلدها در زمان واقعی نیاز دارد، به‌ویژه در عملیات هم‌زمان. این روش می‌تواند سیستم را پیچیده‌تر کرده و مستعد خطا باشد.

روش پیاده‌سازی‌شده

ما ترکیبی از استفاده از کش Redis و پردازش با Celery را به‌عنوان راه‌حل انتخاب کردیم:

۱. استفاده از کش Redis:

میانگین امتیازات و تعداد آن‌ها برای هر مقاله در یک کش سریع مانند Redis ذخیره می‌شوند. هر بار که کاربر این اطلاعات را درخواست می‌کند، داده‌ها از کش خوانده می‌شوند به‌جای اینکه مستقیماً از پایگاه داده محاسبه شوند.

نحوه پیاده‌سازی: هر زمان که امتیازی اضافه، ویرایش یا حذف می‌شود، میانگین و تعداد امتیازات محاسبه و در Redis ذخیره می‌شوند. این امر باعث کاهش چشمگیر تعداد پرس‌وجوهای پایگاه داده و بهبود عملکرد می‌شود.

۲. پردازش دسته‌ای با Celery:

برای تضمین صحت و اعتبار داده‌های ذخیره شده در کش، از Celery برای پردازش‌های دوره‌ای استفاده می‌کنیم. یک وظیفه Celery هر ۱۵ دقیقه اجرا می‌شود تا میانگین و تعداد امتیازات را دوباره محاسبه کرده و هرگونه مشکل را بررسی کند.

نحوه پیاده‌سازی: وظیفه Celery، کش را با داده‌های جدید و معتبر به‌روزرسانی می‌کند. این تضمین می‌کند که حتی در صورت وجود ناهماهنگی‌های نادر، کش به‌طور منظم به‌روزرسانی شده و داده‌ها دقیق باقی می‌مانند.

چالش دوم:

دومین چالش بزرگ، مقابله با vote flooding است، جایی که تعداد زیادی از کاربران، اغلب به صورت هماهنگ از طریق پلتفرم‌هایی مانند تلگرام، به یک مقاله امتیاز می‌دهند. این امتیازات ممکن است بازتاب‌دهنده کیفیت واقعی مطلب نباشند، بلکه با هدف افزایش یا کاهش مصنوعی امتیاز کلی مقاله انجام می‌شوند و برای مقابله با این چالش هم باید فکری کرد.

راه‌حل‌ها:

۱. Rate-Limiting by Time Window:

محدود کردن تعداد امتیازاتی که می‌توانند در یک بازه زمانی مشخص برای یک مقاله ارسال شوند. به عنوان مثال، اگر یک مقاله بیش از تعداد معینی امتیاز در ۱۰ دقیقه دریافت کند، امتیازات اضافی یا نادیده گرفته می‌شوند یا وزن کمتری به آن‌ها داده می‌شود که این از اثرگذاری ناگهانی حجم زیاد امتیازات بر امتیاز کلی مقاله جلوگیری می‌کند.

۲. Exponential Moving Average (EMA):

استفاده از Exponential Moving Average برای محاسبه امتیاز مقاله، به طوری که امتیازات اخیر نسبت به امتیازات قدیمی‌تر تاثیر بیشتری داشته باشند. با این حال، امتیازات در دوره‌های فعالیت غیرعادی می‌توانند با کاهش وزن مواجه شوند که این اجازه می‌دهد که امتیاز به تدریج با روندهای جدید تطبیق یابد و تاثیر تغییرات ناگهانی را تعدیل کند.

۳. تشخیص ناهنجاری با وزن‌دهی پویا:

پیاده‌سازی سیستمی که الگوهای غیرمعمول در ارسال امتیازات را تشخیص دهد، مانند افزایش ناگهانی در امتیازات پایین یا بالا. در صورت تشخیص ناهنجاری، سیستم به صورت پویا وزن این امتیازات را تنظیم می‌کند و تاثیر آن‌ها را بر امتیاز کلی کاهش می‌دهد. که مزیت این روش این است که تضمین می‌کند که سیستم امتیازدهی در برابر حملات هماهنگ مقاوم باقی بماند.

۴. وزن‌دهی براساس User Reputation :

اختصاص وزن‌های مختلف به امتیازات کاربران براساس اعتبار آن‌ها. کاربران با اعتبار بالاتر (مانند کاربران فعال طولانی‌مدت یا حساب‌های تایید شده) تاثیر بیشتری بر امتیاز دارند نسبت به حساب‌های جدید یا کم‌فعال.

۵. تاخیر در اعمال امتیازات:

معرفی تاخیری در اعمال امتیازات جدید به امتیاز کلی، به‌ویژه در دوره‌های فعالیت بالا. این تاخیر به سیستم فرصت می‌دهد تا امتیازات را برای شناسایی دستکاری‌ها تحلیل کند قبل از اینکه در امتیاز کلی مقاله تاثیر بگذارند.

روش پیاده‌سازی شده

به دلیل زمان محدود برای این تسک نتوانستیم همه روش ها را پیاده سازی کنیم و برای این بخش ما ترکیبی از تشخیص ناهنجاری با وزن‌دهی پویا و Rate-Limiting by Time Window را پیاده‌سازی کردیم:

۱. تشخیص ناهنجاری و وزن‌دهی پویا:

در این بخش، سیستم به طور مستمر امتیازات داده شده به هر مقاله را در بازه‌های زمانی مشخص بررسی می‌کند. در صورتی که تعداد امتیازات در یک بازه زمانی (مثلاً ۱۵ دقیقه) از حد معین تجاوز کند و میانگین این امتیازات به‌طور غیرعادی بالا یا پایین باشد، به عنوان ناهنجاری شناسایی می‌شود. امتیازاتی که در این بازه زمانی ثبت شده‌اند، با وزن کمتری در محاسبات نهایی لحاظ می‌شوند.

۲. ذخیره وزن‌ها در کش Redis:

برای اینکه تأثیر ناهنجاری‌ها در محاسبات بعدی حفظ شود، وزن‌های اختصاص داده شده به امتیازات در کش Redis ذخیره می‌شوند. هر بار که امتیازی جدید ثبت می‌شود، این وزن‌ها بازیابی و به‌روز می‌شوند، تا در محاسبات نهایی، تأثیر مناسب داشته باشند.

۳. پردازش دسته‌ای با Celery:

یک وظیفه (Task) در Celery تعریف شد که هر ۱۵ دقیقه اجرا می‌شود. این وظیفه به محاسبه مجدد میانگین امتیازات و تعداد امتیازات برای هر مقاله می‌پردازد و در صورت شناسایی ناهنجاری، وزن‌های امتیازات را تنظیم می‌کند. نتایج این پردازش به‌روزرسانی شده و در کش ذخیره می‌شوند.

۴. محدودسازی نرخ امتیازدهی:

برای جلوگیری از ثبت امتیازات مکرر توسط یک کاربر در بازه زمانی کوتاه، از مکانیزم Throttling استفاده شد. این محدودیت به صورت پیش‌فرض برای هر کاربر تعیین شده که نمی‌تواند بیش از ۳ امتیاز در هر دقیقه ثبت کند.