

抖音搜索核心链路基于Kitex流式的技术改造

演讲人: 马永真



# 日录 | Contents

Part 01 性能优化

抖音搜索的业务简介 &&RPC流式的必要性 Part 02 流式改造

基于Kitex流式架构的搜 索核心链路改造 Part 03 未来期望

未来架构演进期望

# 01

# 性能优化

抖音搜索的业务简介&&RPC流式的必要性

# 抖音搜索业务简介







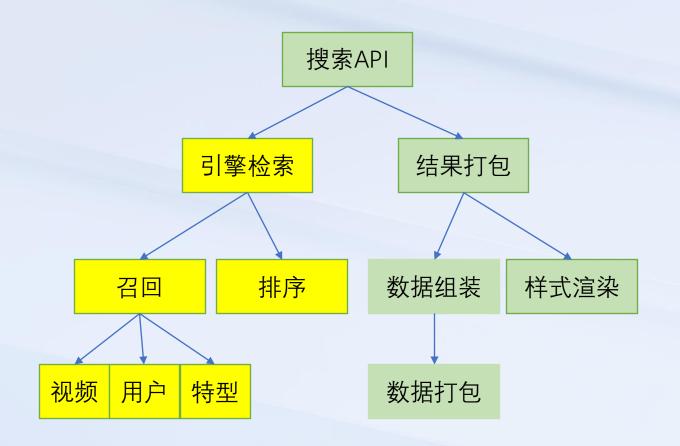
#### 搜索是内容分发的重要技术之一

- 主动发起,目的明确
- 连接人与信息

#### 搜索性能对业务指标有明显影响

- Google曾调研:页面加载速度每增加400ms,流失 0.6%的用户
- 抖音搜索性能劣化实验证明:搜索性能和有点比、搜索PV、综搜30LT等搜索核心指标都有强相关性

# 抖音搜索架构



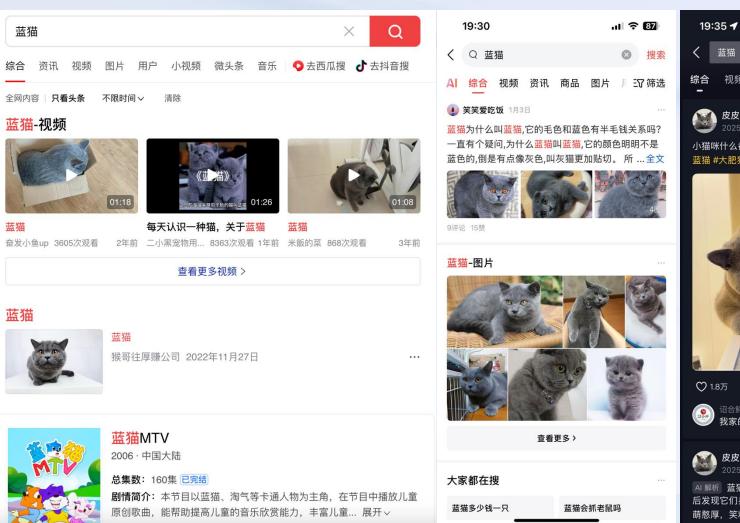
#### 搜索业务性能优化难度高

• 相比于推荐:搜索很难预取,可被取消

• 分阶段优化瓶颈:引擎耗时偏高

• 传统优化手段成本较高或效果不好,如预取,性能cache

### 抖音搜索业务特点



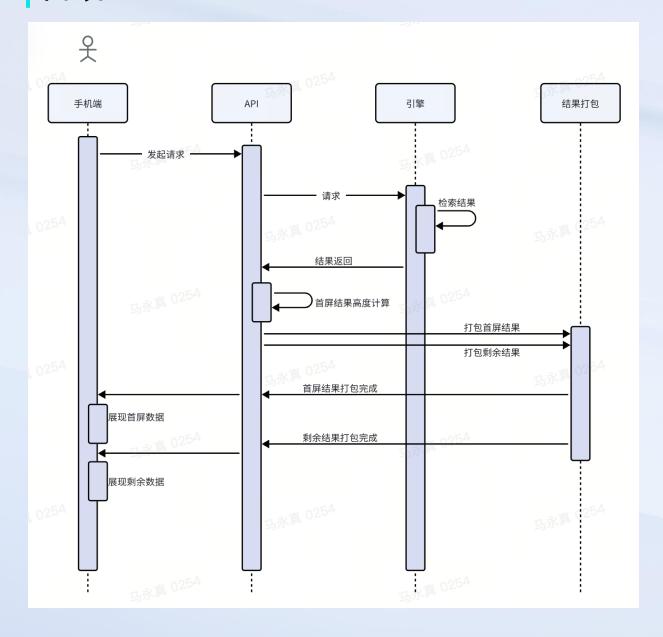


#### 抖音搜索特点

- 大部分是视频结果,卡片纵向高度较高
- 铺满首屏所用的搜索结果更少,一般1-2 个结果即可铺满

头条PC 头条APP 抖音APP

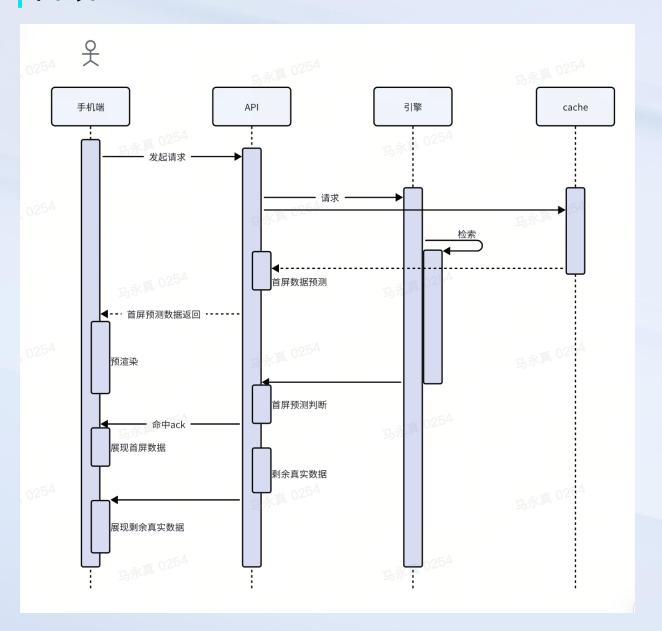
# 首屏结果拆分



#### 首屏拆分

- 分段返回数据,优先返回用户能看到的首屏内容,其次返回剩余数据
- 首屏数据为一般1-2条结果

## 首屏预测思想



#### 首屏预测核心思路

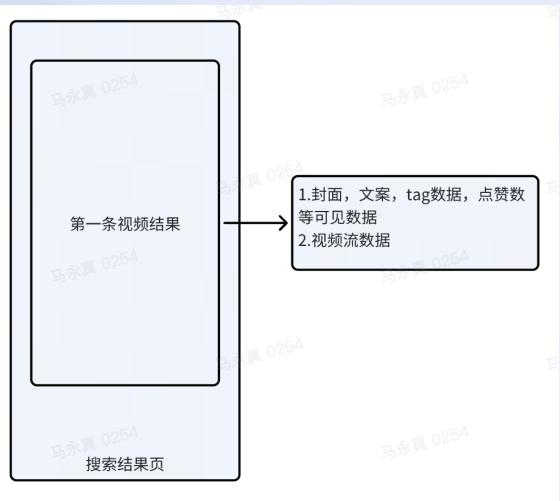
- 分段返回数据,优先预测返回用户能看到的首屏内容
- 返回三次数据,预测数据,ACK数据,剩余数据
- 预测首屏数据命中时,剩余数据为排除掉首屏的内容;未命中时,剩余数据为全部数据内容(包含预测失败的真实首屏数据)

#### HTTP流式架构的必然性

- 实现的两种思路,两次请求方案 or HTTP流式返回
- 两次请求导致请求量翻倍,预测请求和真实请求不落在同一个实例, 最终选择HTTP流式chunk返回是更好的方案

## 其他优化手段-自建打包





#### 自建打包方案

- 首先展现视频静态数据,如封面,文案等能够快速打 包返回
- 随后返回打包较慢的视频流数据

#### 自建打包架构弊端

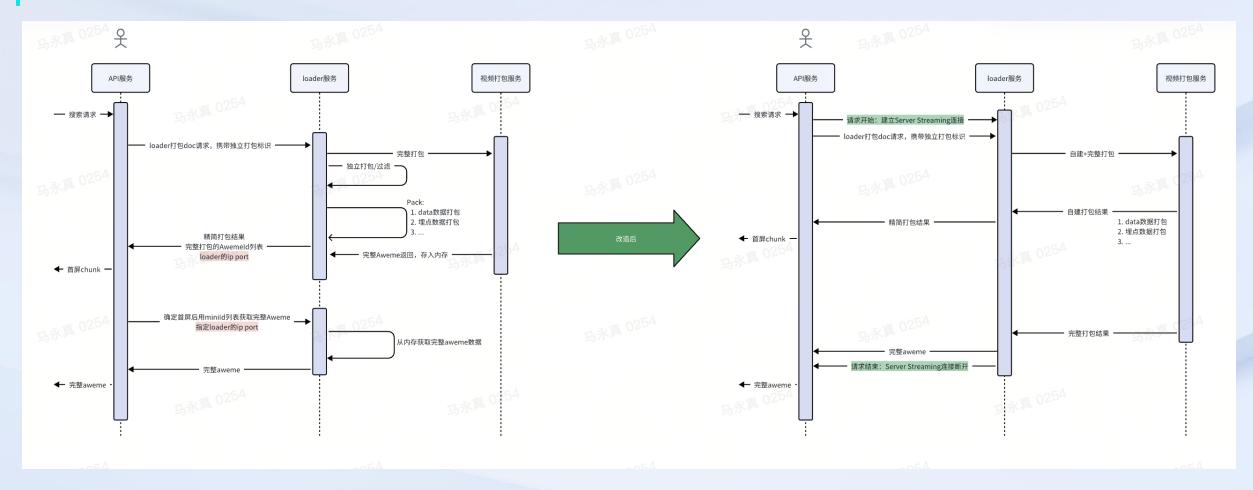
- 由于当时kitex框架对流式打包不支持,只能选择两次打包结果的方案
- 最早的实现方式为通过一次RPC触发两次打包请求, 将视频流数据缓存在实例内存的方案;第二次打包获 取数据通过写死第一次下游实例IP的方式来解决请求 不落在同一实例的问题。
- 为解决无RPC流式的问题而带来的架构复杂度,带来 了其他稳定性问题。

# 02

# 流式改造

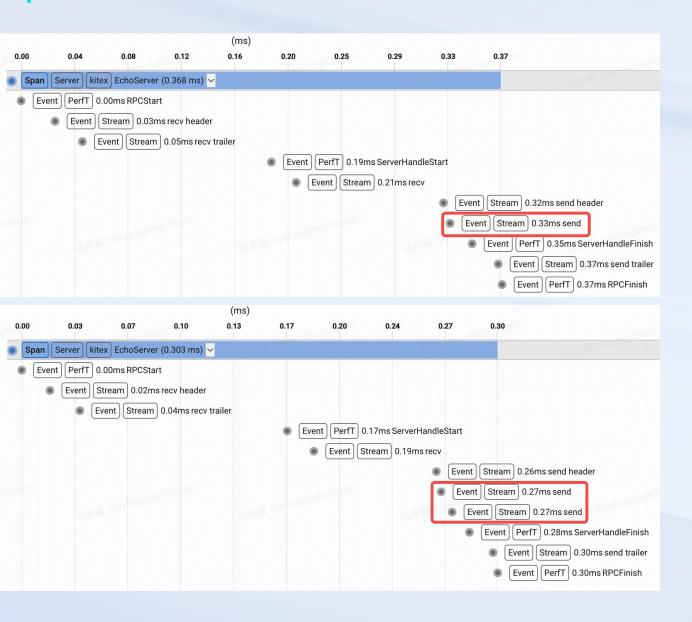
基于Kitex流式架构的搜索核心链路改造

# 打包链路基于kitex流式的架构升级



- 利用流式接口的会话粘性,总体交互流程精简,架构复杂度降低
- 请求量降低,全量数据无需通过二次请求loader内存获取,loader服务常驻内存下降稳定性提升
- 打包成功率提升,改造后极少存在第二次数据无法获取的情况,老方案在服务发布更新,实例销毁时容易出现大量第二次打包数据获取失败的问题

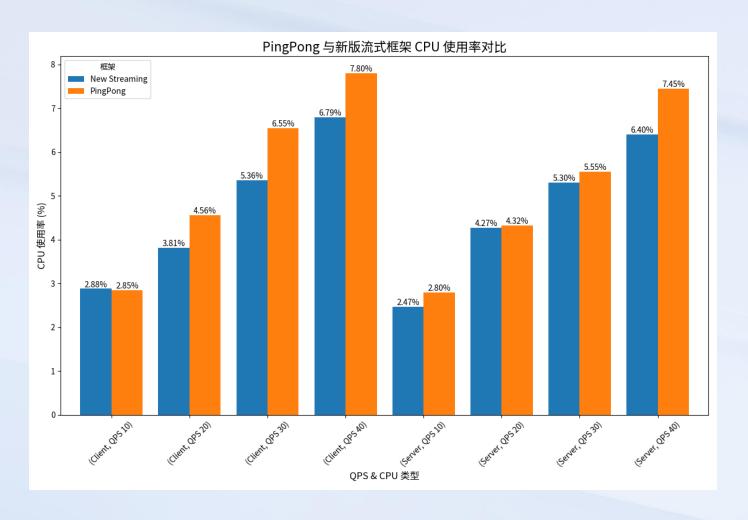
# 迁移流式协议开发体验



#### 可观测性

- · trace包含流式连接全生命周期时序图,可观测性较好
- 流式交互收发包个数+耗时清晰,方便定位问题

### Kitex流式与非流式协议对比



#### 框架本身性能和耗时对比

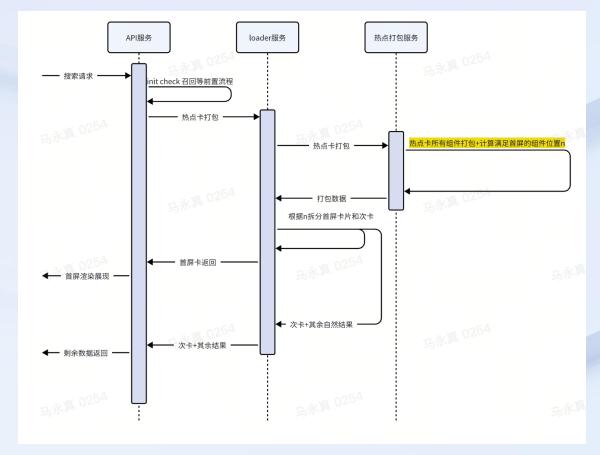
- cpu影响:与非流式时client和server端cpu表现基本一致
- RPC耗时:真实业务场景上进行ab实验,接口耗时上,仅存在不超过 3ms的RPC耗时劣化

#### 流式打包架构下业务收益

- 业务场景中,流式打包在热点卡/活动卡等场景首屏加载速度平均提升 14%以上
- 搜索核心指标中结果有点比,热点卡点击uv等均有显著正向收益

### 更多优化思路--热点卡拆分打包



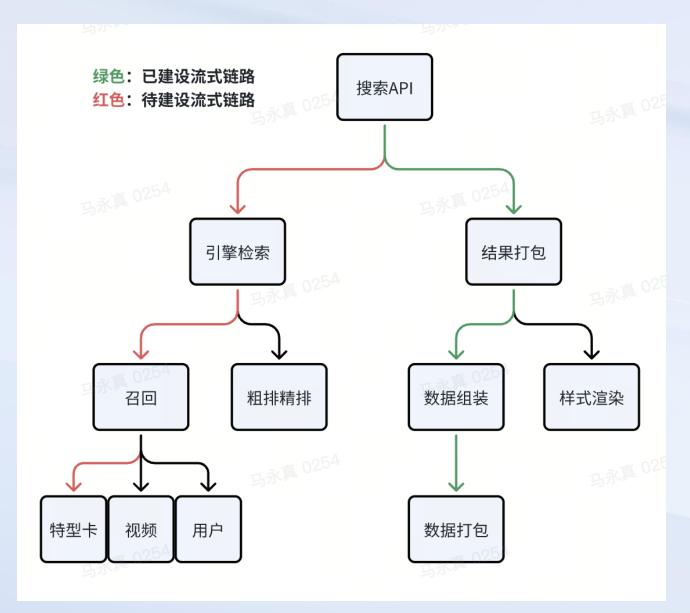


- 热点卡活动卡高度一般都能够铺满2-4屏幕
- 针对背景+视频基本铺满首屏的子卡片进行提前返回,剩余卡片随后返回
- 打包耗时平均优化14%,某些情况下达到50%以上

# 03

# 未来期望

# 理想搜索架构--核心链路流式改造



- 召回链路流式化改造,固排特型卡提前返回,加快搜索结果展现速度
- 个性化策略严重卡片可保持原有链路随后返回

