微视推荐系统性能优化之路

腾讯-在线视频BU-流量生态部-高级工程师
张想

个人简介

硕士毕业
自然语言处理方向

营收平台
智能音箱

微视推荐

2013.7

2018.11

哈尔滨工业大学

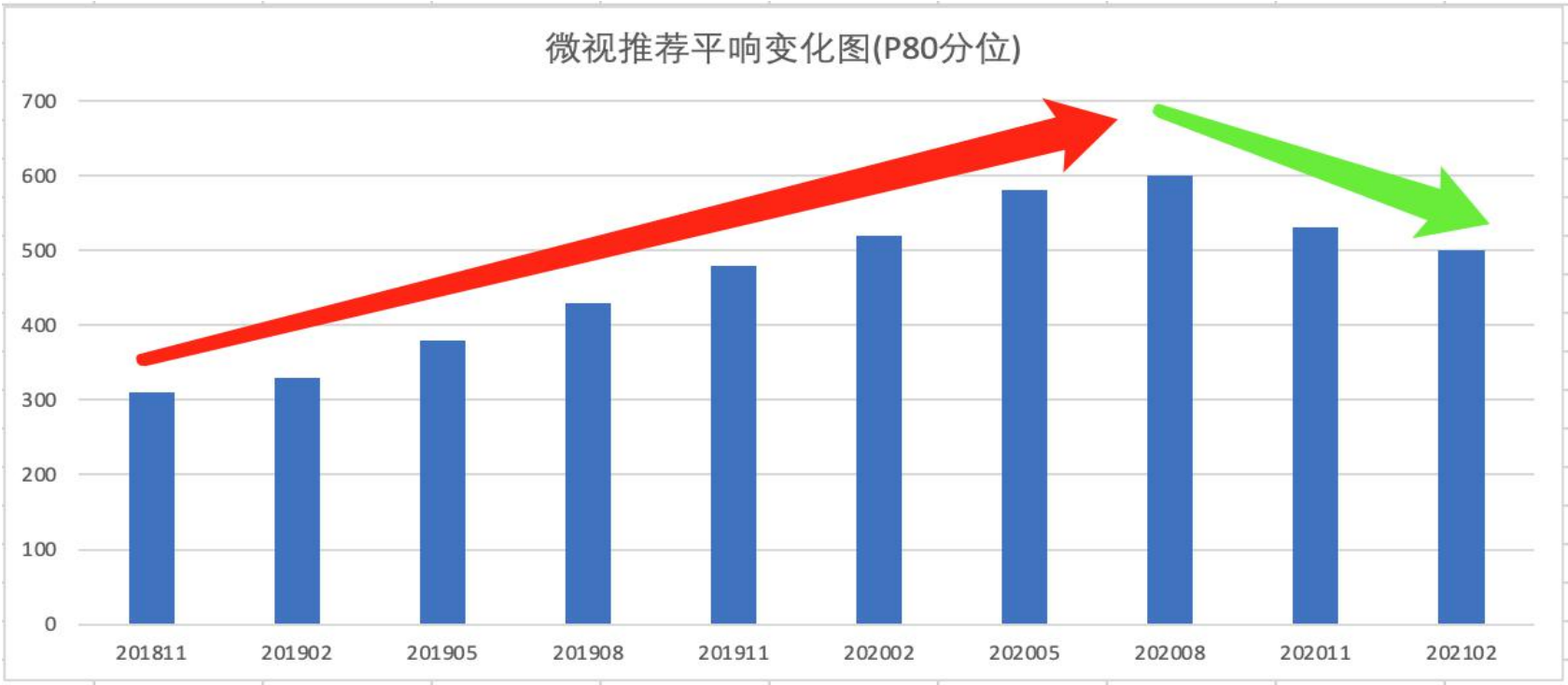
腾讯QQ空间

腾讯微视

见证微视推荐从无到有，目前为微视推荐召回系统负责人，性能与稳定性负责人

目录

背景



推荐业务不断发展，架构规模提升，业务逻辑复杂度升高

- 推荐系统全链路耗时大幅上涨
- 外网用户体验刷feed有迟钝
- 推荐上游团队反馈性能衰退严重

经验表明，链路耗时对产品的指标有影响

Google: +500ms, -20% 流量

Amazon: +100ms, -1% 销售量

优化的收益与困难

- 提升业务指标
- 改善系统稳定性
- 为更复杂策略提供可能

- 系统规模大，访问关系复杂 (线上主要服务30+)
- 迭代变更频繁(研发人员120+，每天发布变更100+)
- 代码质量持续恶化

突破路径

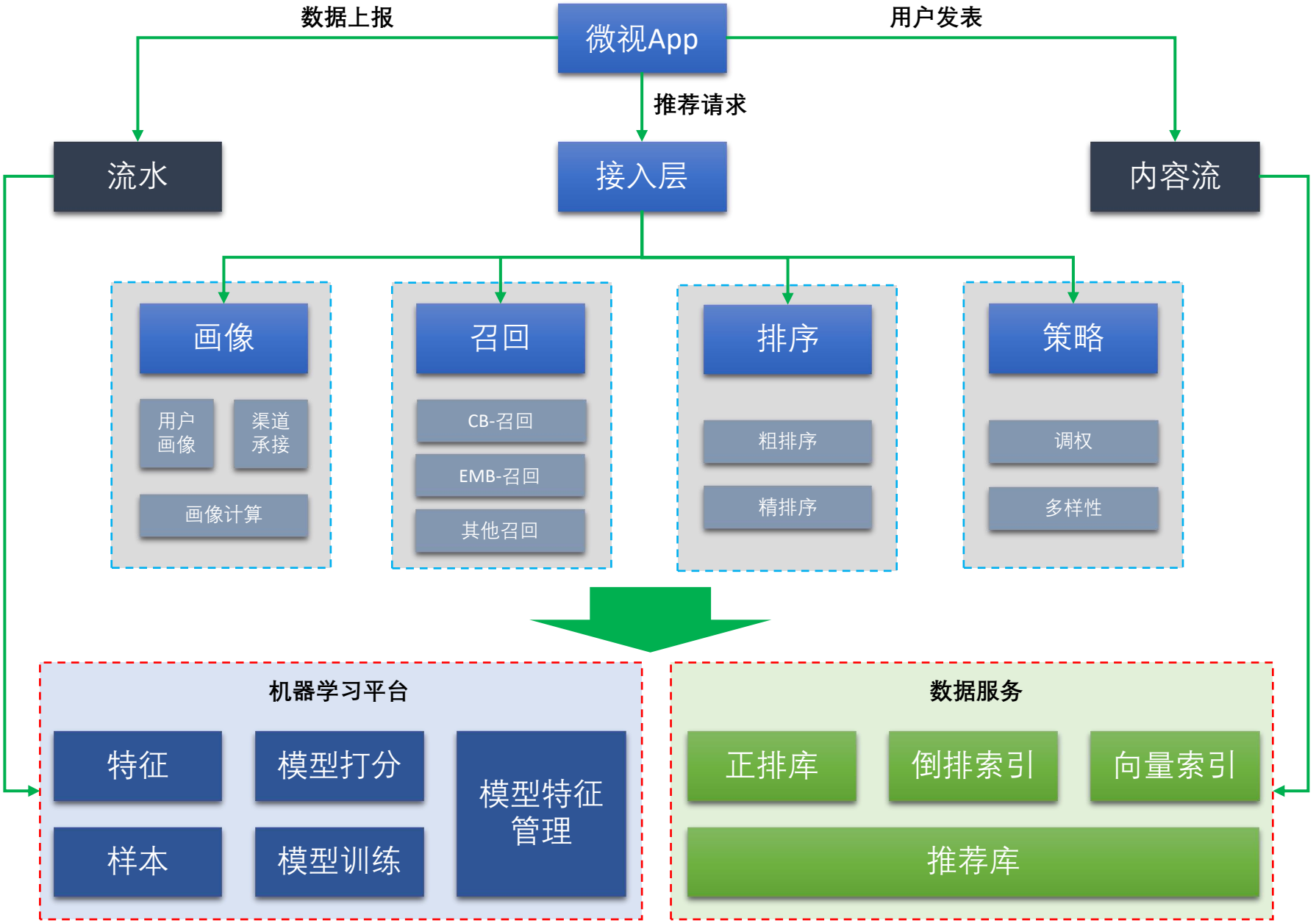
- 从架构/服务/算子多层次看待延时问题并建立立体化监控体系
- 在线服务集成性能分析工具
- 结合业界经验设置合理的耗时优化目标

- 从架构/服务/算子/策略/运维/代码多角度分析系统瓶颈并优化

- 退化治理
- 规范建设

目录

微视推荐架构简介


数据上报
微视App
用户发表
推荐请求
流水
接入层
内容流
画像
用户画像
渠道承接
画像计算
召回
CB-召回
EMB-召回
其他召回
排序
粗排序
精排序
策略
调权
多样性
机器学习平台
特征
模型打分
模型特征管理
样本
模型训练
数据服务
正排库
倒排索引
向量索引
推荐库

目录

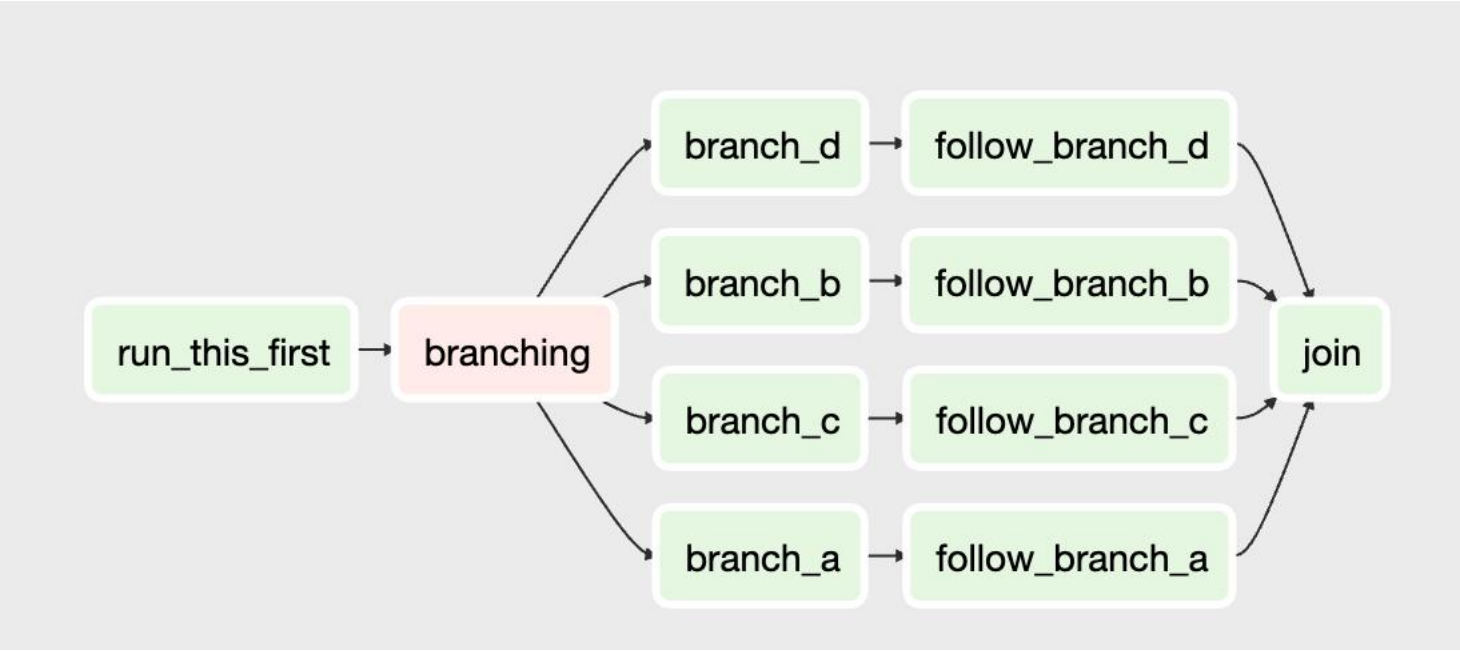
1 推荐类业务研发模式特点

- 不断追求更高的业务指标，需要频繁迭代算法策略，服务变更多
- 算法工程师是策略迭代的主要角色，由于技术栈差异，保障研发质量非常关键

2 对业务框架的要求

- 从研发效率角度，线上逻辑单元可复用，避免重复开发
- 从实验效率角度，需支持方便的abtest实验机制
- 从系统稳定性和耗时角度，需方便定位问题，利于优化

DAG框架简介



推荐工程团队研发了一套DAG框架，具备以下能力

- 算子托管
- 拖拽式配置业务流程
- 实验管理
- 服务变更流程管理

多层次监控埋点

服务耗时监控全覆盖

耗时定位可以精确到服务->算子->阶段

定位系统架构耗时瓶颈

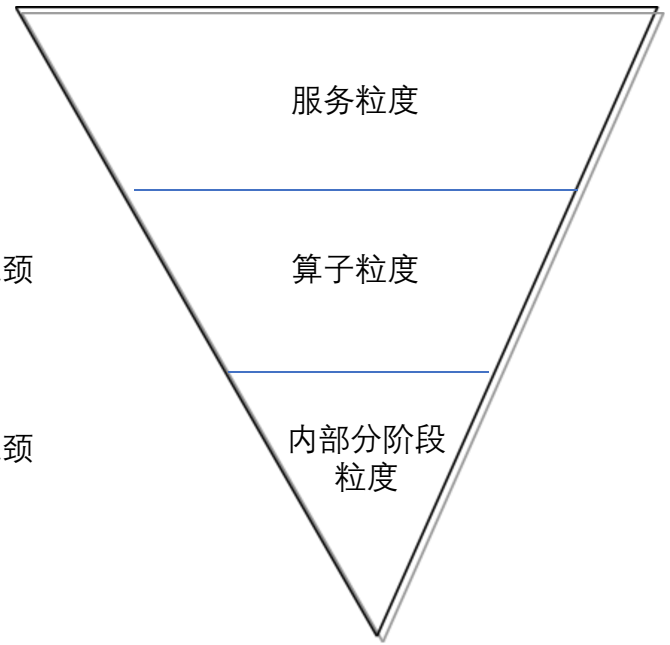
Grafana监控/007服务监控

定位服务内部算子耗时瓶颈

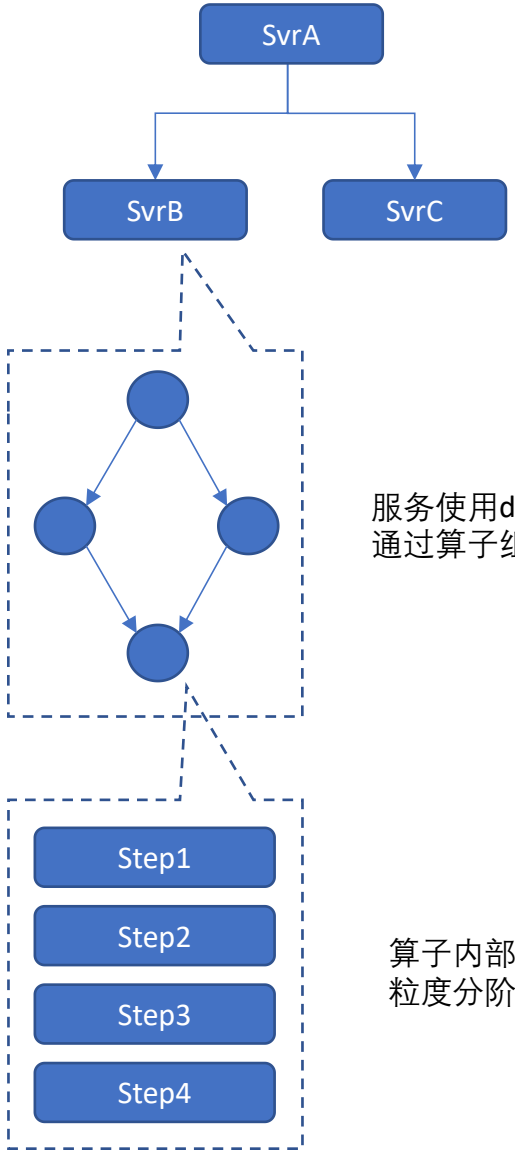
007自定义监控

定位算子内部阶段耗时瓶颈

007自定义监控



Gperftools/perf工具



服务使用dag框架
通过算子组织逻辑

算子内部添加更细
粒度分阶段耗时

服务级别监控


微视推荐系统监控 › WeishiRecallCtrl-监控
上海成功率
上海QPS
广州成功率
广州QPS
上海平均响应时间
上海百分位平均响应时间
广州平均响应时间
广州百分位平均响应时间


服务进程
格式化日志落盘
Logs
匹配日志读取
WatchDog
日志写消息队列
APACHE kafka
A distributed streaming platform
指标聚合计算
Flink
DataSink
Elasticsearch
数据源
Grafana

算子级别监控

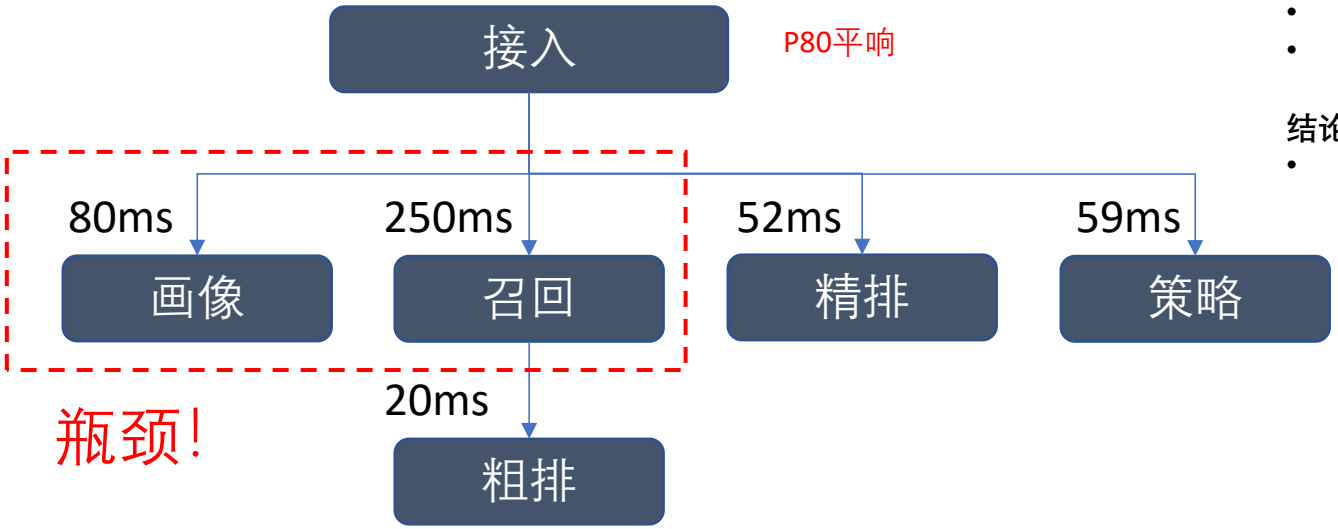
日期	service	ip	set	container	phase	当日count	对比日count	当日success	对比日success	当日failed	对比日failed	当日exception	对比日exception	当日latency	对比日latency
20201219	%	%	client.sh.1	%	VideoRecall					0.00%	0.00%	0.00%	0.00%	266.228	267.275
20201219	%	%	client.sh.1	%	recmd_rank					0.00%	0.00%	0.00%	0.00%	89.811	89.645
20201219	%	%	client.sh.1	%	recmd_recall_multi_layer_cb					0.00%	0.00%	0.01%	0.13%	68.439	69.078
20201219	%	%	client.sh.1	%	recmd_recall_multi_layer					0.00%	0.00%	0.01%	0.07%	63.336	64.649
20201219	%	%	client.sh.1	%	recmd_recall_multi_layer_emb					0.00%	0.00%	0.01%	0.01%	61.143	63.325
20201219	%	%	client.sh.1	%	recmd_recall_multi_layer_direct					0.00%	0.00%	0.03%	0.08%	60.388	61.506
20201219	%	%	client.sh.1	%	simple_basic_rank_boost_v2					1.32%	0.82%	0.00%	0.00%	57.538	57.646
20201219	%	%	client.sh.1	%	recmd_recall_async					0.00%	0.00%	0.01%	0.02%	40.255	40.465
20201219	%	%	client.sh.1	%	recmd_recall					0.00%	0.00%	0.01%	0.02%	38.288	38.467
20201219	%	%	client.sh.1	%	recmd_recall_direct					0.00%	0.00%	0.01%	0.03%	37.875	37.809
20201219	%	%	client.sh.1	%	recmd_recall_other					0.00%	0.00%	0.01%	0.01%	37.783	37.063

算子内分阶段监控

日期	set	container	phase	phase_stage	当日count	对比日 count	当日 latency(ms)	对比日 latency(ms)
20201219	client.sh.1	%	simple_basic_rank_boost_v2	TOTAL			57.863	57.908
20201219	client.sh.1	%	simple_basic_rank_boost_v2	boost_func			2.175	2.041
20201219	client.sh.1	%	simple_basic_rank_boost_v2	cate2_interests			0.034	0.034
20201219	client.sh.1	%	simple_basic_rank_boost_v2	get_params			0.016	0.016
20201219	client.sh.1	%	simple_basic_rank_boost_v2	get_video_detail			8.268	9.376
20201219	client.sh.1	%	simple_basic_rank_boost_v2	simple_predictor			40.328	39.590
20201219	client.sh.1	%	simple_basic_rank_boost_v2	stable_sort			5.307	5.119
20201219	client.sh.1	%	simple_basic_rank_boost_v2	trace			1.779	1.727

目录

瓶颈分析



目标拆解

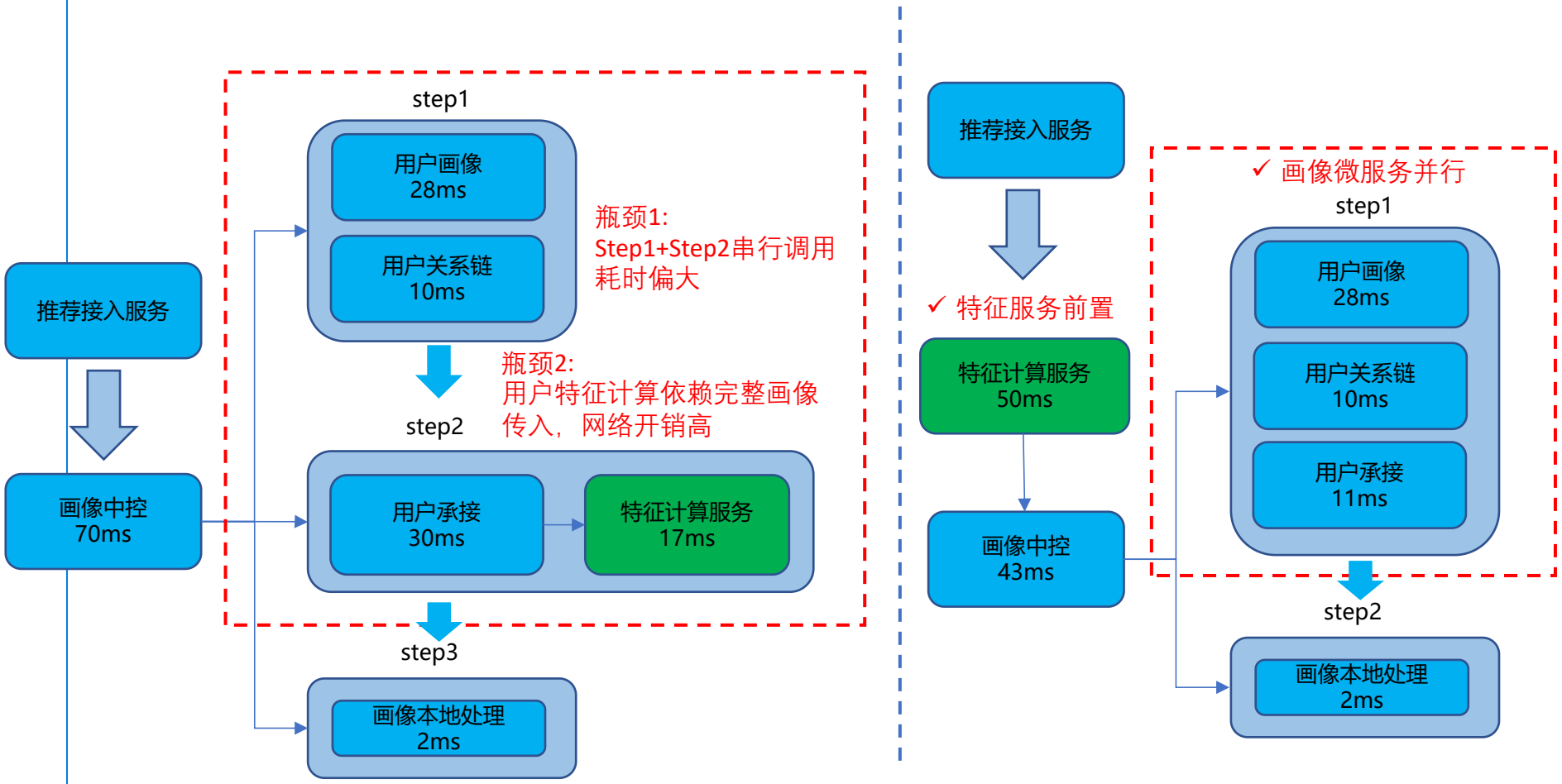
- 针对核心服务设置优化目标
- 优化目标参考业界数据设定

结论

- 召回和画像提升空间较大

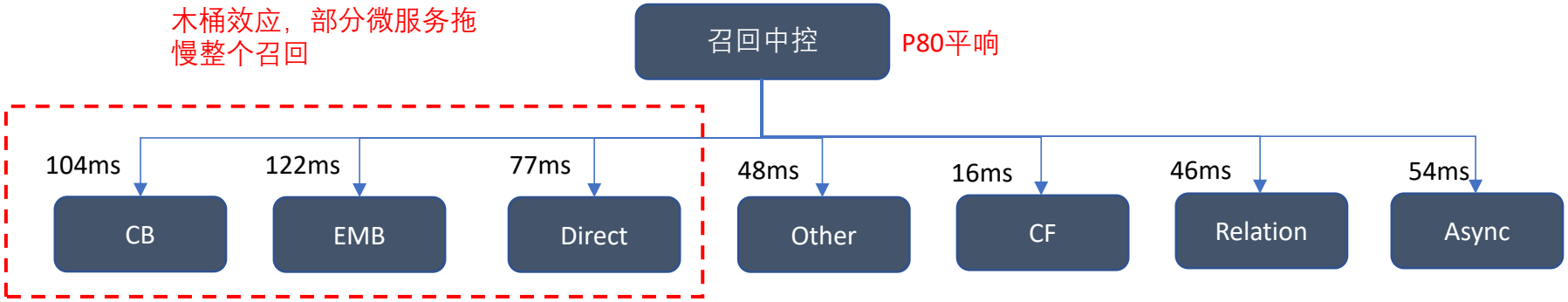
服务	目标耗时(P80)(ms)	上海set优化前(P80)(ms)	广州set优化前(P80)(ms)
接入	400	490	485
画像	55	80	80
召回	160	250	240
粗排	16	20	18
精排	50	52	51
策略	55	59	57

梳理依赖，解除不合理串行，降低画像服务在系统中的层级



✓ 平响优化20ms，P80平响优化25ms

服务优化

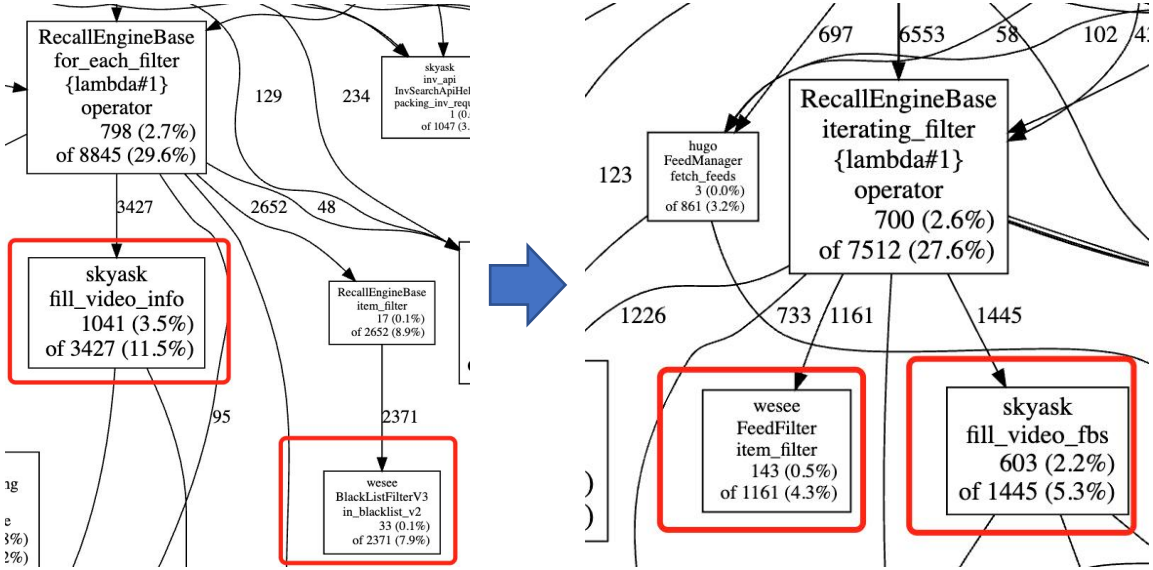


1.改善木桶效应

- 微服务成为瓶颈，CB/EMB/Direct拖慢召回中控
- 数据倾斜问题，CB/EMB召回数量多，队列展现率低，同质化高
- CB/EMB/Direct内部存在耗时高的队列拖慢微服务

2. 消除共性问题

- 热点函数：正排数据填充、队列过滤器
- 正排同步访问：异步化，减少阻塞



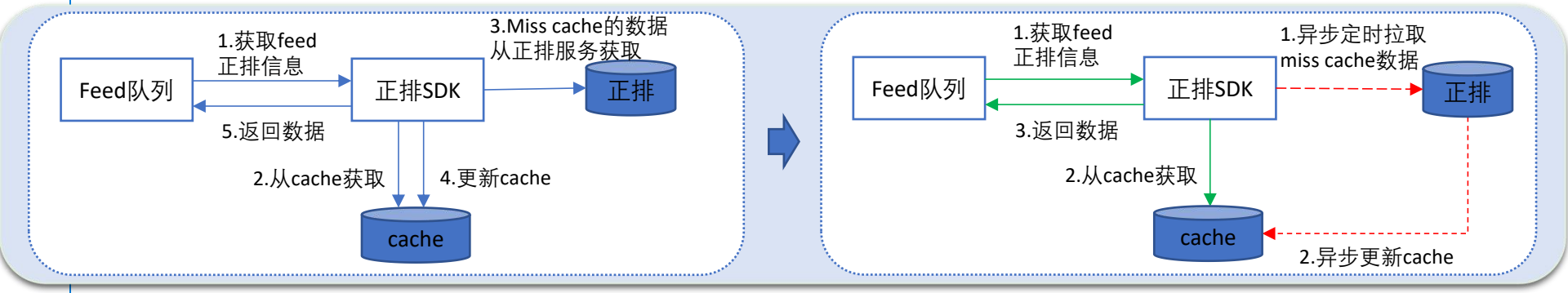
服务优化

共性问题优化汇总

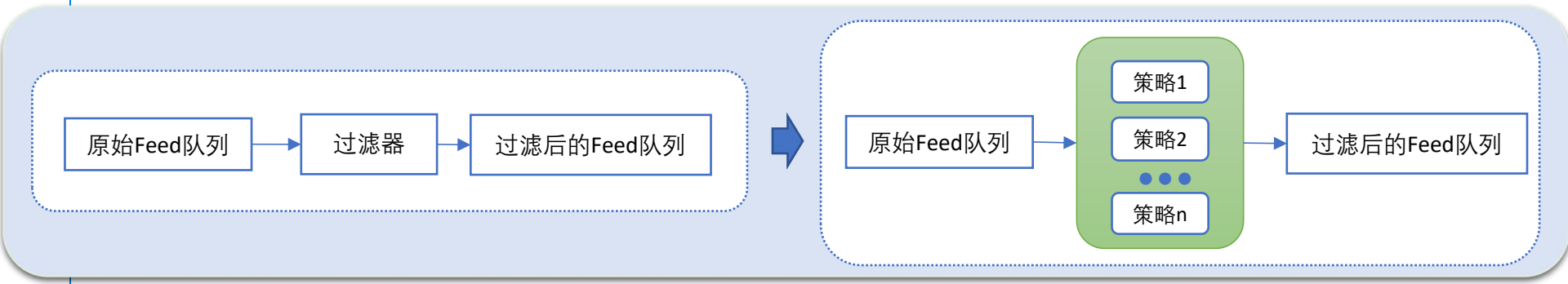
召回中控 P80平响

CB	EMB	Direct	Relation	Async	Other	CF
104ms->58ms	122ms->61ms	77ms->54ms	46ms->34ms	54ms->40ms	48ms->35ms	16ms
-46ms	-61ms	-23ms	-12ms	-14ms	-13ms	

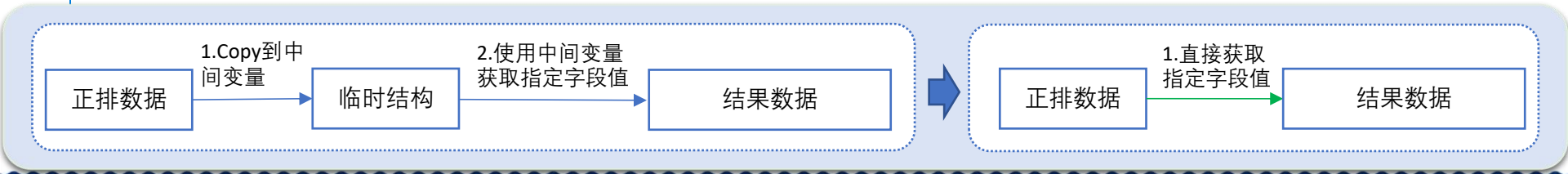
- **正排访问异步化，未命中cache的feed正排信息异步更新回刷，累计优化平响15ms**



- **召回服务负责过滤，我们实现了插件化的过滤器，优化过滤复杂度、过滤下沉至队列提高并行度、过滤算法设置优先级，累计优化19ms**



- **正排数据in-place读取，避免拷贝中间结构，累计优化14ms**



隐式召回服务优化举例

item2item召回场景下的cache化

- 近邻拉链更新不频繁，约10min~30min更新索引
- 重复拉取计算平响高，128维向量，平均10ms的耗时
- i2i类召回存在头部效应，因此cache命中率有一定保证

实验结论：

某个百万级向量索引，当cache条数10000，过期时间为3min时，缓存命中率达50%以上，通路平响下降8ms，后端集群负载减半

隐式召回索引优化举例

✓ 索引精度的评价指标——召回率R@100

对于一个向量x，暴力计算100条近邻拉链L，通过索引计算的100条近邻拉链M，计算L与M的交集大小为n，召回率R@100=n%

为了更准确往往计算一批向量取均值

这里以hnsw索引为例

hnsw索引参数	含义
m	友点数量，影响索引大小
efc	索引构建阶段动态列表长度，影响索引构建速度
efs	索引查询阶段动态列表长度，影响索引查询速度

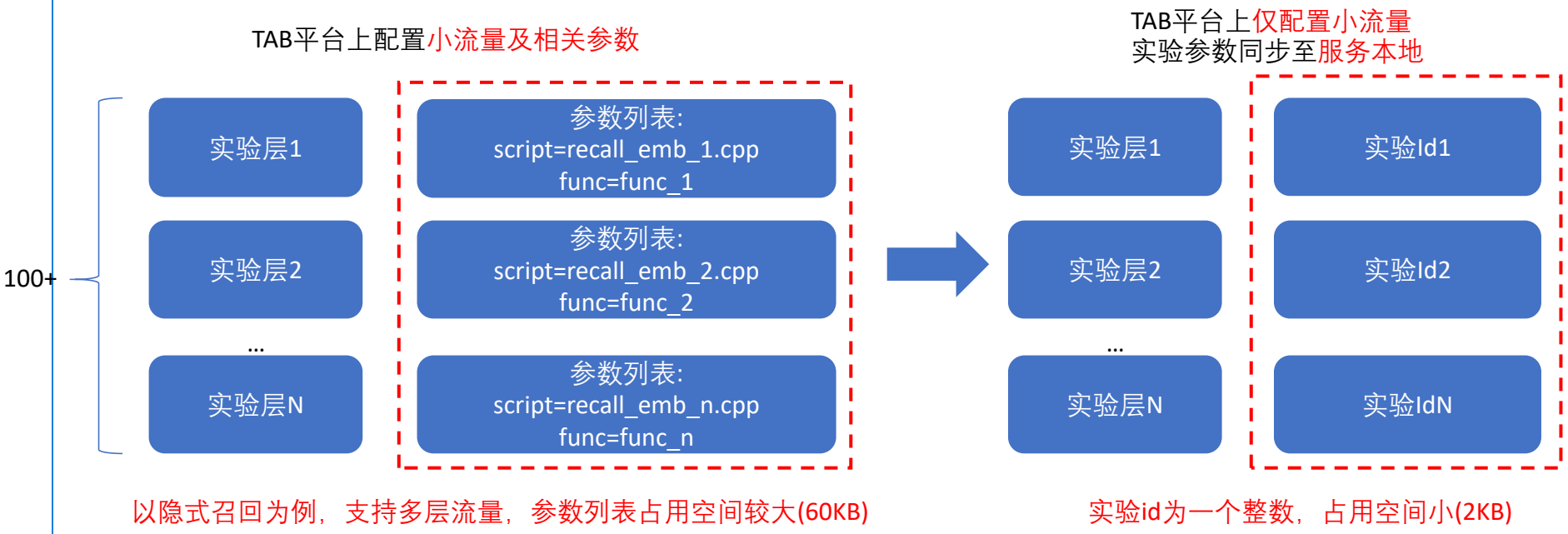
索引参数	索引大小 (100W数据，dim=64)	查询耗时(ms)	R@100,distance=cosine
m=16, efc=200, efs=200	351MB	1.9ms	79.1%
m=16, efc=2000, efs=200	351MB	2ms	85.3%
m=32, efc=2000, efs=200	**483MB**	**2.5ms**	**92.1%**
m=32, efc=2000, efs=2000	483MB	5ms	95.4%

注重索引精度和耗时的平衡，给出合理的索引参数

减包优化

实验框架升级

背景：推荐请求头携带小流量信息全局透传，随着业务复杂度提升
实验层数增多至100多层，占用大小60KB



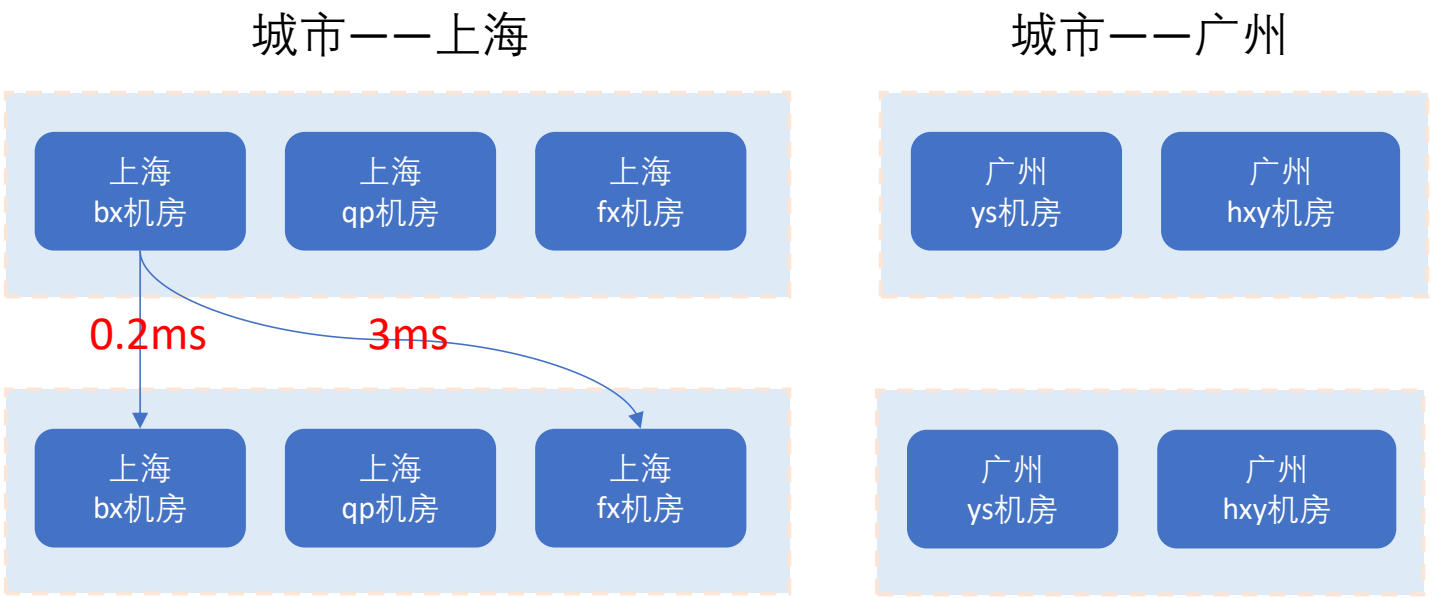
以隐式召回为例，支持多层流量，参数列表占用空间较大(60KB)

实验id为一个整数，占用空间小(2KB)

✓ 减少推荐全链路实验信息透传成本，平响优化约14ms

部署优化

解决跨机房访问，降低跨机房带宽



测试发现：机房内ping延迟仅0.2ms，跨机房ping延迟达3ms，甚至遇到过30ms的情况

方案A:
- 直接部署在一个机房中

问题:
- 单机房资源无法满足
- 不满足容灾要求

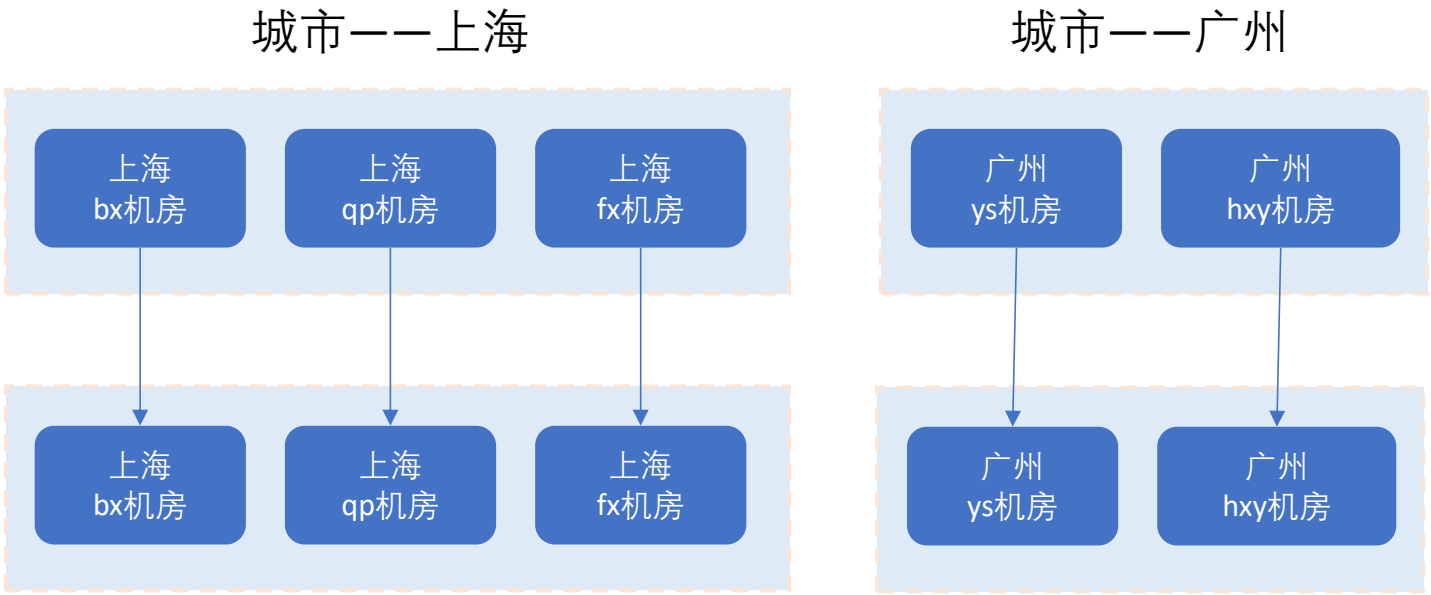
方案B:
- 每个机房都部署全流程，流量只在同一个机房内穿梭

问题:
- 对OP容量规划要求较高，流量划分遵循容量比例，如果失衡可能引发线上问题

比较理想，有实施难度
仍在小规模尝试

部署优化

解决跨机房访问，降低跨机房带宽



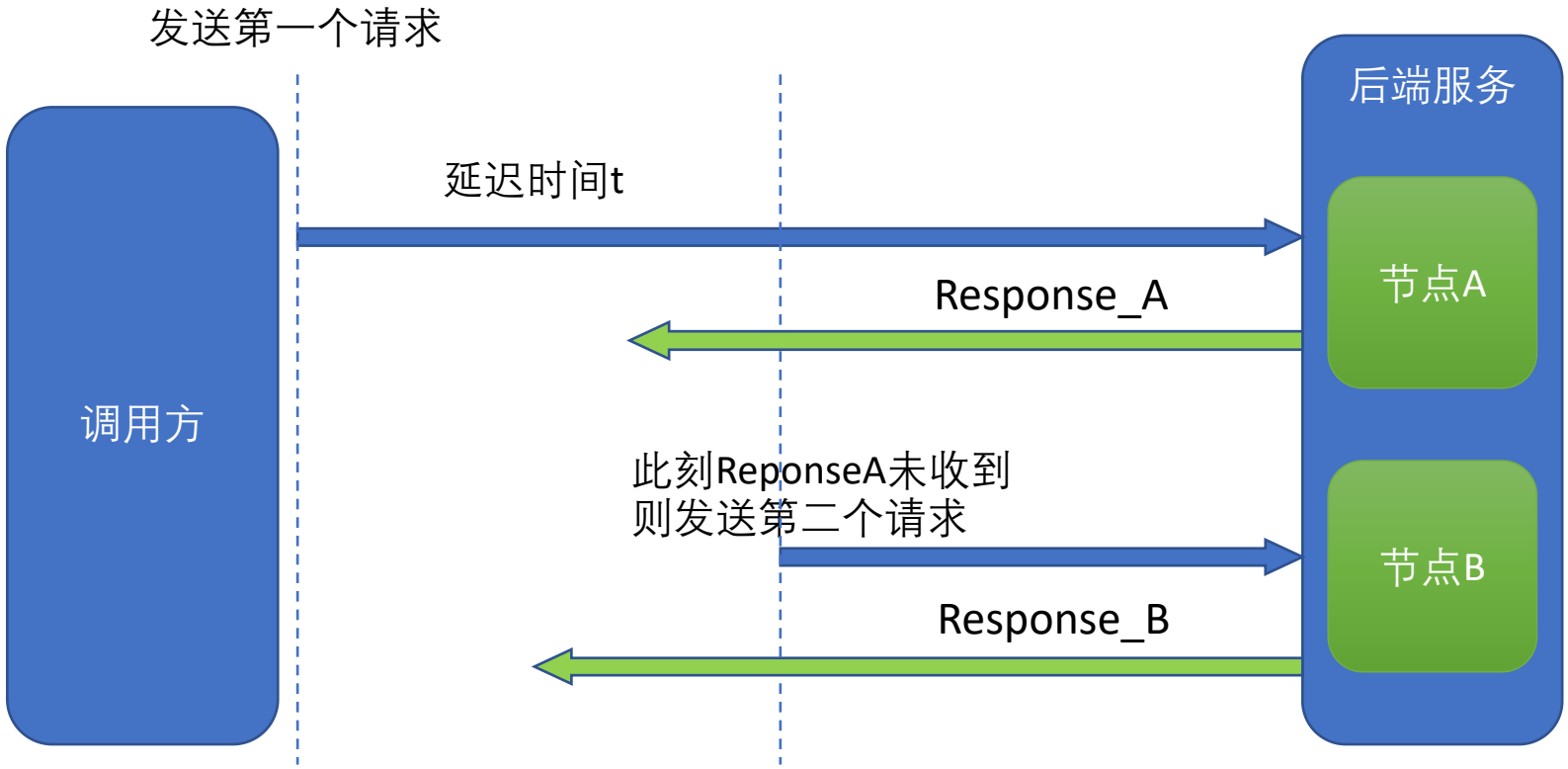
采用了LA负载均衡取代Round-Robin
LA不要求同机房，能够适应下游集群的变化自动调整流量

LA视角下会优先选择耗时低的下游，也就是同机房，然后选择跨机房的下游
也就是调整了下游节点的权重

注意:
处理好业务的错误码，避免异常情况被误认为是低延时

特殊优化——Dynamic Backup Request

常规的backup-request

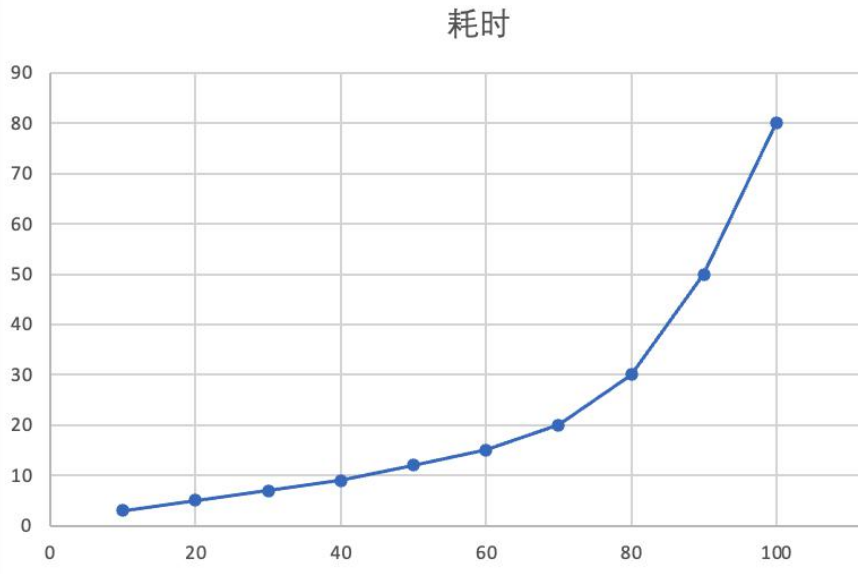


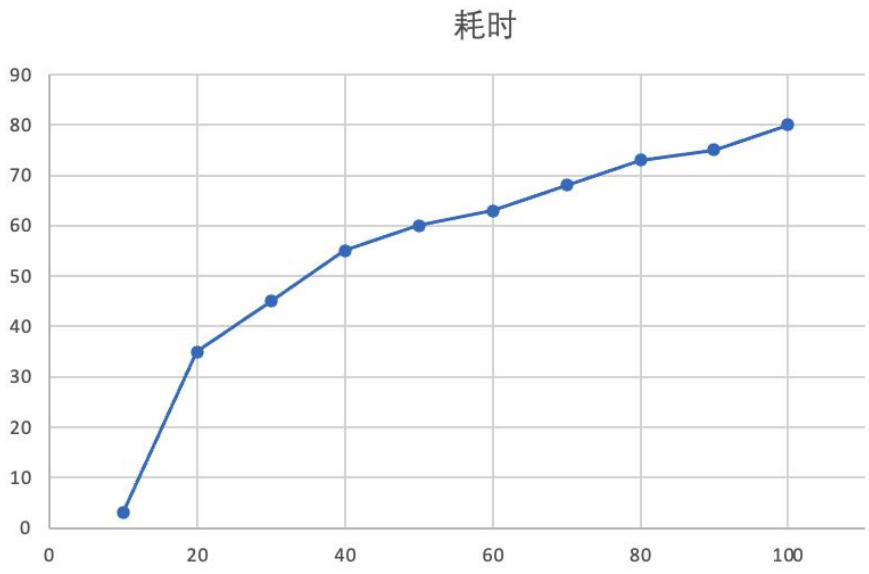
Response_A和Response_B，以先回来的包为准

发送的backup-request可能提前回包，因此可以改善对下游服务访问的耗时

特殊优化——Dynamic Backup Request

服务耗时的cdf曲线





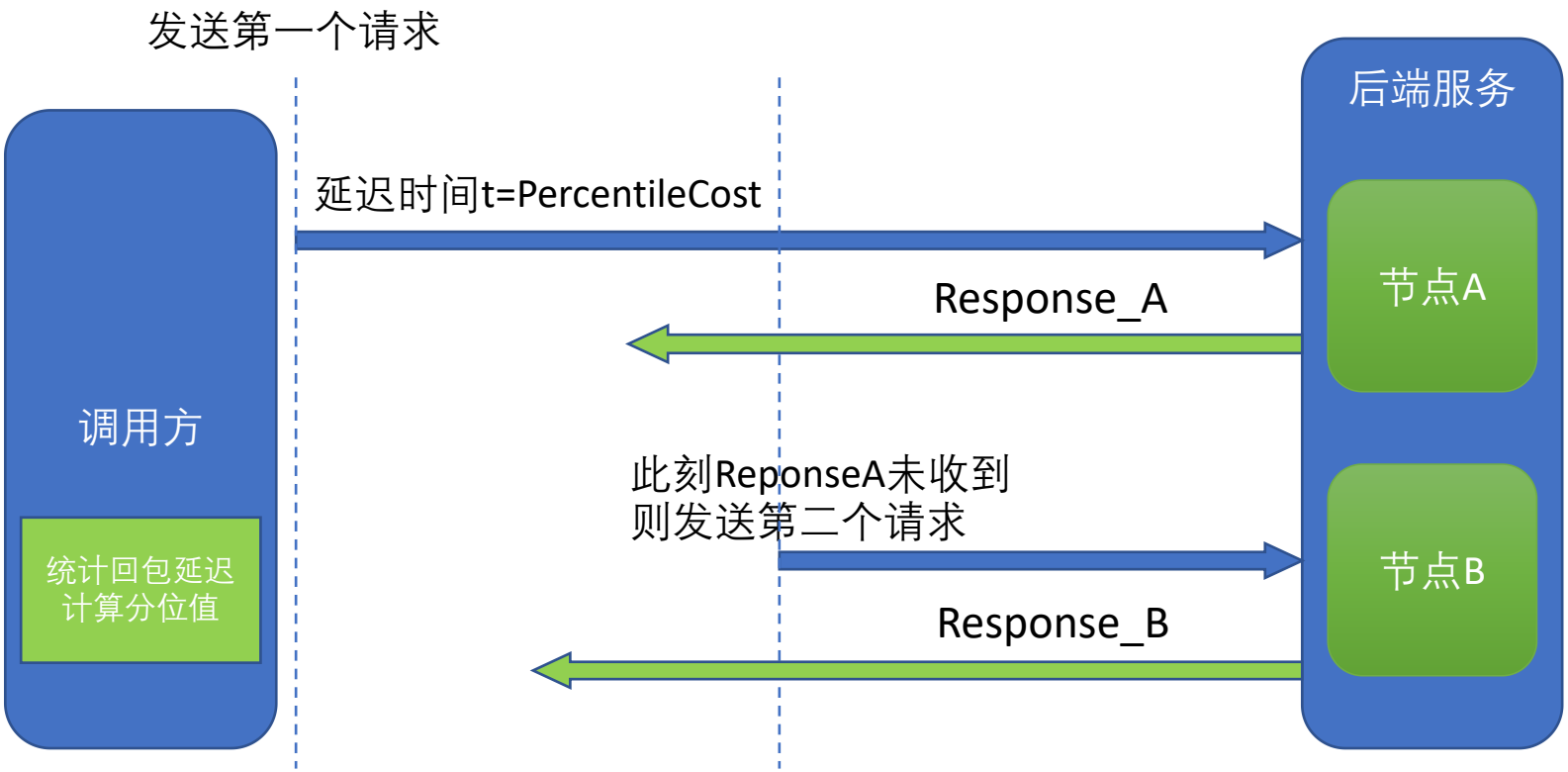
左图中的服务，使用backup-request获得性能提升的潜力要大于右图

当设置backup-request的耗时为40ms时

左图有80%的概率能40ms内回包
右图有23%的概率能40ms内回包

特殊优化——Dynamic Backup Request

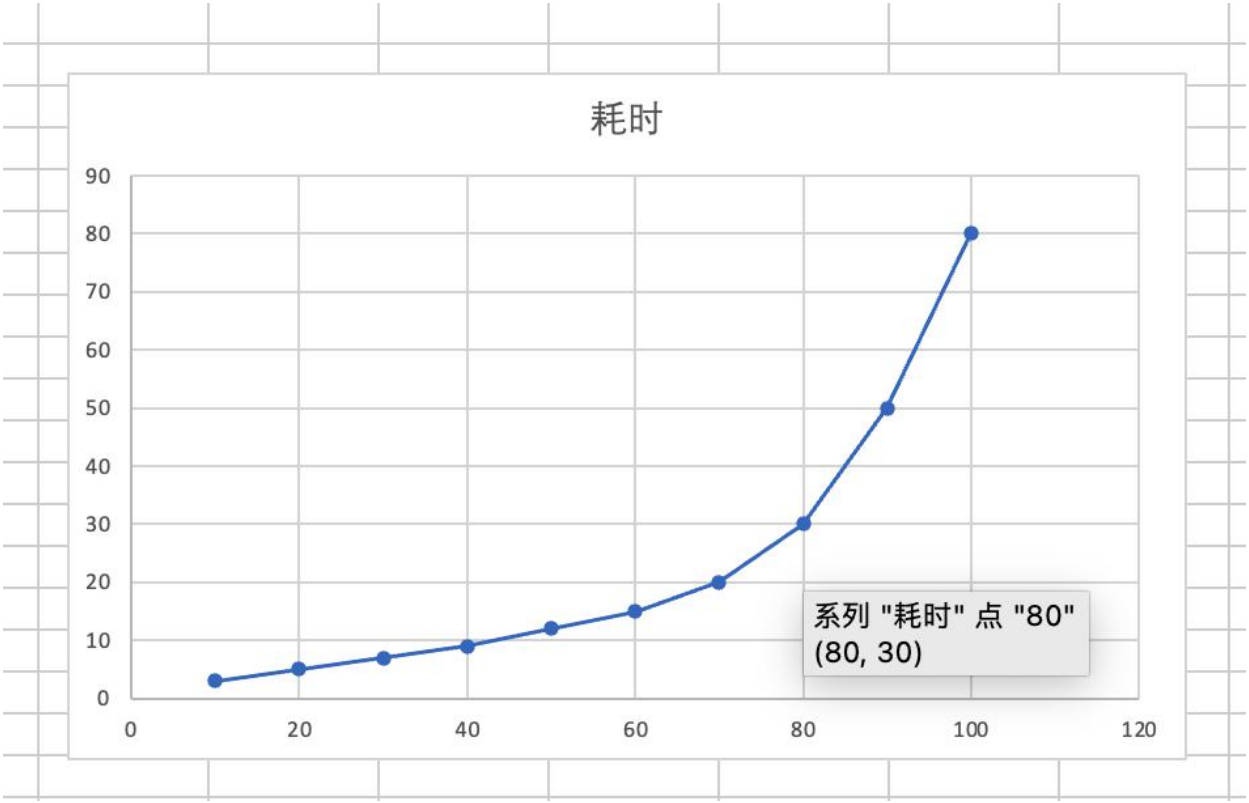
动态的backup-request



普通的backup-request存在问题，无法有效设置一个准确的时延

动态的backup-request的延迟t是动态计算而来，通过定时统计时间窗口内的耗时来设置

特殊优化——Dynamic Backup Request

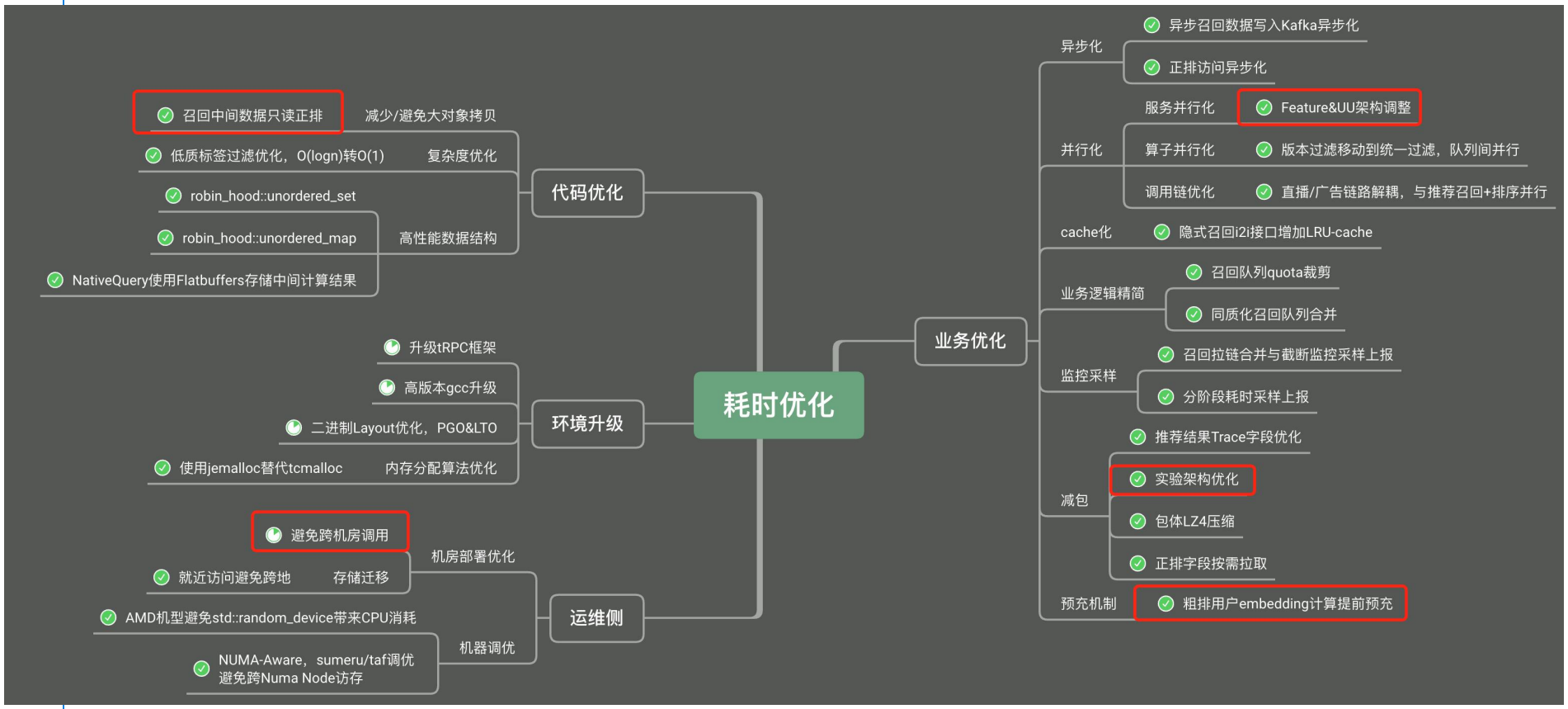


t值由统计
计算最近的时间窗口T内指定分位值作为t，如P80

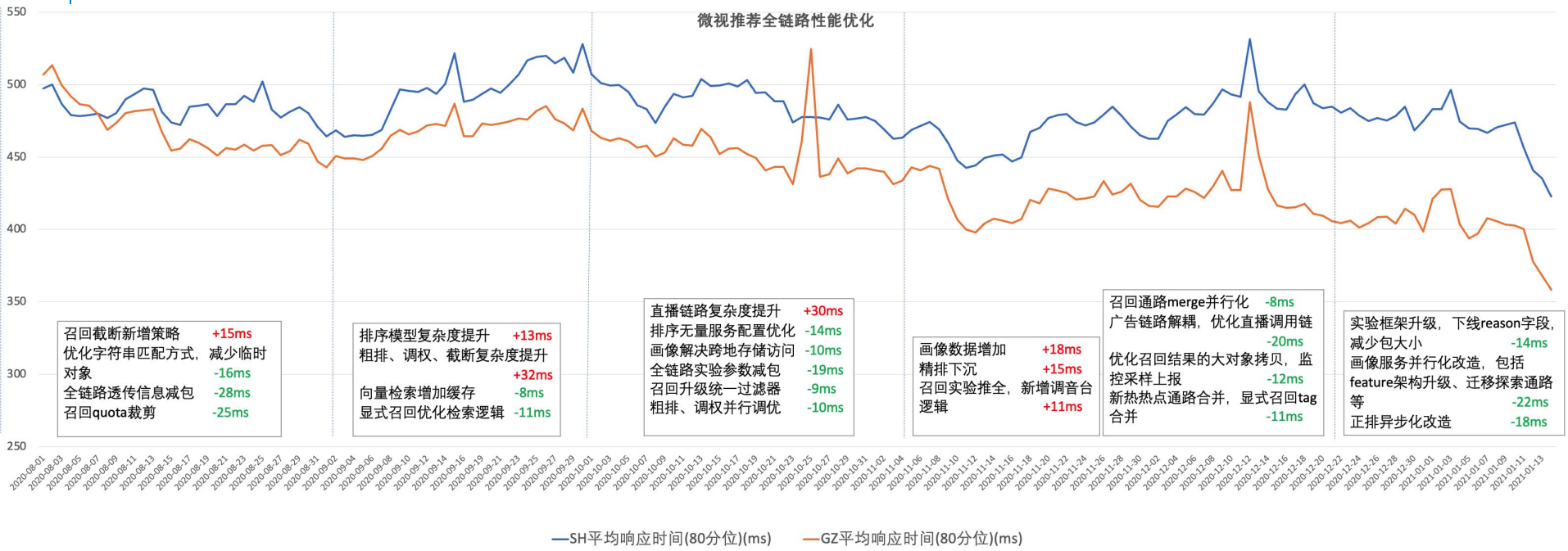
该值的选择需综合考虑latency-cdf曲线和服务负载

✓ 可以有效提高下游服务成功率，降低更高分位耗时

其他方案汇总

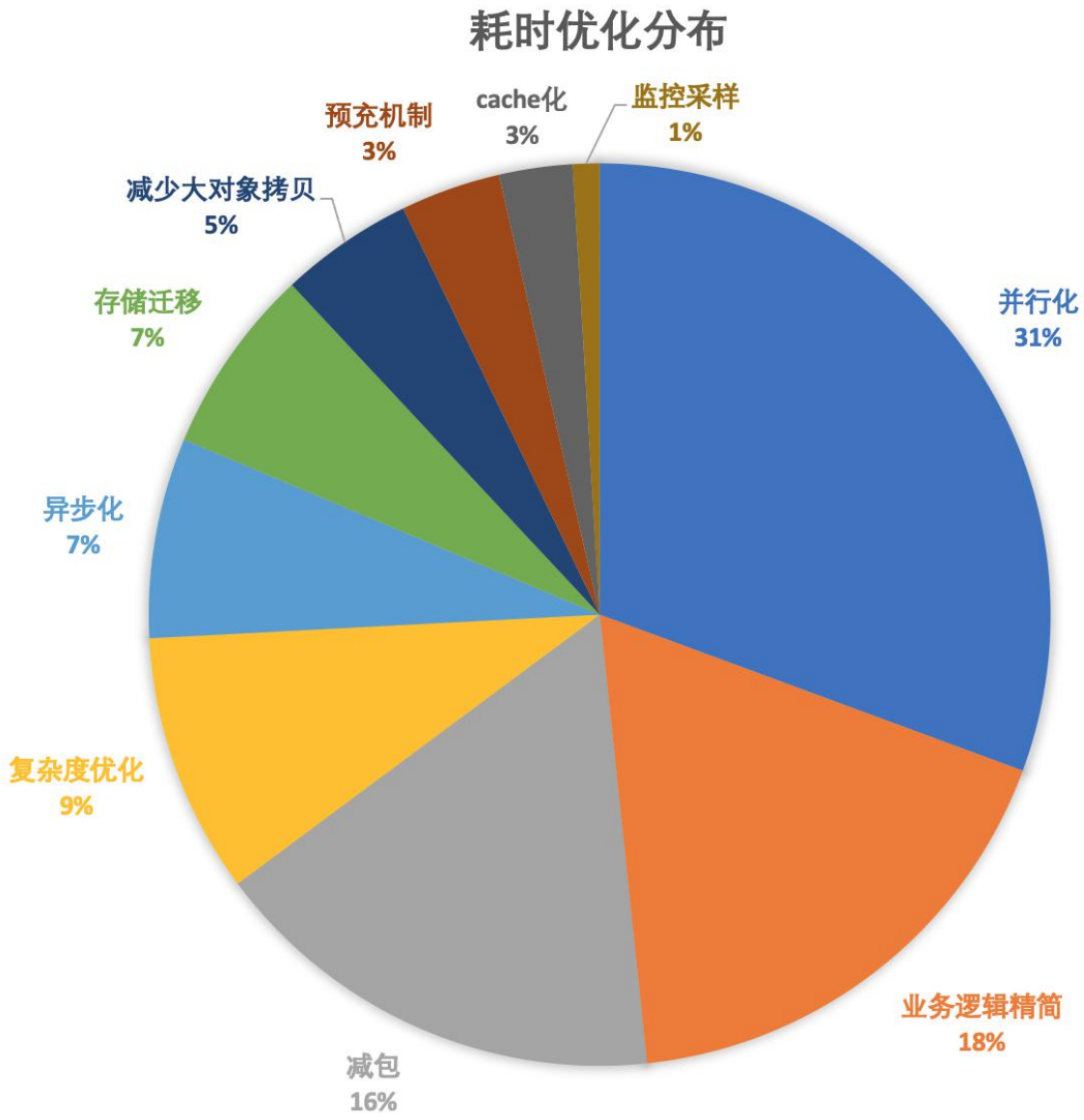

耗时优化
代码优化
召回中间数据只读正排
减少/避免大对象拷贝
低质标签过滤优化，O(logn)转O(1)
复杂度优化
robin_hood::unordered_set
robin_hood::unordered_map
高性能数据结构
NativeQuery使用Flatbuffers存储中间计算结果
环境升级
升级tRPC框架
高版本gcc升级
二进制Layout优化，PGO&LTO
使用jemalloc替代tcmalloc
内存分配算法优化
运维侧
机房部署优化
避免跨机房调用
就近访问避免跨地
存储迁移
AMD机型避免std::random_device带来CPU消耗
机器调优
NUMA-Aware，sumeru/taf调优
避免跨Numa Node访存
业务优化
异步化
异步召回数据写入Kafka异步化
正排访问异步化
并行化
服务并行化
Feature&UU架构调整
算子并行化
版本过滤移动到统一过滤，队列间并行
调用链优化
直播/广告链路解耦，与推荐召回+排序并行
cache化
隐式召回i2i接口增加LRU-cache
业务逻辑精简
召回队列quota裁剪
同质化召回队列合并
监控采样
召回拉链合并与截断监控采样上报
分阶段耗时采样上报
减包
推荐结果Trace字段优化
实验架构优化
包体LZ4压缩
正排字段按需拉取
预充机制
粗排用户embedding计算提前预充

结果一览



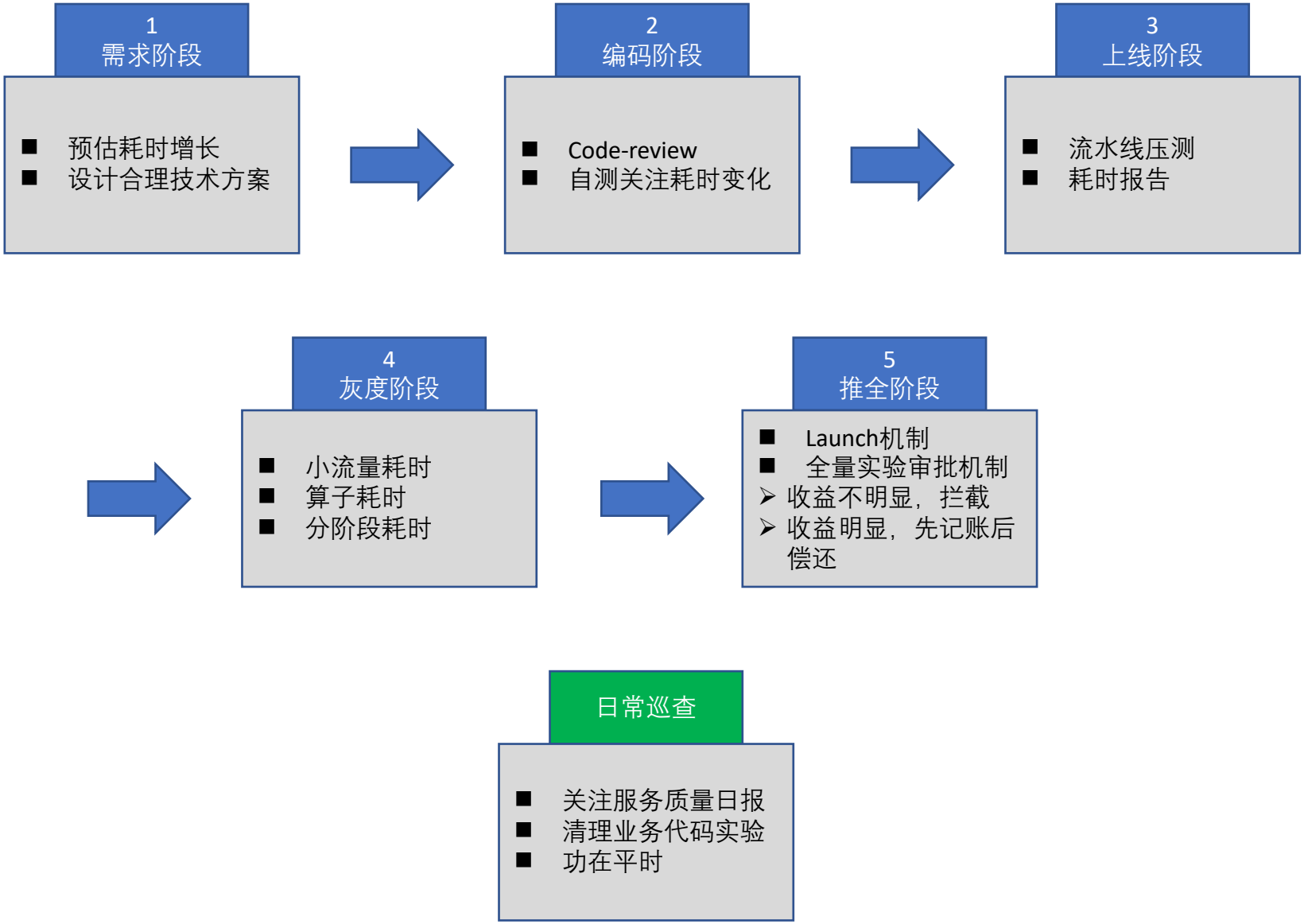
在业务迭代退化累积40项，造成衰退接近100ms的基础上
全链路优化100余项，累积优化75ms，系统稳定性也得到了增强
截至今年1月微视推荐P80耗时
上海415ms(未开启同机房+未开启LA)
广州358ms (开启同机房+开启LA)

优化贡献分布


耗时优化分布
并行化
31%
业务逻辑精简
18%
减包
16%
复杂度优化
9%
异步化
7%
存储迁移
7%
减少大对象拷贝
5%
预充机制
3%
cache化
3%
监控采样
1%

目录

衰退预防


1
需求阶段
预估耗时增长
设计合理技术方案
2
编码阶段
Code-review
自测关注耗时变化
3
上线阶段
流水线压测
耗时报告
4
灰度阶段
小流量耗时
算子耗时
分阶段耗时
5
推全阶段
Launch机制
全量实验审批机制
收益不明显，拦截
收益明显，先记账后偿还
日常巡查
关注服务质量日报
清理业务代码实验
功在平时

目录

总结

- 构建完善的指标体系，保障系统可观测性
- 多层次多粒度的优化
  - 架构上消除不合理调用链
  - 服务上消除短板链路
  - 代码上改善热点函数、计算复杂度、异步化、cache、优化索引
  - 部署上改善跨机房调用
  - 网络调用减少包体大小，降低传输成本
  - LA、Dynamic Backup-Request改善耗时

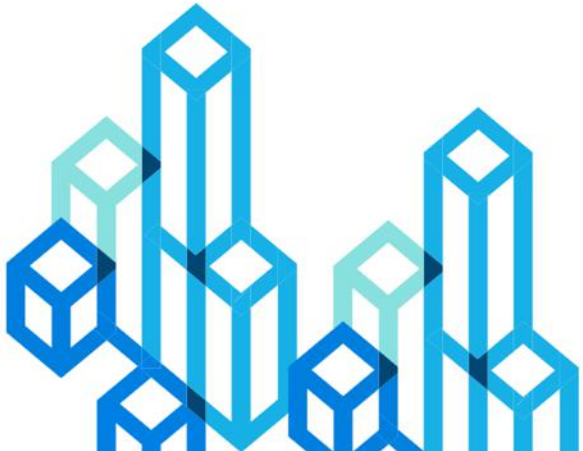
  …
- 衰退避免机制
  - 设定严格的上线规范
  - 实验变更需观测小流量耗时增长情况
  - 定期review清理策略，避免堆积

ZX
广东 深圳

欢迎讨论交流

谢谢

扫一扫上面的二维码图案，加我微信



麦思博(msup)有限公司是一家面向技术型企业的培训咨询机构，携手2000余位中外客座导师，服务于技术团队的能力提升、软件工程效能和产品创新迭代，超过3000余家企业续约学习，是科技领域占有率第1的客座导师品牌，msup以整合全球领先经验实践为己任，为中国产业快速发展提供智库。

高可用架构公众号主要关注互联网架构及高可用、可扩展及高性能领域的知识传播。订阅用户覆盖主流互联网及软件领域系统架构技术从业人员。

高可用架构系列社群是一个社区组织，其精神是“分享+交流”，提倡社区的人人参与，同时从社区获得高质量的内容。