دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

پروژه پایانی

گزارش کار پروژه پایانی درس پایگاه داده

نگارش

زهرا قصابی، پردیس مرادی، یاسمین رجبی ریسه، سید احمد موسوی اول

استاد

مریم رمضانی

تیر ۱۴۰۴

فهرست مطالب

فصل ۱

nyc taxi dataset analyse

۱-۱ انتخاب فناوری، پیش‌پردازش و ذخیره‌سازی دیتاست

۱-۱-۱ چرا OLAP ؟

دادهٔ NYC Taxi حجیم است، محورِ زمان دارد و بیشتر کار ما جمع‌بندی (GROUP BY) و فیلترهای زمانی/مکانی است؛ یعنی بار کاری Read-Heavy با اسکن و تجمیع. پایگاه‌های ردیف‌محور (OLTP) مثل PostgreSQL برای تراکنش‌های ریز و خواندن/نوشتن رکوردی عالی‌اند، اما وقتی به «اسکن‌های بزرگ + تجمیع» می‌رسیم هزینهٔ I/O و CPU بالا می‌رود. موتورهای ستونی OLAP دقیقاً این‌جا می‌درخشند: فقط ستون‌های لازم را می‌خوانند، با اجرای برداری و فشرده‌سازی مناسب، سرعت پاسخ را به شکل محسوسی پایین می‌آورند.

۱-۱-۲ انتخاب : ClickHouse

برای این پروژه ClickHouse را انتخاب کردیم به دلایل زیر:

- **ستونی و کم‌مصرف در I/O**: فقط ستون‌های لازمِ پرس‌وجو خوانده می‌شود و فشرده‌سازی LZ4/ZSTD پهنای خواندن را کم می‌کند.
- **برداری و GROUP BY سریع**: روی میلیون‌ها ردیف جمع‌زدن و شمارش کردن زمان کمی می‌گیرد.
- **پارتیشن زمان‌محور**: با خانوادهٔ MergeTree می‌توان بر اساس ماه/روز پارتیشن کرد و ORDER

BY را طوری چید که بُرش‌های زمانی سریع شوند.

- **سازگاری با SQL و ابزارها:** بارگذاری مستقیم Parquet، درایور پایتون و ابزارهای جانبی بدون دردسر.

نمونهٔ کوئری‌های ما. چیزهایی مثل: تعداد سفر در روز/ساعت، توزیع کرایه و مسافت، الگوی سفر به تفکیک ناحیهٔ شروع/پایان، و صدک‌ها. این‌ها همه «خواندنیِ زیاد + تجمیع» هستند و ClickHouse برای همین الگو بهینه است.

۱-۱-۳ گزینه‌های دیگر و دلیل رد آنها

PostgreSQL (ردیف‌محور). مزیت‌ها: تراکنشی، غنای SQL، اکوسیستم قوی. ضعف در این سناریو: برای اسکن‌های بزرگ و GROUP BYهای سنگین باید یا شاخص‌های متعدد ساخت (که نگهداری می‌خواهد) یا هزینهٔ I/O را پذیرفت. برای گزارش‌گیری حجیم، بدون شاردینگ/افزونه‌ها معمولاً کندتر می‌شود.

Apache Druid (ستونی). مزیت‌ها: رول‌آپ زمان‌محور، کش، تاخیر پایین برای داشبورد. در عوض مدل داده و مسیر ورودی‌اش اختصاصی‌تر است، JOINها محدودتر، و عملیاتی‌کردنش نسبت به ClickHouse سخت‌تر است.

سرویس‌های ابری .(BigQuery/Redshift/Snowflake) از نظر کارایی خوبند و مدیریت‌شده هستند، اما هزینهٔ مصرفی، وابستگی به فروشنده و اتصال دائم ابری دارند. برای اجرای محلی/کلاسی، ClickHouse به‌صرفه‌تر و قابل‌کنترل‌تر است.

۱-۱-۴ طراحی ذخیره‌سازی و شِما

برای جدول خام ny_taxi_trips کار های زیر را انجام دادیم:

- **موتور MergeTree** با PARTITION BY toYYYYMM(pickup_datetime) تا حذف/مدیریت بازه‌ای راحت شود.
- **ORDER BY (pickup_datetime, vendor_id)** تا اسکن‌های بازهٔ زمانی و گروه‌بندی‌های رایج روی زمان سریع شوند.

- **انواع درستِ ستون‌ها:** Decimal/Int/DateTime برای کرایه‌ها؛ LowCardinality برای ستون‌های کم‌تنوع (کد ناحیه/کدراننده).
- **نماهای مادی:** پیش‌تجمیع روزانه/ساعتی (تعداد سفر، میانگین‌ها، صدک‌ها) روی AggregatingMergeTree.

۱-۱-۵ پیش‌پردازش داده

۱. **پاک‌سازی و یکدست‌سازی:** تایپ‌کردن ستون‌های زمانی/عددی، حذف مقادیر پرت (مسافت/کرایهٔ غیرواقعی).

۲. **ویژگی‌های کمکی:** استخراج date، hour و روز هفته؛ نگاشت ناحیه‌ها؛ ساخت باکت برای کرایه/مسافت جهت گزارش‌های سریع.

۳. **ورود از Parquet:** چون ستونی است، هم‌خوان با موتور و سریع؛ ورود به‌صورت batchو موازی برای کاهش سربار

۱-۱-۶ راهبرد کارایی پرس‌وجو

در ClickHouse خبری از ایندکس B-Tree سنتی نیست؛ اما:

- **PRIMARY KEY/ORDER BY** مسیرخوانی را محدود می‌کند و روی WHEREهای زمانی خیلی تاثیرگذار است.
- **Data Skipping Index**ها (مثلاً minmax/bloom) برای فیلترهای خاص کمک می‌کند.
- **نماهای مادی/Projection** برای الگوهای پرتکرار GROUP BY، زمان پاسخ را به‌وضوح پایین می‌آورد.

۱-۲ بهینه‌سازی کوئری‌ها

از ابتدا سعی داشتیم که تمامی کوئری‌ها اعم از عادی و بهینه شده را روی یک جدول انجام شوند در نتیجه با این محدودیت مواجه شدیم که تمام بهینه‌سازی‌ها باید قابلیت فعال و غیرفعال بودن در اجرای سناریوها را داشته باشند.

در ابتدای کار قصد داشتیم چهارنوع بهینه‌سازی را روی پایگاه داده‌مان اعمال کنیم که این چهار مورد عبارتند از:

- استفاده از فشرده‌سازی برای ذخیره‌ی داده‌ها روی ستون‌های جدول
- ایندکس گذاری روی جدول
- تعریف projection روی جدول
- تعریف materialized view

وقتی در عمل به سراغ پیاده‌سازی موارد بالا رفتیم به این مورد رسیدیم که در click house امکان تعریف index و همچنین فشرده‌سازی روی materialized view وجود ندارد و تنها میتوان آنها را روی جدول اصلی اعمال کرد در این صورت تاثیر بهینه‌سازی ناشی از آنها هم روی حالت عادی و هم روی حالت بهینه‌شده مشهود میشد که برخلاف خواسته‌ی‌مان بود پس ناچار شدیم از آنها چشم بپوشیم و تنها دو مورد باقی‌مانده را پیاده‌سازی کنیم.

برای تعریف materialized view های موثر از الگوهای مشترک میان کوئری‌ها استفاده کردیم که نتایج آن به صورت زیر شد:

۱. کوئری‌های ۱، ۶ و ۸ روی تاریخ یک روز partition بندی میشوند و به تعداد سفرها و P90-distance در هر روز نیاز دارند به همین خاطر یک materialized view با partition بندی روی تاریخ و ستون‌های از پیش aggregate شده برای برآورده کردن خواسته‌ی کوئری‌ها تعریف کردیم.

۲. در کوئری ۱۰ نیاز به GROUP BY روی تاریخ روزانه و VendorID داریم که برای برآورده کردن آن materialized view مورد نیاز را تعریف کردیم.

۳. در کوئری‌های ۳، ۵، ۷ و ۹ روی حداقل یکی از ستون‌های pulocation-id و dolocation-id از GROUP BY استفاده شده است که برای بهینه کردن آن یک materialized view با GROUP BY روی موقعیت‌های مکانی و aggregate روی تعداد سفرها، مسافت پیموده شده، هزینه‌ی نهایی سفر و درصد انعام به کل هزینه استفاده کردیم.

۴. در کوئری دوم نیاز به میانگین هزینه به ازای هر VendorID داشتیم و از آنجایی که این مورد تنها نیازمند یک aggregate ساده روی جدول می‌باشد لذا یک projection روی جدول اصلی تعریف کردیم.

۵. در کوئری چهارم نیز نیاز به average گرفتن روی ستون tip به ازای هر روش پرداخت داریم که این مورد را نیز با تعریف یک projection روی جدول برآورده کردیم.

۳-۱ پایش و سنجش انتخاب

برای اطمینان از optimize شدن با انتخابمان، هنگام اجرای سناریوها این‌ها را اندازه می‌گیریم:

- **KPIها:** Latency (زمان اجرا از دید کلاینت) و Throughput (ردیف بر ثانیه).
- **سه‌فازهٔ منابع در کلاینت/سیستم:** حافظه (RSS)، CPU%، تعداد تردها، تعداد FDهای باز و نرخ شبکه (KB/s) در حالت‌های Pre/During/Post.

چرایی انتخاب متریک ها

- **Memory (RSS):** نشان‌دهندهٔ بافرینگ سمت کلاینت و بزرگی خروجی؛ افت پس از اجرا طبیعی است، ماندگاری یعنی نشت یا نگه‌داری بی‌جا.
- **CPU%:** اگر حین اجرا بالا برود یعنی گلوگاه محاسباتی/رقابت پردازنده داریم؛ مقایسه با Pre/Post خط پایه را روشن می‌کند.
- **Threads (count):** بازتاب الگوی همزمانی کلاینت؛ افزایش بی‌دلیل یا باقی‌ماندن بعد از اجرا نشانهٔ نشت ترد یا تنظیم نادرست pool است.
- **FDs Open (count):** کیفیت مدیریت اتصال/سوکت؛ اگر Post بالا بماند یعنی connection leak داریم.
- **Network Rate (KB/s):** حجم و سرعت انتقال نتیجه بین سرور و کلاینت؛ اوج در During یعنی خروجی حجیم یا گلوگاه شبکه.

فصل ۲

اجرای پروژه با شبکه

برای اجرای پروژه با چند کلاینت و اضافه‌کردن بخش شبکه، ابتدا کدی برای شبیه‌سازی کلاینت‌هایی که قصد پیاده‌سازی داشتیم نوشته شد. سپس سرور پروژه به دو روش مختلف شبیه‌سازی گردید:

۱. در حالت ساده، بدون هیچ بهینه‌سازی خاص.

۲. در حالت بهینه‌سازی‌شده با استفاده از تکنیک‌های زیر:

- Connection Pooling
- Application-Level Caching
- Asynchronous Processing & Queuing

این روش‌ها برای بهبود کارایی و مدیریت بهتر استفاده همزمان چند کلاینت از پایگاه داده به کار گرفته شدند.

۱-۲ سرور ساده Simple Logic Server

در این حالت، به ازای هر درخواستی که از یک کلاینت به سرور ارسال می‌شود، یک اتصال مستقیم به پایگاه داده گرفته می‌شود و کوئری‌های مورد نیاز همان لحظه اجرا می‌گردند.

۲-۲ سرور بهینه‌سازی‌شده Optimized Logic Server

در این روش، ابتدا تعداد مشخصی اتصال به پایگاه داده ایجاد و در قالب یک Connection Pool نگهداری می‌شود.

سپس هر درخواستی که از سمت کلاینت‌ها دریافت گردد، به‌جای اجرای مستقیم، در یک صف ذخیره می‌شود. در ادامه، چندین Thread مجزا مسئول پردازش این صف هستند. هر زمان که یکی از اتصال‌های موجود در Pool آزاد شود، کوئری با بیشترین اولویت (بر اساس معیارهایی نظیر Priority و زمان انتظار) از صف خارج شده و برای اجرا به پایگاه داده ارسال می‌شود.

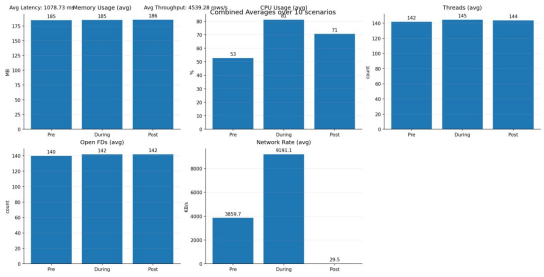
علاوه بر این، نتایج کوئری‌ها با استفاده از Redis کش می‌شوند و برای مدت زمان مشخصی در حافظه باقی می‌مانند. در نتیجه، اگر نتیجه یک کوئری قبلاً در کش ذخیره شده باشد، بدون نیاز به مراجعه مجدد به پایگاه داده یا صف پردازش، مستقیماً به کلاینت بازگردانده می‌شود.

فصل ۳

تحلیل نتایج

در نهایت برای سنجش مدل‌مان با استفاده از ۵ عدد Client و در طول ۱۰ سناریو که هر یک ترکیب‌های متفاوتی از کوئری‌هایمان در حالت عادی و بهینه (استفاده از متدهای معرفی شده در بخش ۱.۲ و Connection pool) هستند، خروجی مدل را بررسی کردیم.
نتایج اولیه به شرح زیر است:

- مقدار متریک average latency در حالت نرمال برابر با 45473.87 میلی ثانیه است ولی در حالت بهینه شده به مقدار 1078.73 میلی ثانیه کاهش می‌یابد. که با توجه به عددها میزان تسریع 42.15 است.
- مقدار متریک average throughput در حالت نرمال برابر با row/sec 101.92 است ولی در حالت بهینه شده به مقدار row/sec 4539.28افزایش می‌یابد با توجه به عددها میزان بهبود 44.53 است.
- از میان پنج متریک اصلی که برای سنجش عملکرد مدل تعریف کرده بودیم به جز Memory usage که در حالت عادی و بهینه تفاوتی تقریبا ناچیز دارد در سایر متریک‌ها افزایش چشم‌گیر در حالت بهینه نسبت به حالت عادی مشاهده میشود که دلیل آن این است که حالت بهینه با استفاده بیشتر از منابع باعث کاهش تاخیر ناشی از کوئری‌ها میشود.



شکل ۳-۱: Normal

شکل ۳-۲: Optimized

نتایج مشاهده شده در تصویر بالا حاصل میانگین‌گیری روی ۱۰ سناریو می‌باشند و نتایج هر سناریو به طور جداگانه در پوشه‌ی /results/. در فایل اصلی پروژه موجود می‌باشد.