弹幕视频网站应用项目报告

本项目开源在这儿。

1、应用需求

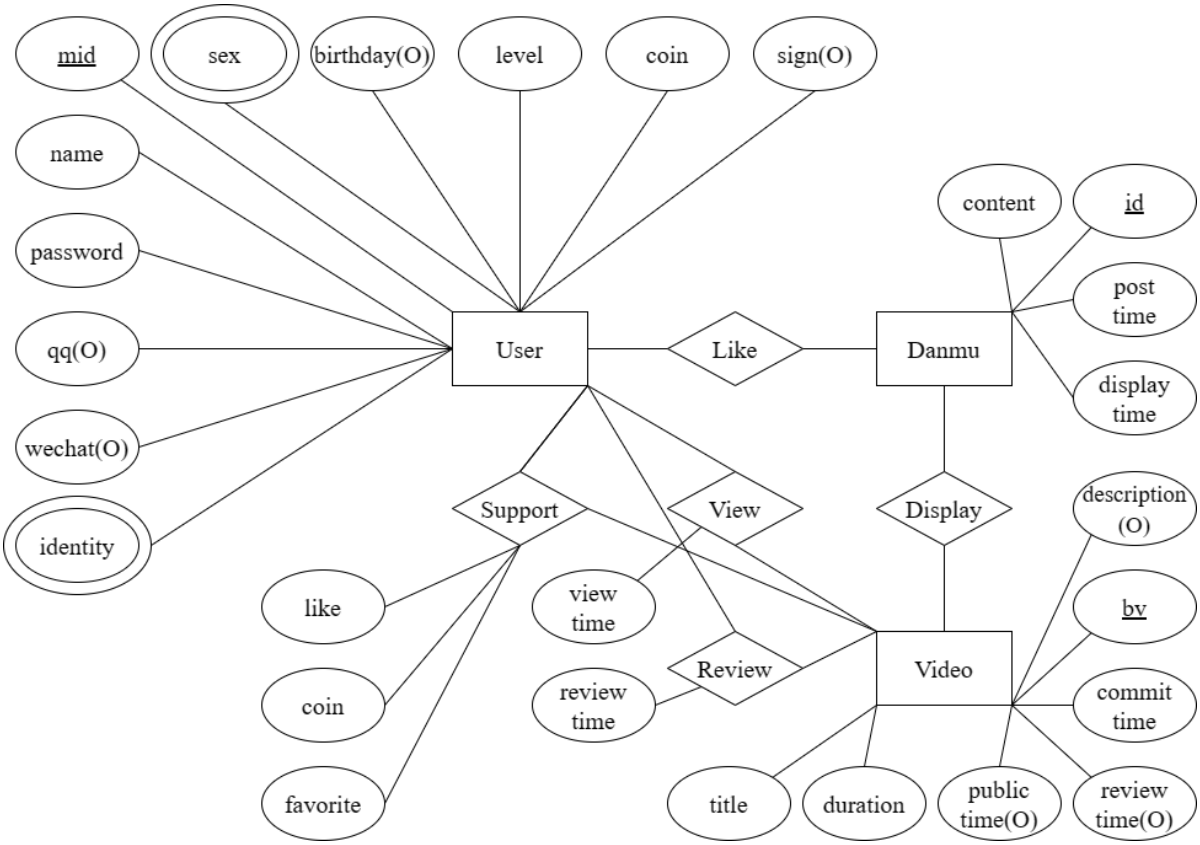
本项目要求使用SpringBoot框架与Postgresql共同实现简单的弹幕视频网站应用后端。该应用要求能处理用户、视频、弹幕三类数据，包括插入、删除、搜索和创建关系。根据现实中弹幕视频网站应用需求和提供的测试数据，用户数量应大于视频数量、弹幕数量应大于用户数量。但在本次项目中，提供的用户数据和弹幕数据均在 30000 左右，视频数据在 400 左右。因此数据库架构与更契合现实的架构略有不同。

在弹幕视频网站应用中，估计查询（如搜索、观看视频、显示弹幕）与修改（如上传视频、修改信息）的分布是偏向查询。因此数据库需要对高频查询有辅助表，对少量的修改可以加入触发器以简化业务逻辑。对特别高频的查询，如视频推荐等，还应当加入分层缓存，以进一步提高查询效率。

对于不同数据的关系，一个用户可以对多个用户有“关注”关系，在现实中较少有很高关注量，最高一般不超过 3000。但一个用户对视频的“观看”关系数量可以很高，例如达到 50000 以上。"点赞"、"投币"、"收藏"三个关系与"观看"类似，同样可以达到 10^4 量级。用户对弹幕的"喜爱"关系可以更高，因为可以对一个视频内多个弹幕"喜爱"。因此单用户的"喜爱"关系可以达到 10^5量级以上。因此，对于这些基于用户的关系，需要对数据表有所分区，以加速对用户和视频的查询。

2、数据库设计

应用ER图

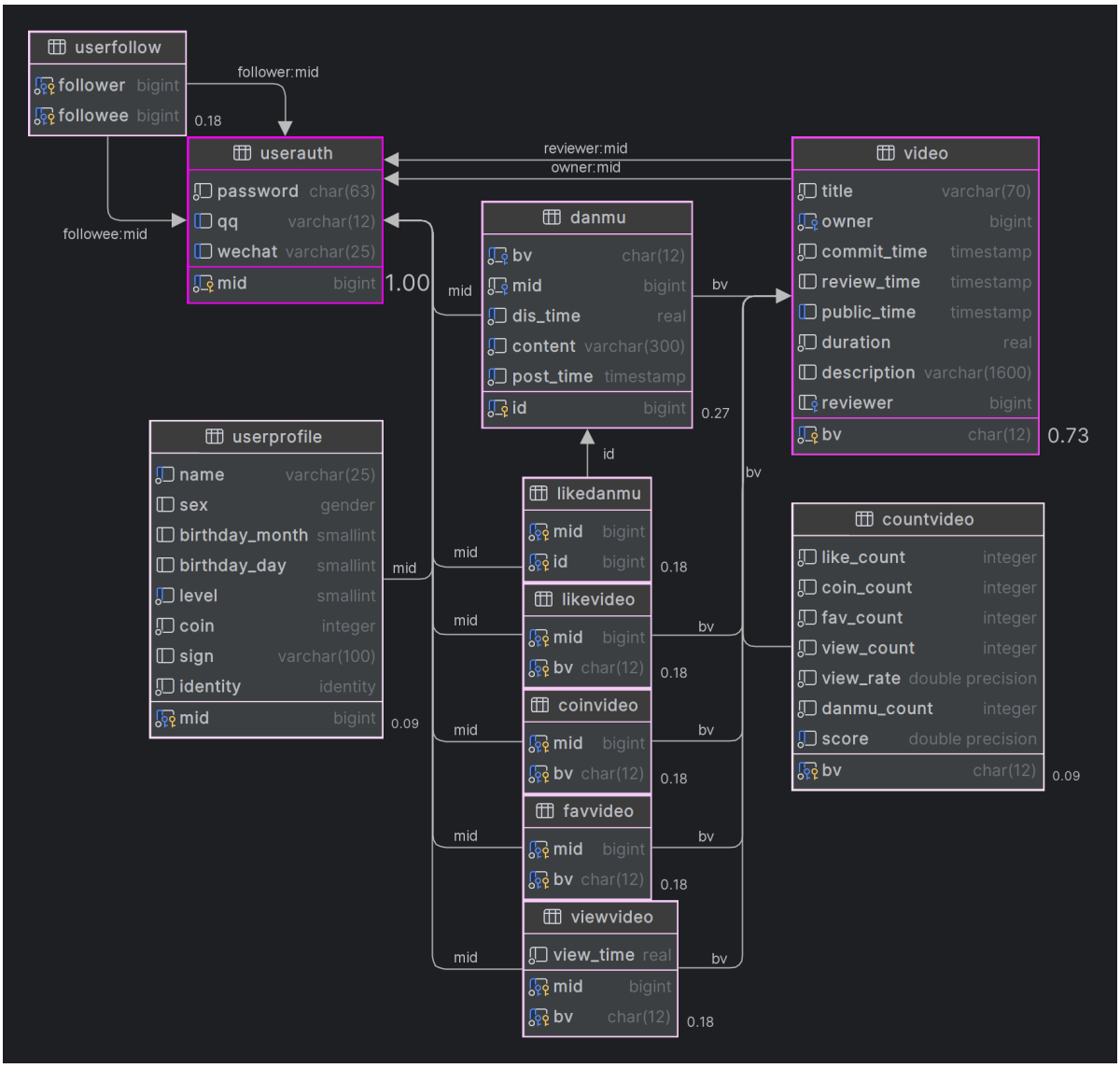


逻辑设计

基于以上考虑，我在本项目中的数据库系统架构设计如下：
注：未注明的表均为普通表。

- 静态信息表
  - 对用户创建两个表 `UserAuth` 与 `UserProfile`，前者存储 `mid, password, qq, wechat`，仅用于用户认证，并作为核心表，提供键值 `mid` 让其他与用户有关数据表依赖。注意，此处的 `password` 字段不存储明文密码，仅存储经过Argon2算法编码后的结果。后者存储 `mid, name, sex, birthday_month, birthday_day, level, coin, sign, identity` 为个人的所有静态信息。
  - 对视频创建一个表 `Video`，存储 `bv, title, owner, commit_time, review_time, public_time, duration, description, reviewer` 为单个视频除弹幕外的所有静态信息，并作为核心表，提供键值 `bv` 让其他与视频有关数据表依赖。
  - 对弹幕创建一个表 `Danmu`，存储 `id, bv, mid, dis_time, content, post_name` 为一条弹幕的静态信息，并作为核心表，提供键值 `id` 让其他与弹幕有关数据表依赖。
- 关联表
  - 关联表均有分区，在部分查询中可减少单表数据量提高速度，在另一部分查询中可充分利用多线程在不同表内同时查询提高速度。
  - 对用户创建一个表 `UserFollow`，有键值 `follower, followee`，均依赖 `UserAuth.mid`，存储"关注"关系，且以 `follower` 为分区键基于 `Hash` 四分区。
  - 对视频创建四个表 `LikeVideo, CoinVideo, FavVideo, ViewVideo`，其中 `bv` 依赖 `Video`，`mid` 依赖 `UserAuth`，存储"点赞""投币""收藏""观看"四种关系，前三个表均以 `mid` 为分区键建立四分区，最后一个表建立八分区。
  - 对弹幕创建一个表 `LikeDanmu`，其中 `id` 依赖 `Danmu`，`mid` 依赖 `UserAuth`，存储"喜爱"关系，以 `id` 为分区键。此处不以 `mid` 为分区键的原因是对喜爱弹幕的统计一般出现在视频中，网页渲染会将高喜爱量的弹幕添加点赞标识。由于 `id` 在 `Danmu` 中为主键，以 `id` 分区可以提高 `Danmu` 与 `LikeDanmu` 的 `JOIN` 性能。
- 辅助表
  - 对视频的推荐功能创建一个表 `CountVideo`，以统计特定 `bv` 的各项数据，用于提高视频推荐结果生成速度和分页查询速度。当可能存在其他缓存技术如Redis时，应启用并对热点数据缓存。
- 缓存表
  - 视频搜索时会创建 `UNLOGGED` 类型的 `PublicVideo` 表，该表作为查询结果的缓存表，存储上一次查询的结果。
  - 在更新 `PublicVideo` 表时，在会话中会创建 `TEMP` 类型的 `TempVideo` 表，用于关键词匹配不在缓存中的数据，之后再与缓存表合并供查询。

数据库图



注：

1. 中间列五个相连的关联表均有对应分区表，在图中未体现。
2. 各数据类型综合依照数据集和BiliBili网站限制设定。

3、业务实现

- 在任何修改数据库的操作中，均以事务包装，对任何查询命令，均不加入事务。
- 由于该应用侧重于查询，因此性能分析只针对必要的业务，省略简单语句分析。
- 对于每个业务的详细查询语句均位于 `DatabaseServiceImpl` 中，因此报告内省略具体查询语句内容。

数据导入

对于业务中实际调用次数极少的本模块（一般数据库初始化有且仅调用一次），我为了减少数据表间的耦合，选择对每个表单独进行初始化。注意，在输入数据集中，`UserAuth.password` 为明文，因此需要对其进行Argon2编码以提高安全性。由于性能测试时，编码时间也会被计入，因此在本次实现中，编码模块的参数为 `saltLength = 8, hashLength = 16, parallelism = 1, memory = 128, iterations = 1`。这个编码参数是不安全的，但编码速度相对较快，而默认的编码参数为 `saltLength = 16, hashLength = 32, parallelism = 1, memory = 4096, iterations = 3`。盐值和散列长度

减小是为了减少数据表的体积，内存使用和迭代次数减少会极大提高编码速度，但对安全性影响较高。尽管如此，在Profiling中，编码时长仍占据大量程序CPU运行时间。

此外，在导入时我使用 `COPY` 命令以加速数据插入性能，在数据表创建完毕后再创建索引和相关约束。对于每次 `COPY` 命令传输的数据量根据表的不同有所区别，静态信息表选择 `1000` 个数据作为一次提交，关联表选择 `10000` 个数据作为一次提交。分批大小的选择会一定程度影响导入性能。对于关联表，从较小的 `1000` 到较大的 `400000` 均有抽样测试，最大优化幅度是 17% 左右，体现为接近 10 秒的时间优化。

导入数据基本信息

性能分析快照位于仓库中 `snapshot` 目录下。`public` 所有表空间大小（包括索引表）：$2691MB$。各数据表行数据量：

表名	行数据量
ViewVideo	8515433
UserFollow	5958770
LikeVideo	4501094
CoinVideo	4151635
FavVideo	4079928
LikeDanmu	1257908
UserAuth	37881
UserProfile	37881
Danmu	32672
Video	407
CountVideo	407

单线程导入性能分析

1. 根据基于JVM的插桩分析，Argon2编码时长占到总体时长 50% 以上。而且在Argon2编码实现中，我已经使用Java自带的多线程库ForkJoinWorker。在未开启采样时，受本机所限，并行库能以 6 线程执行，将CPU负载提升到 95% 以上。因此这段耗时是无法再优化，是数据导入过程的主要瓶颈。
2. 除编码外，在数据库导入时，我使用 `COPY` 命令导入数据。数据导入被捕获在显示为空白的行内，因为该命令通过jdbc的回调函数实现，无法被探针捕获。探针只能基于 `jdbc.execute()` 汇总性能分析信息，因此位于空白分组内。在此前提下，对 `ViewVideo` 的限制建立过程耗时极高，`UserFollow` 和 `LikeVideo` 紧随其后。这是由于这三个表的数据量最高，需要较长时间建立索引。但由于不同表之间有层级依赖关系，例如 `ViewVideo` 和 `UserFollow` 在依赖关系中处于同层。因此可以考虑部分并行，以提高数据库系统使用率，加快整体数据导入速度。
3. 观察到 `ViewVideo` 的数据表过大，四分区时单表仍有 2×10^6 的数据量。因此在改进中会将该表的分区调整为八分区，以减少单表数据量，提高查询效率。

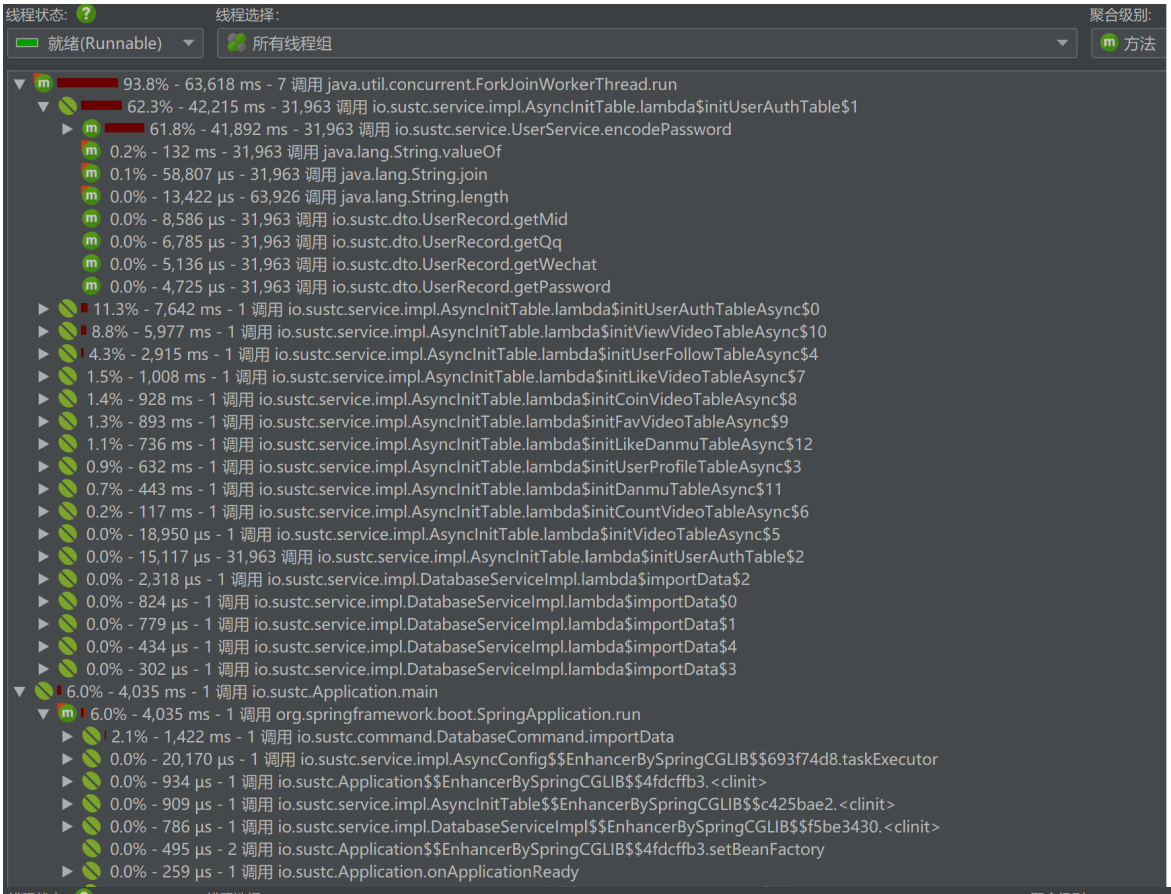
注1：性能分析基于JProfiler的插桩模式，对性能有一定影响，应关注各参数的比率而非绝对数值。
注2：此时对 `viewVideo` 的分区策略为四分区，根据性能分析结果后调整为八分区。

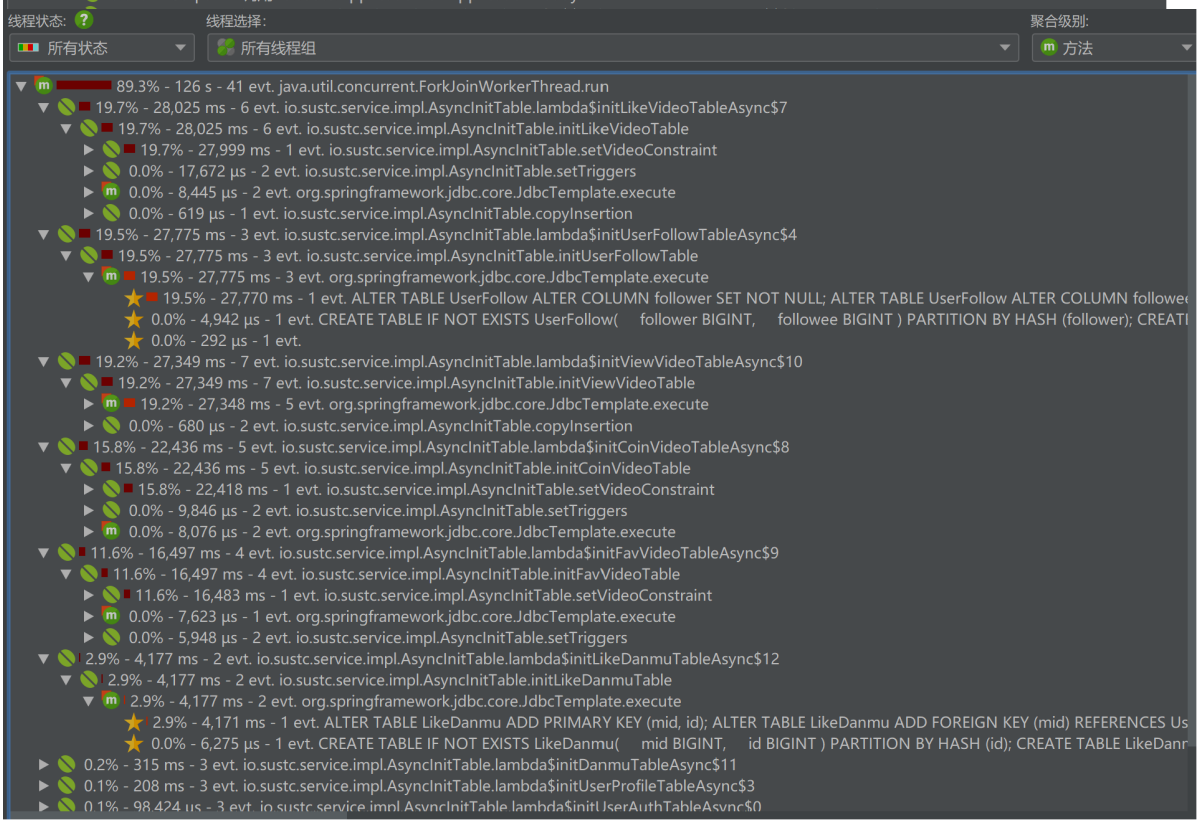
热点	时间	平均时间	事件
ALTER TABLE ViewVideo ALTER COLUMN view_time SET NOT NULL; ALTER TABLE ViewVideo...	18,729 ms (25 %)	18,729 ms	1
ANALYZE;	12,733 ms (17 %)	12,733 ms	1
ALTER TABLE UserFollow ALTER COLUMN follower SET NOT NULL; ALTER TABLE UserFollow ...	10,435 ms (14 %)	10,435 ms	1
ALTER TABLE LikeVideo ADD PRIMARY KEY (mid, bv); ALTER TABLE LikeVideo ADD FOREIGN ...	9,902 ms (13 %)	9,902 ms	1
ALTER TABLE FavVideo ADD PRIMARY KEY (mid, bv); ALTER TABLE FavVideo ADD FOREIGN ...	9,491 ms (12 %)	9,491 ms	1
ALTER TABLE CoinVideo ADD PRIMARY KEY (mid, bv); ALTER TABLE CoinVideo ADD FOREIG...	9,296 ms (12 %)	9,296 ms	1
ALTER TABLE LikeDanmu ADD PRIMARY KEY (mid, id); ALTER TABLE LikeDanmu ADD FOREIG...	2,609 ms (3 %)	2,609 ms	1
ALTER TABLE UserProfile ALTER COLUMN name SET NOT NULL; ALTER TABLE UserProfile ALT...	181 ms (0 %)	181 ms	1
SELECT setval(pg_get_serial_sequence(?, ?), (SELECT MAX(mid) FROM UserAuth)); ALTER TA...	126 ms (0 %)	126 ms	1
SELECT setval(pg_get_serial_sequence(?, ?), (SELECT MAX(id) FROM Danmu)); ALTER TABLE ...	109 ms (0 %)	109 ms	1
ALTER TABLE Video ALTER COLUMN title SET NOT NULL; ALTER TABLE Video ALTER COLUM...	10,175 µs (0 %)	10,175 µs	1
ALTER TABLE CountVideo ALTER COLUMN like_count SET NOT NULL; ALTER TABLE CountVid...	8,542 µs (0 %)	8,542 µs	1
DO $$ BEGIN IF NOT EXISTS (SELECT ? FROM pg_type WHERE typname = ?) THEN ...	5,504 µs (0 %)	5,504 µs	1
CREATE OR REPLACE FUNCTION calc_score() RETURNS TRIGGER AS $$ BEGIN IF NEW.vie...	5,119 µs (0 %)	5,119 µs	1
CREATE TABLE IF NOT EXISTS LikeVideo(mid BIGINT, bv CHAR(?)) PARTITION BY HAS...	4,965 µs (0 %)	4,965 µs	1
CREATE TABLE IF NOT EXISTS FavVideo(mid BIGINT, bv CHAR(?)) PARTITION BY HAS...	4,938 µs (0 %)	4,938 µs	1
CREATE TABLE IF NOT EXISTS LikeDanmu(mid BIGINT, id BIGINT) PARTITION BY HAS...	4,632 µs (0 %)	4,632 µs	1
CREATE TABLE IF NOT EXISTS UserFollow(follower BIGINT, followee BIGINT) PARTITIO...	4,590 µs (0 %)	4,590 µs	1
CREATE TABLE IF NOT EXISTS CoinVideo(mid BIGINT, bv CHAR(?)) PARTITION BY HAS...	4,166 µs (0 %)	4,166 µs	1
CREATE TABLE IF NOT EXISTS ViewVideo(mid BIGINT, bv CHAR(?), view_time REAL) ...	3,765 µs (0 %)	3,765 µs	1
CREATE TABLE IF NOT EXISTS Video(bv CHAR(?), title VARCHAR(?), owner BIGINT, ...	3,409 µs (0 %)	3,409 µs	1
CREATE OR REPLACE FUNCTION increase_view_count() RETURNS TRIGGER AS $$ BEGIN U...	2,912 µs (0 %)	2,912 µs	1
CREATE TABLE IF NOT EXISTS UserAuth (mid BIGSERIAL, password CHAR(?), qq VA...	2,881 µs (0 %)	2,881 µs	1
CREATE TABLE IF NOT EXISTS Danmu(id BIGSERIAL, bv CHAR(?), mid BIGINT, dis ...	2,807 µs (0 %)	2,807 µs	1
CREATE OR REPLACE FUNCTION increase_coin_count() RETURNS TRIGGER AS $$ BEGIN U...	2,798 µs (0 %)	2,798 µs	1
CREATE OR REPLACE FUNCTION increase_like_count() RETURNS TRIGGER AS $$ BEGIN UP...	2,380 µs (0 %)	2,380 µs	1
CREATE OR REPLACE FUNCTION increase_fav_count() RETURNS TRIGGER AS $$ BEGIN UP...	2,171 µs (0 %)	2,171 µs	1
CREATE TABLE IF NOT EXISTS CountVideo(bv CHAR(?), like_count INTEGER DEFAULT ?,...	1,901 µs (0 %)	1,901 µs	1
CREATE TABLE IF NOT EXISTS UserProfile(mid BIGINT, name VARCHAR(?), sex Gend...	1,623 µs (0 %)	1,623 µs	1
	1,237 µs (0 %)	618 µs	2
CREATE OR REPLACE FUNCTION get_hotspot(bv_value CHAR(?)) RETURNS TABLE (hotspot I...	1,222 µs (0 %)	1,222 µs	1
SET work_mem = ?; SET maintenance_work_mem = ?; SET effective_cache_size = ?; SET temp...	1,191 µs (0 %)	1,191 µs	1
CREATE OR REPLACE FUNCTION increase_danmu_count() RETURNS TRIGGER AS $$ BEGIN ...	1,092 µs (0 %)	1,092 µs	1
DO $$ BEGIN IF NOT EXISTS (SELECT ? FROM pg_type WHERE typname = ?) THEN ...	872 µs (0 %)	872 µs	1

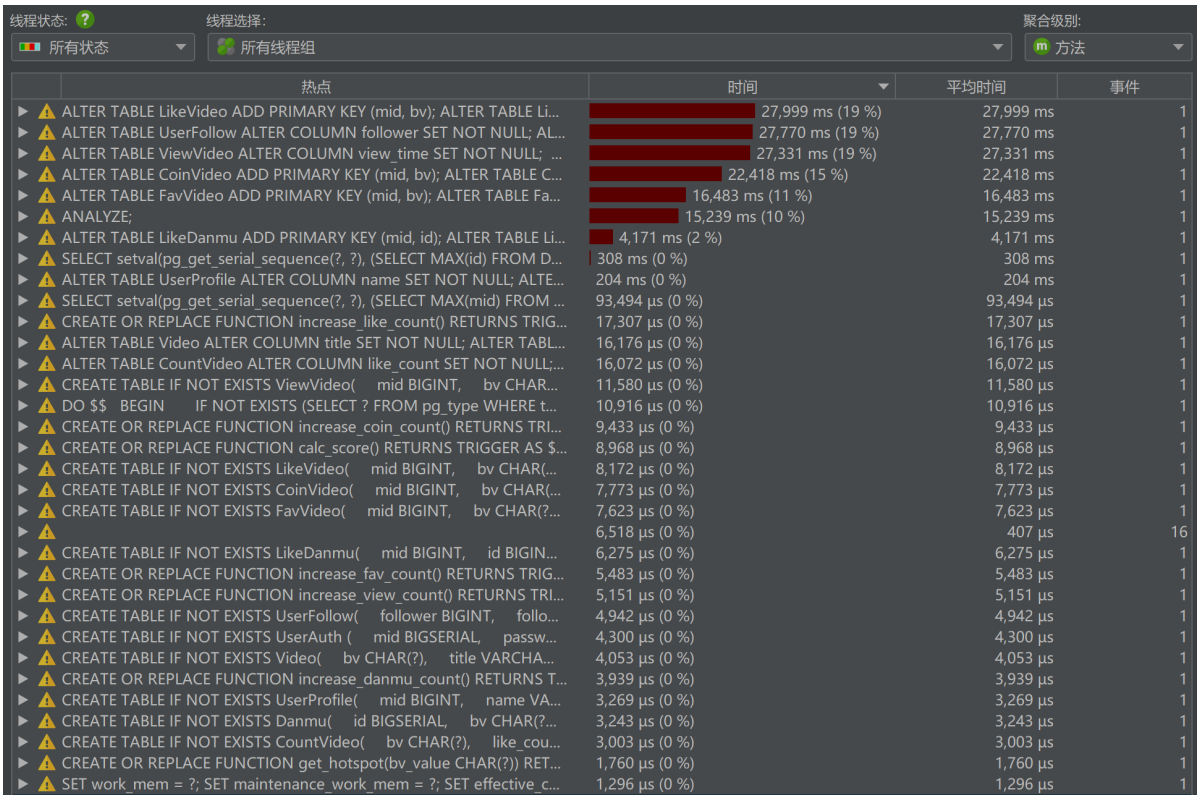
多线程导入性能分析

可以观察到，在根据拓扑关系使用多线程导入数据后，与数据库传输部分CPU耗时相对下降 30%，但由于数据库处理能力有限，执行命令的等待时间占比提升。在本机测试中，总运行时间为 55.741 秒。相比于导入部分全串行的运行时间 111.522 秒，加速比为 2.001，加速效率为 33.35%，实际加速效率与性能分析的估计加速性能接近，符合分析预期。其中，`ANALYZE` 命令不计入数据统计结果中，在提交代码中是不执行该语句的。提交代码的性能分析报告也在仓库目录中，由于主要瓶颈基本未发生变化，因此不重复说明。

注：多线程插桩对性能影响更高，再次强调绝对数值的意义不大。







热点	时间	平均时间	事件
ALTER TABLE LikeVideo ADD PRIMARY KEY (mid, bv); ALTER TABLE Li...	27,999 ms (19 %)	27,999 ms	1
ALTER TABLE UserFollow ALTER COLUMN follower SET NOT NULL; AL...	27,770 ms (19 %)	27,770 ms	1
ALTER TABLE ViewVideo ALTER COLUMN view_time SET NOT NULL; ...	27,331 ms (19 %)	27,331 ms	1
ALTER TABLE CoinVideo ADD PRIMARY KEY (mid, bv); ALTER TABLE C...	22,418 ms (15 %)	22,418 ms	1
ALTER TABLE FavVideo ADD PRIMARY KEY (mid, bv); ALTER TABLE Fa...	16,483 ms (11 %)	16,483 ms	1
ANALYZE;	15,239 ms (10 %)	15,239 ms	1
ALTER TABLE LikeDanmu ADD PRIMARY KEY (mid, id); ALTER TABLE Li...	4,171 ms (2 %)	4,171 ms	1
SELECT setval(pg_get_serial_sequence(?, ?), (SELECT MAX(id) FROM D...	308 ms (0 %)	308 ms	1
ALTER TABLE UserProfile ALTER COLUMN name SET NOT NULL; ALTE...	204 ms (0 %)	204 ms	1
SELECT setval(pg_get_serial_sequence(?, ?), (SELECT MAX(mid) FROM ...	93,494 µs (0 %)	93,494 µs	1
CREATE OR REPLACE FUNCTION increase_like_count() RETURNS TRIG...	17,307 µs (0 %)	17,307 µs	1
ALTER TABLE Video ALTER COLUMN title SET NOT NULL; ALTER TABL...	16,176 µs (0 %)	16,176 µs	1
ALTER TABLE CountVideo ALTER COLUMN like_count SET NOT NULL;...	16,072 µs (0 %)	16,072 µs	1
CREATE TABLE IF NOT EXISTS ViewVideo(mid BIGINT, bv CHAR...	11,580 µs (0 %)	11,580 µs	1
DO $$ BEGIN IF NOT EXISTS (SELECT ? FROM pg_type WHERE t...	10,916 µs (0 %)	10,916 µs	1
CREATE OR REPLACE FUNCTION increase_coin_count() RETURNS TRI...	9,433 µs (0 %)	9,433 µs	1
CREATE OR REPLACE FUNCTION calc_score() RETURNS TRIGGER AS $...	8,968 µs (0 %)	8,968 µs	1
CREATE TABLE IF NOT EXISTS LikeVideo(mid BIGINT, bv CHAR(...	8,172 µs (0 %)	8,172 µs	1
CREATE TABLE IF NOT EXISTS CoinVideo(mid BIGINT, bv CHAR(...	7,773 µs (0 %)	7,773 µs	1
CREATE TABLE IF NOT EXISTS FavVideo(mid BIGINT, bv CHAR(?...	7,623 µs (0 %)	7,623 µs	1
	6,518 µs (0 %)	407 µs	16
CREATE TABLE IF NOT EXISTS LikeDanmu(mid BIGINT, id BIGIN...	6,275 µs (0 %)	6,275 µs	1
CREATE OR REPLACE FUNCTION increase_fav_count() RETURNS TRIG...	5,483 µs (0 %)	5,483 µs	1
CREATE OR REPLACE FUNCTION increase_view_count() RETURNS TRI...	5,151 µs (0 %)	5,151 µs	1
CREATE TABLE IF NOT EXISTS UserFollow(follower BIGINT, follo...	4,942 µs (0 %)	4,942 µs	1
CREATE TABLE IF NOT EXISTS UserAuth (mid BIGSERIAL, passw...	4,300 µs (0 %)	4,300 µs	1
CREATE TABLE IF NOT EXISTS Video(bv CHAR(?), title VARCHA...	4,053 µs (0 %)	4,053 µs	1
CREATE OR REPLACE FUNCTION increase_danmu_count() RETURNS T...	3,939 µs (0 %)	3,939 µs	1
CREATE TABLE IF NOT EXISTS UserProfile(mid BIGINT, name VA...	3,269 µs (0 %)	3,269 µs	1
CREATE TABLE IF NOT EXISTS Danmu(id BIGSERIAL, bv CHAR(?...	3,243 µs (0 %)	3,243 µs	1
CREATE TABLE IF NOT EXISTS CountVideo(bv CHAR(?), like_cou...	3,003 µs (0 %)	3,003 µs	1
CREATE OR REPLACE FUNCTION get_hotspot(bv_value CHAR(?)) RET...	1,760 µs (0 %)	1,760 µs	1
SET work_mem = ?; SET maintenance_work_mem = ?; SET effective_c...	1,296 µs (0 %)	1,296 µs	1

用户服务

- 注册
  - 应用层校验各项数据符合要求后，返回序列值 `mid`
  - 数据库操作三次 `UserAuth`：查询 `qq` 和 `wechat` 是否存在、查询 `name` 是否存在、插入新用户
- 删除
  - 身份校验
    - 数据库获取一次 `UserAuth` 对应字段后应用层校验
  - 身份校验后对比用户权限，返回删除结果
  - 数据库操作四次 `UserAuth`：查询待删除用户是否存在、查询当前权限、查询待删除用户权限、删除用户
- 获取信息
  - 应用层调用多线程查询，返回用户信息
  - 数据库操作八次
    - 一次 `UserAuth` 查询指定 `mid` 是否存在
    - 五次关联表中查询指定 `mid` 的所有信息（异步）
    - 一次 `Video` 查询指定 `mid` 上传的视频（异步）
    - 一次 `UserProfile` 查询指定 `mid` 硬币数量
- 关注
  - 身份校验和目标检查后，返回关注结果
  - 数据库操作三次：`UserAuth` 查询被关注者 `mid` 是否存在、`UserFollow` 查询当前关注关系、修改 `UserFollow` 为指定关系（插入或删除）

弹幕服务

- 发送弹幕
  - 身份校验、视频存在校验、观看校验
  - 数据库操作三次：`Video` 获取视频持续时间、`ViewVideo` 获取观看记录、插入 `Danmu`
- 显示弹幕
  - 身份校验、视频存在校验、获取弹幕信息
  - 数据库操作两次
    - `Video` 获取视频持续时间
    - 查询 `Danmu`
      - 对弹幕总量最多的视频 `BV19e4y1y7HV` 执行全区间过滤查询的分析结果如下

> Unique (cost=494.07..496.52 rows=489 width=41) (actual time=0.497..0.608 rows=414 loops=1)
> -> Sort (cost=494.07..495.30 rows=490 width=41) (actual time=0.496..0.519 rows=494 loops=1)
> "   Sort Key: content, post_time"
>   Sort Method: quicksort Memory: 59kB
>   -> Bitmap Heap Scan on danmu (cost=18.66..472.18 rows=490 width=41) (actual time=0.129..0.228 rows=494 loops=1)
>   Recheck Cond: ((bv = 'BV19e4y1y7HV'::bpchar) AND (dis_time >= '0'::double precision) AND (dis_time <= '6688'::double precision))
>   Heap Blocks: exact=7
>   -> Bitmap Index Scan on danmubvdistimeindex (cost=0.00..18.54 rows=490 width=0) (actual time=0.116..0.116 rows=494 loops=1)
>   Index Cond: ((bv = 'BV19e4y1y7HV'::bpchar) AND (dis_time >= '0'::double precision) AND (dis_time <= '6688'::double precision))
> Planning Time: 0.215 ms
> Execution Time: 0.677 ms

- 喜爱弹幕
  - 身份校验、弹幕存在校验、观看校验
  - 数据库操作四次：`Danmu` 查询 `id` 对应 `bv`、`ViewVideo` 查询是否观看、`LikeDanmu` 查询喜爱状态、修改 `LikeDanmu` 为指定关系（喜爱或不喜爱）

视频服务

- 上传视频
  - 用户校验、视频存在校验
  - 数据库操作两次：`Video` 查询指定 `title` 是否存在、插入 `Video`
- 删除视频
  - 用户校验、权限检查
  - 数据库操作三次：`Video` 查询 `bv` 视频拥有者、`UserProfile` 查询 `mid` 当前权限、如果允许，删除 `Video` 指定 `bv`
- 更新视频
  - 用户校验、视频信息检查

    - 数据库操作三次：`Video` 查询 `bv` 视频拥有者、`Video` 查询 `bv` 视频原始信息、更新 `Video` 指定 `bv` 如果允许
- 查询平均观看比
    - 用户校验、视频观看校验
    - 数据库操作两次：`Video` 查询 `bv` 是否允许观看、`CountVideo` 查询 `view_rate` 和 `view_count`
- 查询视频热点
    - 用户校验、弹幕存在校验
    - 数据库操作两次
        - `Danmu` 查询指定 `bv` 是否有弹幕
        - `get_hotspot()` 自定义函数查询，对弹幕总量最多的视频执行查询分析结果如下

```
CTE Scan on bvcount  (cost=493.65..504.79 rows=2 width=4) (actual
time=0.435..0.451 rows=1 loops=1)
Filter: (count = $1)
Rows Removed by Filter: 289
CTE bvcount
 ->  HashAggregate  (cost=473.84..482.51 rows=495 width=12) (actual
time=0.295..0.339 rows=290 loops=1)
      Group Key: (floor((danmu.dis_time / '10'::double precision)))::integer
      Batches: 1 Memory Usage: 73kB
      ->  Bitmap Heap Scan on danmu  (cost=16.25..471.37 rows=495 width=12)
(actual time=0.065..0.184 rows=495 loops=1)
            Recheck Cond: (bv = 'BV19e4y1y7HV'::bpchar)
            Heap Blocks: exact=7
            ->  Bitmap Index Scan on danmubvdistimeindex  (cost=0.00..16.12
rows=495 width=0) (actual time=0.053..0.053 rows=495 loops=1)
                  Index Cond: (bv = 'BV19e4y1y7HV'::bpchar)
InitPlan 2 (returns $1)
 ->  Aggregate  (cost=11.14..11.15 rows=1 width=8) (actual time=0.126..0.126
rows=1 loops=1)
      ->  CTE Scan on bvcount bvcount_1  (cost=0.00..9.90 rows=495 width=8)
(actual
time=0.000..0.101 rows=290 loops=1)
Planning Time: 0.435 ms
Execution Time: 0.543 ms
```

- 审核视频
    - 用户校验、权限校验、相关检查
    - 数据库操作四次：`UserProfile` 查询 `mid` 权限、`Video` 查询 `bv` 的拥有者、`Video` 查询 `bv` 是否已被审核、修改 `Video` 审核指定 `bv`
- 三连
    - 用户校验、视频可三连检查、当前状态检查
    - 数据库操作四次（投币六次）
        - `Video` 查询 `bv` 拥有者
        - `Video` 查询 `bv` 审核状态

        - （`UserProfile` 查询当前硬币数量）
        - 关系表查询当前状态
        - 修改关系表
        - （修改 `UserProfile` 硬币数量）
- 搜索视频
    - 对指定搜索词的查询结果建立 `UNLOGGED` 缓存表并统计
    - 数据库操作四次以上
        - 创建或更新 `PublicVideo` 表，该表根据 `mid` 的可观看视频创建，性能分析是基于新建数据表

            ```
            Hash Join  (cost=29.85..745.66 rows=167 width=53) (actual time=0.301..7.149 rows=407 loops=1)
            Hash Cond: (video.bv = countvideo.bv)
            ->  Nested Loop  (cost=15.69..730.23 rows=167 width=309) (actual time=0.037..1.686 rows=407 loops=1)
                ->  Hash Anti Join  (cost=15.40..58.48 rows=167 width=305) (actual time=0.022..0.352 rows=407 loops=1)
                        Hash Cond: (video.bv = publicvideo.bv)
                        ->  Seq Scan on video  (cost=0.00..40.07 rows=407 width=305) (actual time=0.008..0.175 rows=407 loops=1)
                        ->  Hash  (cost=12.40..12.40 rows=240 width=13) (actual time=0.006..0.007 rows=0 loops=1)
                              Buckets: 1024 Batches: 1 Memory Usage: 8kB
                              ->  Seq Scan on publicvideo  (cost=0.00..12.40 rows=240 width=13) (actual time=0.006..0.006 rows=0 loops=1)
                ->  Index Scan using userprofile_pkey on userprofile  (cost=0.29..4.02 rows=1 width=20) (actual time=0.003..0.003 rows=1 loops=407)
                        Index Cond: (mid = video.owner)
            ->  Hash  (cost=9.07..9.07 rows=407 width=17) (actual time=0.162..0.163 rows=407 loops=1)
                Buckets: 1024 Batches: 1 Memory Usage: 28kB
                ->  Seq Scan on countvideo  (cost=0.00..9.07 rows=407 width=17) (actual time=0.015..0.090 rows=407 loops=1)
            Planning Time: 2.952 ms
            Execution Time: 7.303 ms
            ```

        - 若创建表则添加限制和索引，若更新表则删去不可观看、更新观看数、加入数据
            - 全表更新两次，以全 `Video` 表更新为例分析结果如下，结果存储于临时表 `TempVideo` 中

                ```
                Update on tempvideo  (cost=0.00..86.40 rows=0 width=0) (actual time=3.447..3.448 rows=0 loops=1)
                ->  Seq Scan on tempvideo  (cost=0.00..86.40 rows=2176 width=10) (actual time=0.010..1.018 rows=407 loops=1)
                Planning Time: 0.076 ms
                Execution Time: 3.472 ms
                ```

            - 当关键词相同时，在会话内使用临时表存储中间结果，减少对 `PublicVideo` 表的修改次数，但仍需要再将新数据插入 `PublicVideo` 中

> Insert on publicvideo (cost=0.00..80.64 rows=0 width=0) (actual time=4.590..4.590 rows=0 loops=1)
> -> Seq Scan on tempvideo (cost=0.00..80.64 rows=3264 width=92) (actual time=0.008..0.131 rows=407 loops=1)
> Planning Time: 0.050 ms
> Trigger for constraint publicvideo_bv_fkey: time=3.721 calls=407
> Execution Time: 8.367 ms

    - 查询分页结果，此处查询分析是全结果输出

> Sort (cost=37.10..37.15 rows=21 width=299) (actual time=0.066..0.067 rows=11 loops=1)
> " Sort Key: relevance DESC, view_count DESC"
> Sort Method: quicksort Memory: 31kB
> -> Bitmap Heap Scan on publicvideo (cost=4.44..36.64 rows=21 width=299) (actual time=0.030..0.039 rows=11 loops=1)
>   Recheck Cond: (relevance > 0)
>   Heap Blocks: exact=8
>   -> Bitmap Index Scan on publicvideoindex (cost=0.00..4.43 rows=21 width=0) (actualtime=0.016..0.016 rows=22 loops=1)
>   Index Cond: (relevance > 0)
> Planning Time: 0.092 ms
> Execution Time: 0.099 ms

- 注：在更新数据时，还需要对数据库操作关键词数量次，操作对象是 `PublicVideo` 或 `TempVideo`，取决于是第一次建表或者增量更新

推荐服务

- 通用视频推荐
    - 根据辅助表 `CountVideo` 的结构，可直接根据 `score` 字段返回结果。当辅助表无变化时，未来可以结合 `WHERE` 子句过滤加速分页结果，但由于应用性质和已有的索引，该筛选不能显著提高效率
    - 数据库查询一次，使用 `LIMIT` 和 `OFFSET` 完成分页
- 相关视频推荐
    - 在 `ViewVideo` 表中获取关联 `bv` 后，根据 `CountVideo` 的 `view_count` 排序返回结果
    - 数据库查询一次，对数据集播放量最高的 `BV1jL4y1e7Uz` 查询分析结果如下

> Limit (cost=52361.59..52361.61 rows=5 width=17) (actual time=2911.583..2911.597 rows=5 loops=1)
> InitPlan 2 (returns $1)
> -> Append (cost=51221.31..52347.27 rows=17964 width=13) (actual time=14.143..2227.799 rows=8189668 loops=1)
>   InitPlan 1 (returns $0)
>   -> Append (cost=0.43..51220.88 rows=35349 width=8) (actual time=0.063..6.762 rows=36428 loops=1)
>   -> Index Scan using viewvideo_1_bv_idx on viewvideo_1 (cost=0.43..6391.10 rows=4386 width=8) (actual time=0.062..0.791 rows=4616 loops=1)
>   Index Cond: (bv = 'BV1jL4y1e7Uz'::bpchar)
>   此处省略其余7个分区表的记录

```
      ->  Index Only Scan using viewvideo_1_pkey on viewvideo_1 vv_1
(cost=0.43..127.61 rows=2246 width=13) (actual time=14.142..231.195
rows=1038079 loops=1)
            Index Cond: (mid = ANY ($0))
            Heap Fetches: 0
      此处省略其余7个分区表的记录
->  Sort  (cost=14.32..14.35 rows=10 width=17) (actual time=2911.580..2911.581
rows=6 loops=1)
    Sort Key: cv.view_count DESC
    Sort Method: top-N heapsort Memory: 25kB
    ->  Seq Scan on countvideo cv  (cost=0.00..14.16 rows=10 width=17) (actual
time=2905.660..2911.460 rows=407 loops=1)
          Filter: (bv = ANY ($1))
Planning Time: 0.803 ms
Execution Time: 2925.670 ms
```

- 用户视频推荐
  - 鉴权后先使用一次查询判断是否有感兴趣的视频，作为方法调用条件。第二次查询使用两个子查询，分别求出 `friends` 列表和 `excluded_videos` 列表，再将其与 `ViewVideo` 连接查询合法视频 `bv`。最后将该中间结果与 `Video` 和 `UserProfile` 连接查询以获取排序依据，使用 `LIMIT` 和 `OFFSET` 组合输出结果
  - 数据库查询两次，测试 `mid` 参数是关注量最多的用户之一 `362870`，`OFFSET` 参数是根据结果取最后部分为 `170`，查询分析结果如下：

```
Limit  (cost=3462.19..3462.22 rows=10 width=31) (actual time=3.191..3.198 rows=5
loops=1)
->  Sort  (cost=3461.77..3462.75 rows=394 width=31) (actual time=3.183..3.193
rows=175 loops=1)
"       Sort Key: (count(vv.mid)) DESC, up.level DESC, v.public_time DESC"
    Sort Method: quicksort Memory: 37kB
    ->  HashAggregate  (cost=3430.27..3445.04 rows=394 width=31) (actual
time=3.070..3.102 rows=175 loops=1)
"            Group Key: vv.bv, up.level, v.public_time"
          Filter: (count(vv.mid) > 0)
          Batches: 1 Memory Usage: 97kB
          ->  Nested Loop  (cost=690.01..3418.45 rows=1182 width=31) (actual
time=0.580..2.778 rows=1216 loops=1)
              ->  Hash Join  (cost=689.71..2155.04 rows=1182 width=37) (actual
time=0.571..2.175 rows=1216 loops=1)
                    Hash Cond: (vv.bv = v.bv)
                    ->  Hash Anti Join  (cost=642.52..2104.71 rows=1182 width=21) (actual
time=0.395..1.778 rows=1216 loops=1)
                          Hash Cond: (vv.bv = viewvideo.bv)
                          ->  Nested Loop  (cost=623.38..2064.13 rows=2919 width=21)
(actual time=0.335..1.341 rows=3149 loops=1)
                                ->  Merge Join  (cost=622.95..733.40 rows=13 width=16) (actual
time=0.321..0.601 rows=14 loops=1)
                                      Merge Cond: (uf1.followee = uf2.follower)
                                      ->  Index Only Scan using userfollow_2_pkey on
userfollow_2 uf1  (cost=0.43..102.27 rows=3077 width=16) (actual
```

```
time=0.013..0.206 rows=2939 loops=1)
                                    Index Cond: (follower = 362870)
                                    Heap Fetches: 0
                                ->  Sort  (cost=622.52..622.91 rows=157 width=16) (actual
time=0.286..0.295 rows=188 loops=1)
                                    Sort Key: uf2.follower
                                    Sort Method: quicksort Memory: 32kB
                                    ->  Append  (cost=4.73..616.79 rows=157 width=16)
(actual time=0.017..0.253 rows=188 loops=1)
                                        ->  Bitmap Heap Scan on userfollow_1 uf2_1
(cost=4.73..153.03 rows=39 width=16) (actual time=0.016..0.049 rows=40 loops=1)
                                            Recheck Cond: (followee = 362870)
                                            Heap Blocks: exact=40
                                            ->  Bitmap Index Scan on
userfollow_1_followee_idx  (cost=0.00..4.72 rows=39 width=0) (actual
time=0.010..0.010 rows=40 loops=1)
                                                Index Cond: (followee = 362870)
                                        此处省略其余3个分区表的记录
                            ->  Append  (cost=0.43..84.37 rows=1799 width=21) (actual
time=0.010..0.040 rows=225 loops=14)
                                ->  Index Only Scan using viewvideo_1_pkey on viewvideo_1
vv_1  (cost=0.43..9.44 rows=225 width=21) (actual time=0.014..0.037 rows=245
loops=1)
                                    Index Cond: (mid = uf1.followee)
                                    Heap Fetches: 0
                                此处省略其余7个分区表的记录
                        ->  Hash  (cost=16.34..16.34 rows=224 width=13) (actual
time=0.054..0.054 rows=232 loops=1)
                            Buckets: 1024 Batches: 1 Memory Usage: 19kB
                            ->  Index Only Scan using viewvideo_6_pkey on viewvideo_6
viewvideo  (cost=0.42..16.34 rows=224 width=13) (actual time=0.012..0.029
rows=232 loops=1)
                                Index Cond: (mid = 362870)
                                Heap Fetches: 0
                    ->  Hash  (cost=42.10..42.10 rows=407 width=29) (actual
time=0.169..0.169 rows=407 loops=1)
                        Buckets: 1024 Batches: 1 Memory Usage: 34kB
                        ->  Seq Scan on video v  (cost=0.00..42.10 rows=407 width=29)
(actual time=0.013..0.118 rows=407 loops=1)
                            Filter: ((public_time IS NULL) OR (public_time <
LOCALTIMESTAMP))
                ->  Memoize  (cost=0.30..4.03 rows=1 width=10) (actual time=0.000..0.000
rows=1 loops=1216)
                    Cache Key: v.owner
                    Cache Mode: logical
                    Hits: 1063 Misses: 153 Evictions: 0 Overflows: 0 Memory Usage: 18kB
                    ->  Index Scan using userprofile_pkey on userprofile up
(cost=0.29..4.02 rows=1 width=10) (actual time=0.001..0.001 rows=1 loops=153)
                        Index Cond: (mid = v.owner)
```

> Planning Time: 3.992 ms
> Execution Time: 3.517 ms

- 用户关注推荐
  - 鉴权后，对 `UserFollow` 表查询特定关注关系，满足关注者不是指定 `mid` 的关注对象，且关注对象是 `mid` 的关注对象。对筛选出的关注关系分组统计排序，使用 `LIMIT` 和 `OFFSET` 组合输出结果
  - 数据库操作一次，测试参数 `mid` 是关注量最多的用户之一 `362870`，分页选择 `OFFSET` 为 `20000`，查询分析结果如下：

```
Limit  (cost=7260.80..7260.80 rows=1 width=18) (actual time=768.664..768.685
rows=10 loops=1)
InitPlan 1 (returns $0)
 ->  Index Only Scan using userfollow_2_pkey on userfollow_2 userfollow
(cost=0.43..102.27 rows=3077 width=8) (actual time=0.137..0.471 rows=3000
loops=1)
      Index Cond: (follower = 362870)
      Heap Fetches: 0
->  Sort  (cost=7157.87..7158.53 rows=263 width=18) (actual time=765.714..767.696
rows=20010 loops=1)
"       Sort Key: (count(uf.followee)) DESC, up.level DESC"
    Sort Method: quicksort Memory: 2195kB
    ->  GroupAggregate  (cost=7129.57..7147.30 rows=263 width=18) (actual
time=675.143..759.460 rows=20287 loops=1)
"             Group Key: uf.follower, up.level"
          Filter: (count(uf.followee) > 1)
          Rows Removed by Filter: 2187
          ->  Sort  (cost=7129.57..7131.54 rows=788 width=18) (actual
time=675.129..702.669 rows=435252 loops=1)
"                Sort Key: uf.follower, up.level DESC"
              Sort Method: quicksort Memory: 32691kB
              ->  Hash Join  (cost=1417.54..7091.66 rows=788 width=18) (actual
time=22.090..561.252 rows=435252 loops=1)
                    Hash Cond: (uf.follower = up.mid)
                    ->  Append  (cost=157.22..5829.27 rows=788 width=16) (actual
time=13.348..450.858 rows=435252 loops=1)
                          ->  Bitmap Heap Scan on userfollow_1 uf_1  (cost=157.22..1441.97
rows=195 width=16) (actual time=13.348..99.300 rows=108211 loops=1)
                                Recheck Cond: (followee = ANY ($0))
                                Filter: ((follower <> 362870) AND (NOT (hashed SubPlan 2)))
                                Rows Removed by Filter: 8804
                                Heap Blocks: exact=7964
                                ->  Bitmap Index Scan on userfollow_1_followee_idx
(cost=0.00..47.20 rows=390 width=0) (actual time=11.879..11.879 rows=117015
loops=1)
                                      Index Cond: (followee = ANY ($0))
                                SubPlan 2
                                  ->  Index Only Scan using userfollow_2_pkey on userfollow_2
userfollow_1  (cost=0.43..102.27 rows=3077 width=8) (actual time=0.015..0.446
rows=3000 loops=4)
                                        Index Cond: (follower = 362870)
```

```
                                Heap Fetches: 0
                            此处省略其余3个分区表的记录
                        ->  Hash  (cost=786.81..786.81 rows=37881 width=10) (actual
time=8.715..8.717 rows=37881 loops=1)
                                Buckets: 65536 Batches: 1 Memory Usage: 2288kB
                                ->  Seq Scan on userprofile up  (cost=0.00..786.81 rows=37881
width=10) (actual time=0.024..4.997 rows=37881 loops=1)
Planning Time: 0.750 ms
Execution Time: 772.373 ms
```

4、优化总结

1. 用户安全性高
   - 使用 `Argon2` 编码用户口令，不在数据库中以明文存储，且有较高可扩展性，能简单通过调整参数提高口令安全性。
2. 多线程导入
   - 使用异步机制结合 `COPY` 命令完成数据的多线程导入，仅计算导入过程性能提升 30%。
3. 各业务查询优化
   - 最慢业务是相关视频推荐，最大数据平均运行时间 2 秒，冷查询时间 2.7 秒，热查询运行时间 2.3 秒，缓存内运行时间 1.6 秒。该业务可以在视频播放时同步计算，因此用户的体感等待时间（指无事可做等待系统刷新）可忽略。
   - 次慢业务是用户关注推荐，最大数据平均运行时间 0.7 秒，冷查询时间 0.8 秒，缓存内运行时间 0.6 秒。该业务运行时间与用户加载正常大小图片时间接近，略有等待感。
   - 其余业务运行时间均在 0.1 秒以内，在实际业务中不会出现由数据库服务导致的卡顿。
4. 运行资源占用低
   - 数据库内存需求最大的服务是用户关注推荐，语句分析结果显示最高占用 $32MB$ 左右，`work_mem` 设为 $256MB$ 可满足内存中排序的需求。
   - JVM的性能分析结果显示内存使用峰值为数据导入，提交 $1.3GB$，使用 $1.1GB$ 左右。
   - 整体CPU占用峰值出现在口令编码和多线程导入，数据库最多启动 6 个会话，峰值占用时间总计为 12 秒左右。其余时间大部分只用单线程运行，在部分涉及分区表的业务数据库系统会暂时使用多线程加速，峰值占用时间不超过 1 秒。
5. 应用架构清晰
   - 与数据库连接的有关服务均在 `DatabseService` 类中，限制应用访问数据库的范围。
   - 异步方法在独立类中实现，提高可维护性和可调试性。
   - 对可能修改数据库的语句，均使用事务包裹，保证数据一致性和完整性。
   - 所有服务均有完整的输入校验和日志记录。