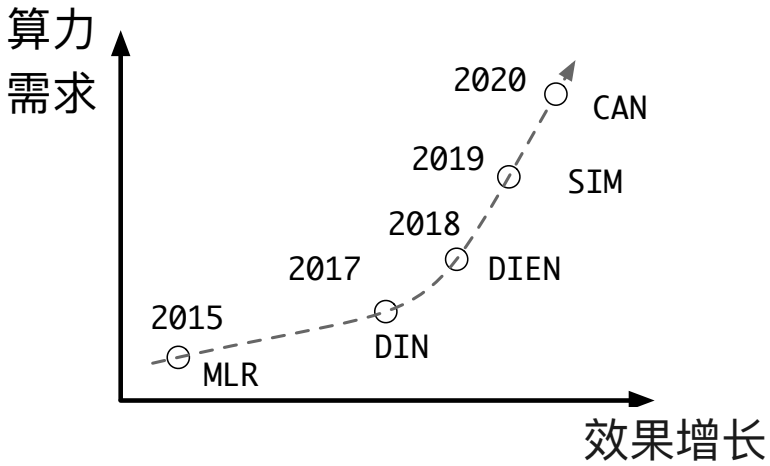
算力效能技术体系@阿里定向广告

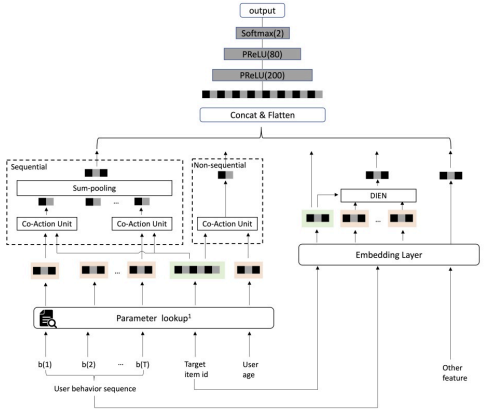
阿里妈妈定向广告算法效能团队
姜碧野

算力效能优化背景

- 深度模型的迭代带来业务提效，但算力需求也在成倍增长
- 算力供给增长放缓，“付费”的算力红利难以为继

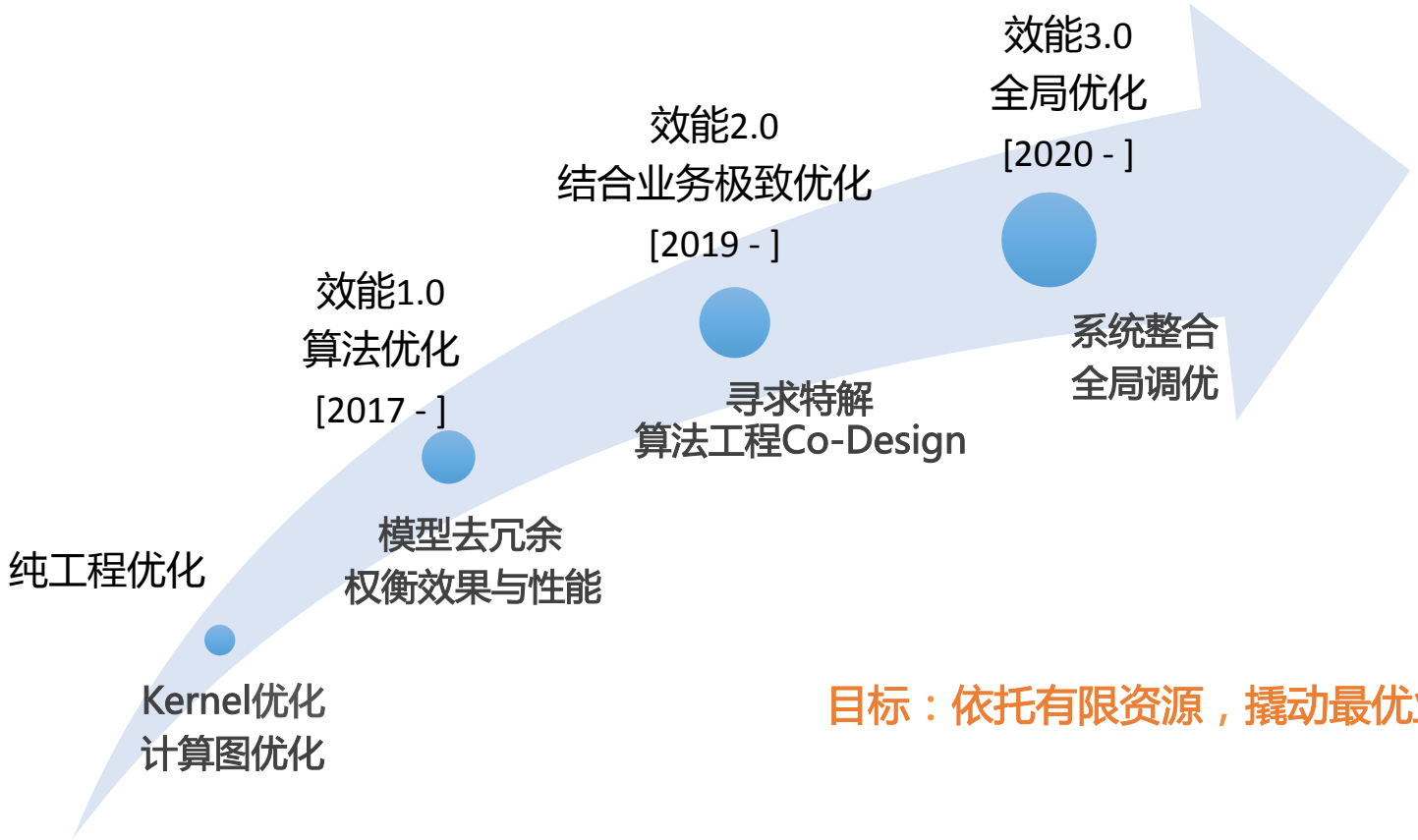


定向广告团队的模型迭代路径
年化算力增长2~3倍



Zhou et al, *Deep Interest Network for click-through rate prediction*
Zhou et al, *Deep Interest Evolution Network for click-through rate prediction*
Pi et al, *Search-based User Interest Modeling with Lifelong Sequential Behavior Data*
Wang et al, *COLD: Towards the Next Generation of Pre-Ranking System*
Zhou et al, *CAN: Revisiting Feature Co-Action for Click-Through Rate Prediction*

算力效能优化技术发展路径



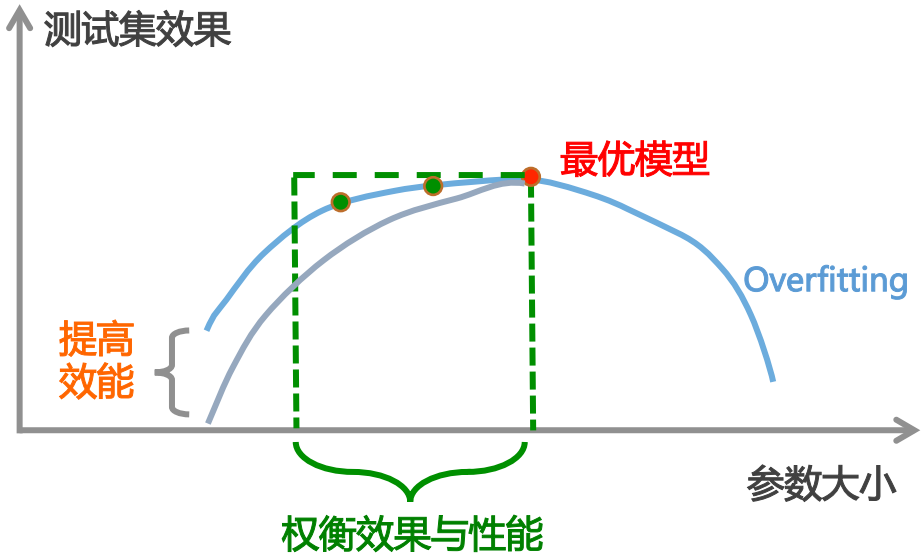
目标：依托有限资源，撬动最优业务收益

Alibaba Group
阿里巴巴集团

效能1.0：算法优化

从工程优化到算法优化

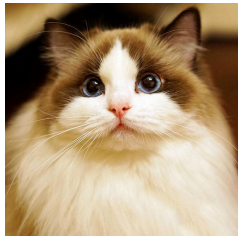
- 工程优化侧重优化模型的执行过程，算法优化需要优化模型本身
  - 找到高性价比的模型结构，权衡效果与性能
- 优化数值精度
  - 低精度Embedding，低精度计算
- 优化模型大小
  - 特征剪枝，模型剪枝
- 优化模型结构
  - 模型Op精简



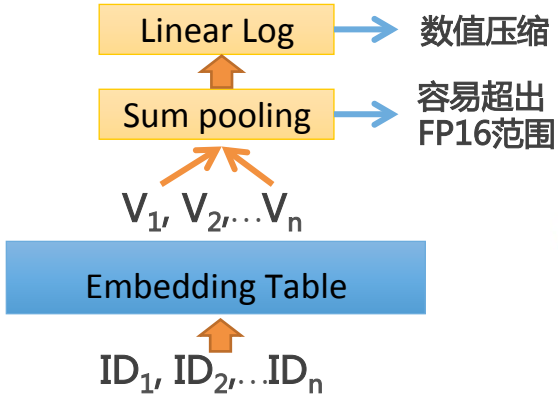
优化数值精度

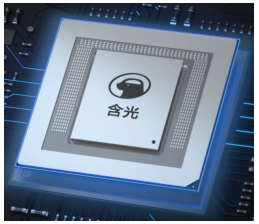
- 收益：降低存储、减少访存、利用Tensor Core/NPU加速

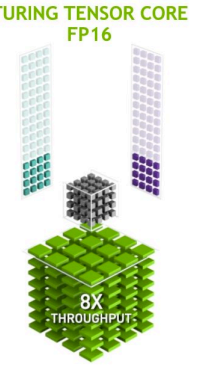
图像CNN模型
INT8普遍应用
RGB像素，长宽固定



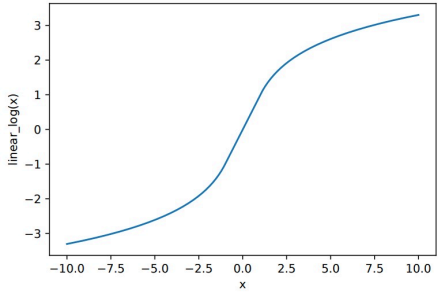
广告推荐模型
FLOAT16 为主
离散特征，长尾明显







$$linear\_log(x) = \begin{cases} -log(-x) - 1 & x < -1 \\ x & -1 \le x \le 1. \\ log(x) + 1 & x > 1 \end{cases}$$



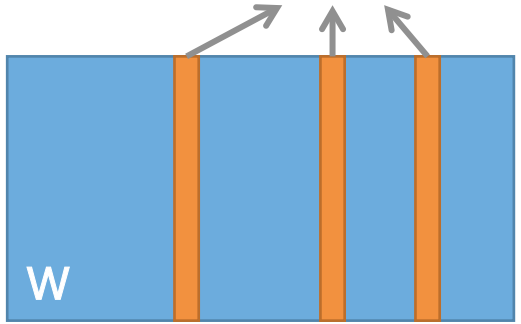
- 数值压缩:无参normalization
- 简单DNN模型性能提升100%+，复杂模型约40%

优化模型大小

- 对模型进行裁剪，在给定精度要求下，尽可能降低算力消耗
- 如何定义算力消耗？
  - 算力消耗 ≠ FLOPS　　稀疏矩阵与稠密矩阵差别很大
  - 性能收益往往需基于压测工具实测数据
- 剪枝方案：对FC Layer神经元个数进行裁剪。服务能力提升20%~30%
  - 基于End2End loss训练
  - 直接产出模型结构

$$min_{w,\lambda} \frac{1}{N} \sum_{i=1}^{N} L(y_i, C(x_i, W, \lambda)) + R(w) + R_s(\lambda)$$

优化神经元的权重系数λ，权重小的被移除



优化模型结构

- 模型持续迭代，模型结构容易存在冗余

- 定向广告精排模型主要OP：
  - GRU：串行计算耗时较大，随着提效和GPU硬件的发展，性价比变低
  - 多路Attention：序列长度边际效应递减

- 优化低性价比模块得到降级模型，同等算力提升rpm收益

Alibaba Group
阿里巴巴集团

效能2.0：结合业务极致优化

从通用优化到极致优化

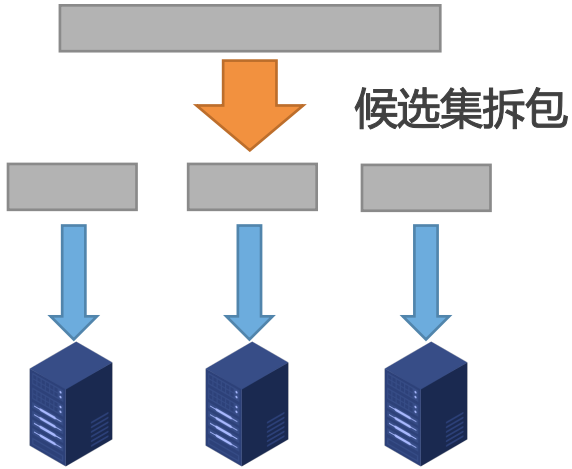
- 传统模型优化往往是通用优化，难以优化到极致

- 算法 - 工程 Co-Design
  - 广告模型：对离散ID预估的高维稀疏模型
  - 使用场景确定、系统约束确定
  - 深入挖掘业务/系统特性可找到优化空间

通用引擎	vs.	专用引擎
TensorFlow		XDL-Blaze
服务所有场景 CV/NLP/RL		服务特定业务

Case Study: 平衡Latency与系统吞吐

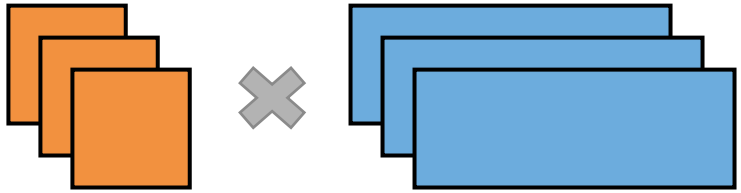
- 在线服务对Latency要求极高，大量使用并行化
  - 使用GPU→价格昂贵
  - 服务级拆包并行→带来额外计算
  - 减小单包batch_size→硬件效率低



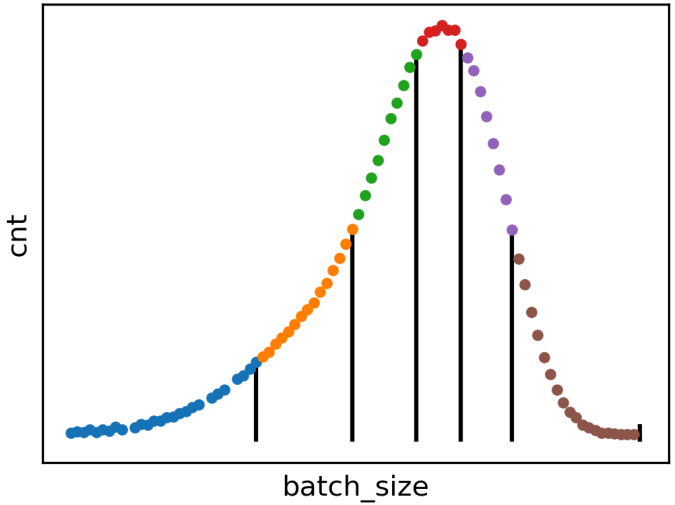
- 解法：结合业务特性，挖掘可被放宽的Latency约束
  - CVR等轻量级模型，使用CPU进行预估
  - 放宽广告召回的实时性要求，离线进行检索和排序，可用万级别Batch_size
  - 预测模型计算时间(基于用户特征个数)，个性化地决定并行度，服务能力提升20%

Case Study: 系统参数精细调优

- 优化特定尺寸的矩阵乘法
  - NVIDIA标准库的性能未必最优
  - 优化计算Kernel，部分情况服务能力提升30%~50%

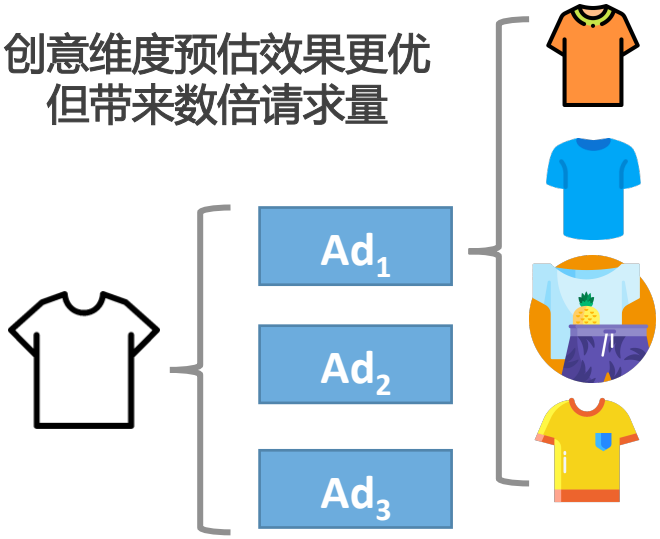


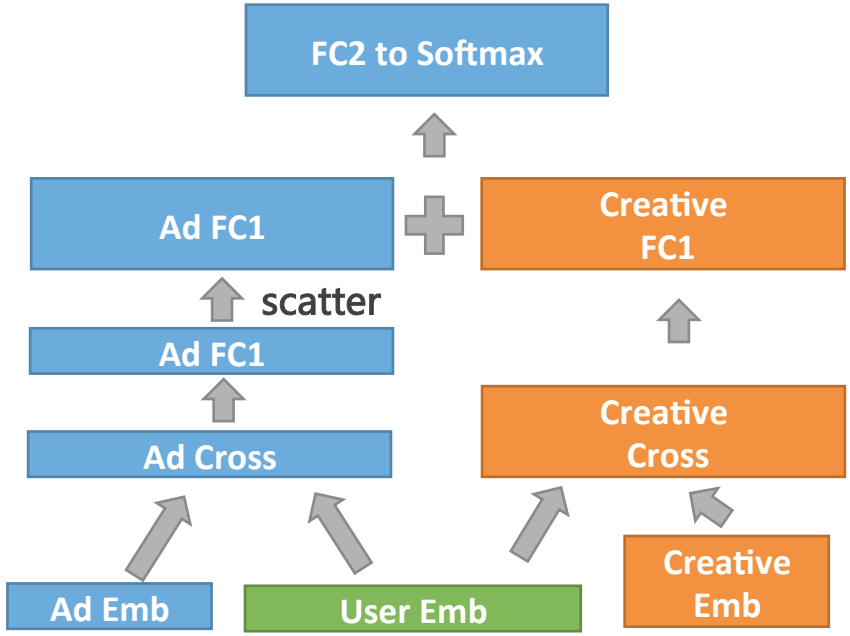
- 基于实际请求数据调优系统参数
  - Tensorflow XLA等系统优化只支持固定batch_size
  - 对于可变batch_size，需进行分桶padding
  - 转化为类似加油站选址的动态规划问题
  - 相比经验参数，服务能力提升10%



Case Study: 层级预估的性能优化

- 策略的维度具有层级：商品维度→广告计划维度→创意维度→位置维度
- 多创意计算融合：尽可能地复用计算，服务能力提升35%



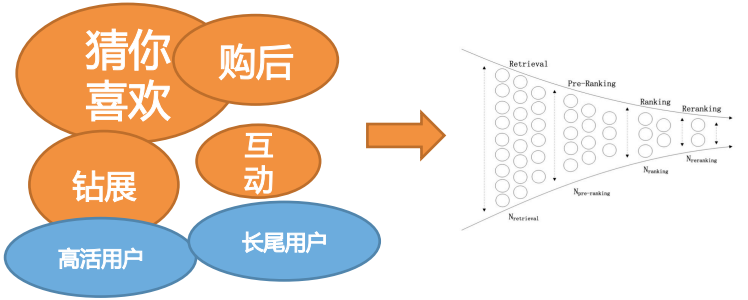


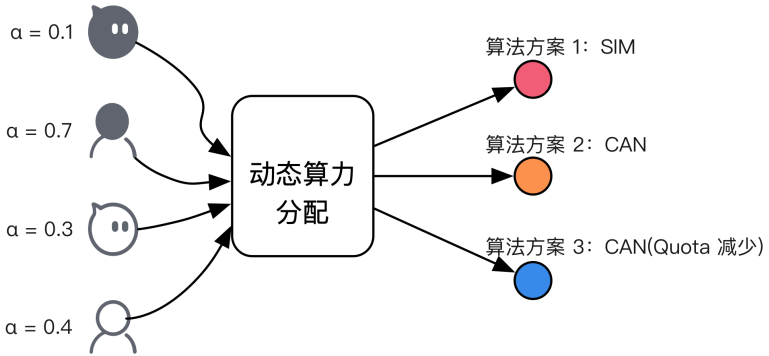
Alibaba Group
阿里巴巴集团

效能3.0：全局优化

从单点优化到全局优化

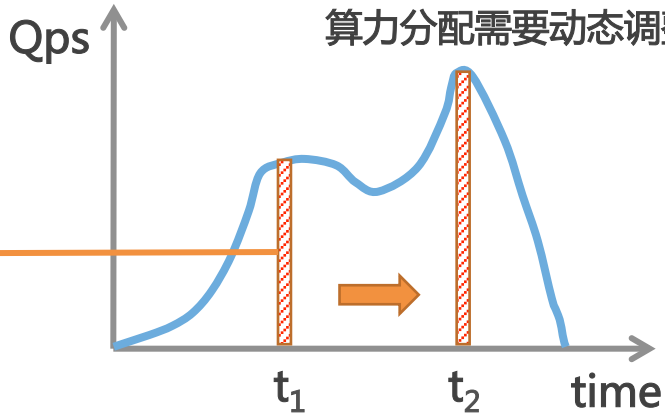
- 广告系统是多场景多模块
- 流量价值差异很大，需根据性价比分配算力
- 流量发生波动时，系统需动态调整分配策略





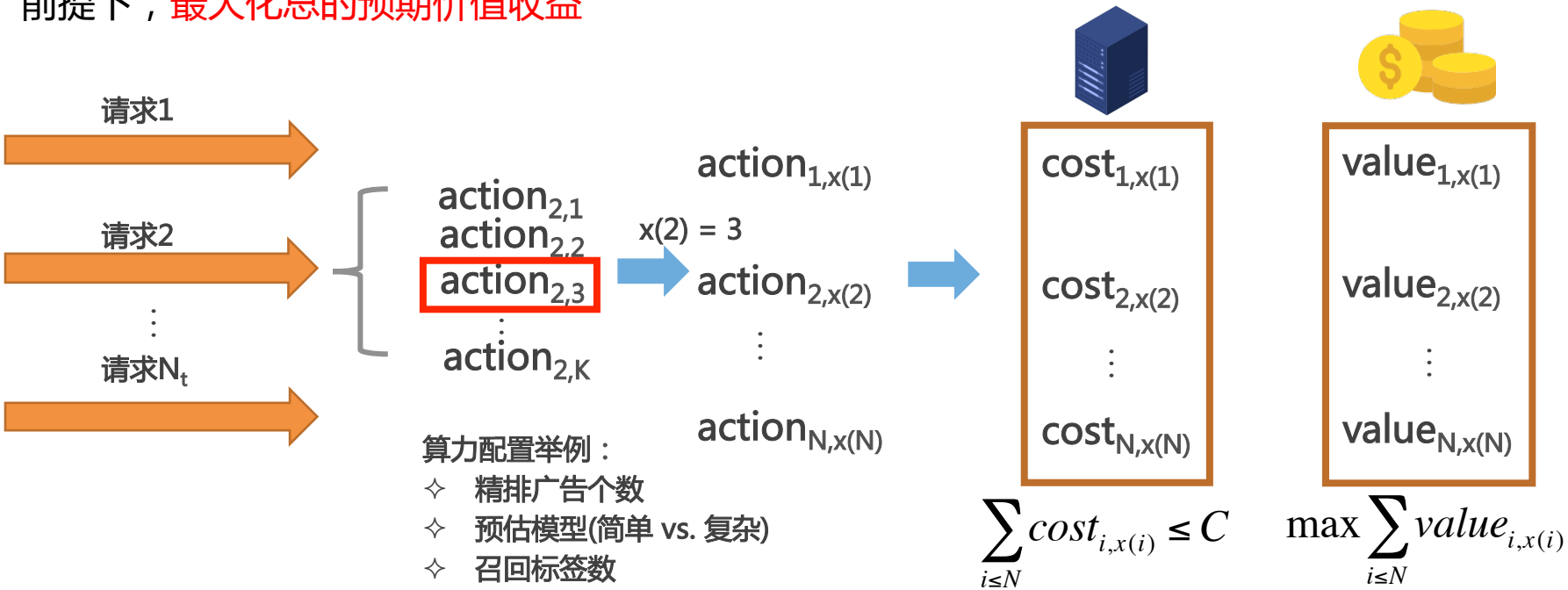
根据性价比分配算力

总资源有限，流量变化时，算力分配需要动态调整



算力分配问题定义

- 每个请求有K个候选算力档位配置。每个算力配置$action_{ij}$有其预计的算力消耗$cost_{ij}$和价值$value_{ij}$
- 对于当前时刻收到的N_t个请求，需要为其选定各自的算力配置，在预估总算力消耗不超过C的前提下，最大化总的预期价值收益



算力配置举例：

- 精排广告个数
- 预估模型(简单 vs. 复杂)
- 召回标签数

Dynamic Computation Allocation Framework (DCAF)

- 将配置选择函数x(i)用indictor的方式表示，可将原问题表示为

$$\max_{x \in dom\ X} \sum_{i \le N, j \le K} x_{i,j} value_{i,j} \quad (dom\ X: x_i \text{ is one-hot vector})$$

$$\text{s.t.} \left(\sum_{i \le N, j \le K} x_{i,j} \text{cost}_{i,j} \right) \le C$$

- 使用Lagrangian 求解带约束的优化问题，引入额外变量 λ，可构造对偶函数g(λ)

$$L(x, \lambda) = -\sum_{i,j} x_{i,j} value_{i,j} + \lambda \left(\sum_{i,j} x_{i,j} cost_{i,j} - C \right)$$

$$g(\lambda) = \min_{x \in dom\ X} -\sum_{i,j} x_{i,j} (value_{i,j} - \lambda cost_{i,j}) - \lambda C$$

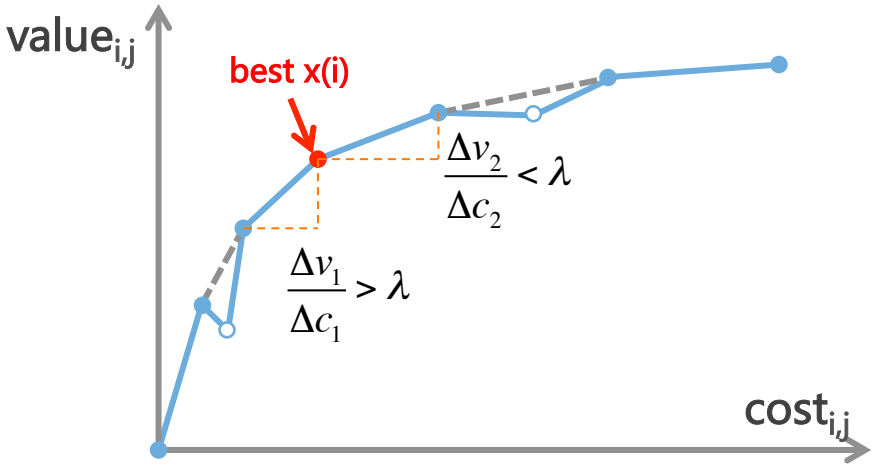
$$= -\left[\sum_{i \le N} \max_{j <= K} (value_{i,j} - \lambda cost_{i,j}) + \lambda C \right]$$

每条流量的的最优选择x(i)可独立确定

- 对于任意给定的λ，对偶函数g(λ)在 $x(i) = \text{argmax}_{j <= K} (value_{i,j} - \lambda cost_{i,j})$ 时最优

λ的实际含义

- 给定λ，各请求的最优策略可独立确定： $x(i) = \underset{j<=K}{\operatorname{argmax}}(value_{i,j} - \lambda cost_{i,j})$
- 类似用单位价值来求解背包问题，λ代表着对”算力性价比”的限制（效能抓手）
  - 当λ变小，放松性价比约束，档位变大，使用的总算力变多，总价值也会变大



什么时候g(λ)取到极大值？

- 当所用算力恰好等于C时取到极值
  - Weak Duality: $g(\lambda^*) \le f(x^*)$ 极值点恰好为可行解

$$g(\lambda) = -[\sum_{i \le N} \max_{j<=K}(value_{i,j} - \lambda cost_{i,j}) + \lambda C]$$

$$= -[\sum_{i \le N} value_{i,x(i)} + \lambda(C - \sum_{i \le N} cost_{i,x(i)})]$$

总价值　　λ * 剩余可用算力

- 静态情况可基于单调性用二分法求解

在真实系统中应用DCAF

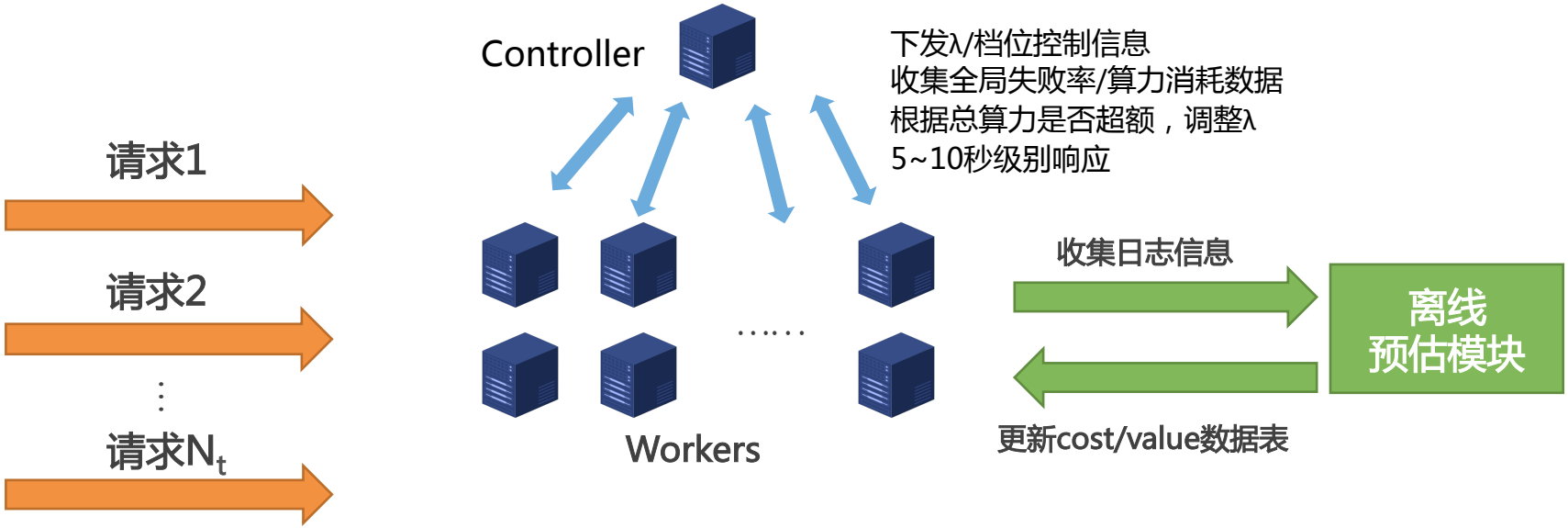
- 如何在分布式环境下，动态求出每时刻的最优解？
  - 求解时，各个请求可独立计算，唯一需要同步的是λ和总算力消耗$\Sigma cost_{i,x(i)}$
  - 只需在全局维护一份当前的λ并实时统计算力消耗
  - 根据总算力消耗与约束C的大小关系，动态调节λ可以时刻保证最优解

- 流量价值value、算力消耗cost、算力约束C如何提供？
  - 影响分配策略的是value/cost的相对效能比值，允许误差，单位无要求
  - 实际算力消耗较难观测，可用系统当前的失败率/超时率反映算力消耗是否满足约束

Jiang et al. *DCAF: A Dynamic Computation Allocation Framework for Online Serving System*

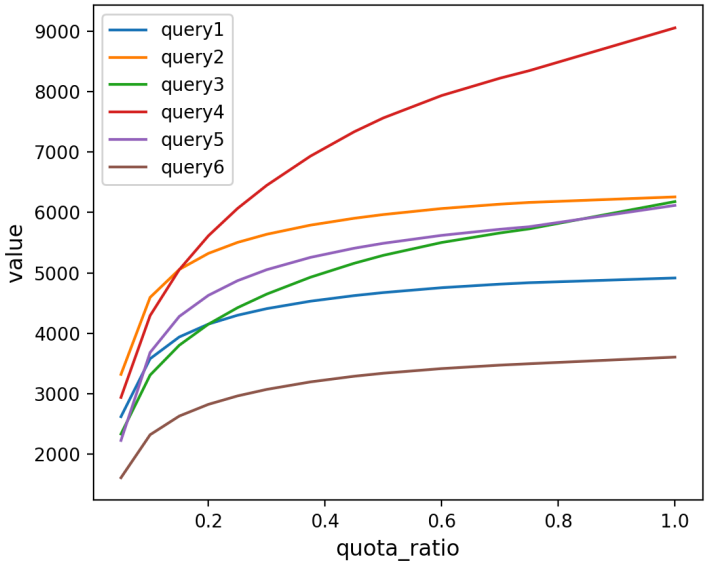
动态引擎架构：Transformers

- 分布式环境下的通用解决方案，适用于召回、排序、调价、创意等多个模块
- 插件式的接入方式，已在定向广告引擎主要模块上线生效



动态引擎落地实践

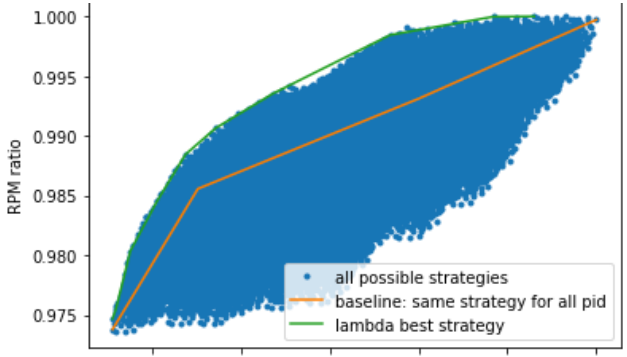
- 确定需要调控的模块和功能点
  - 召回侧：标签式召回、向量检索、TDM树结构
  - Rank侧：特征计算(CPU) + DNN inference(GPU)
- 档位设计
  - 各模块不同算力消耗下的最优配置
- 计算算力消耗cost
  - 体系化地对系统各模块的算力进行度量
- 预估流量价值value
  - 可在场景维度、用户维度基于历史数据统计均值



动态引擎实际收益

✓ 基于流量价值实现最优算力分配

- 在场景/用户间分配算力相比统一档位可带来业务收益
- 精排模块在同等算力下rpm提升2%

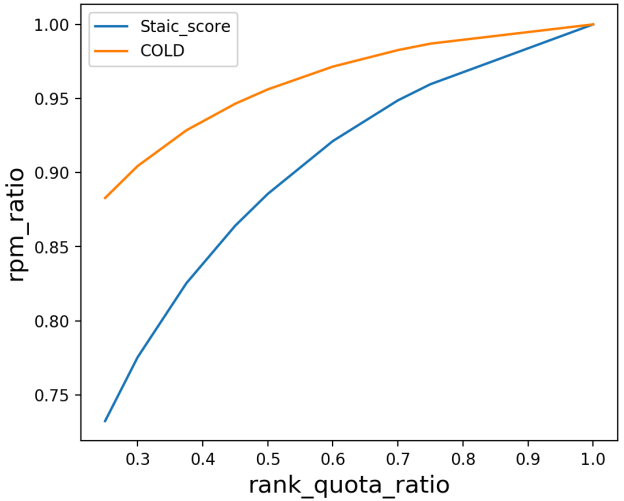


✓ 流量变化无需手动操作预案

- 系统持续维持在临界水位

✓ 基于算力性价比指导整体资源规划

- 初排升级后，使用更少的精排quota可达到同样的效果



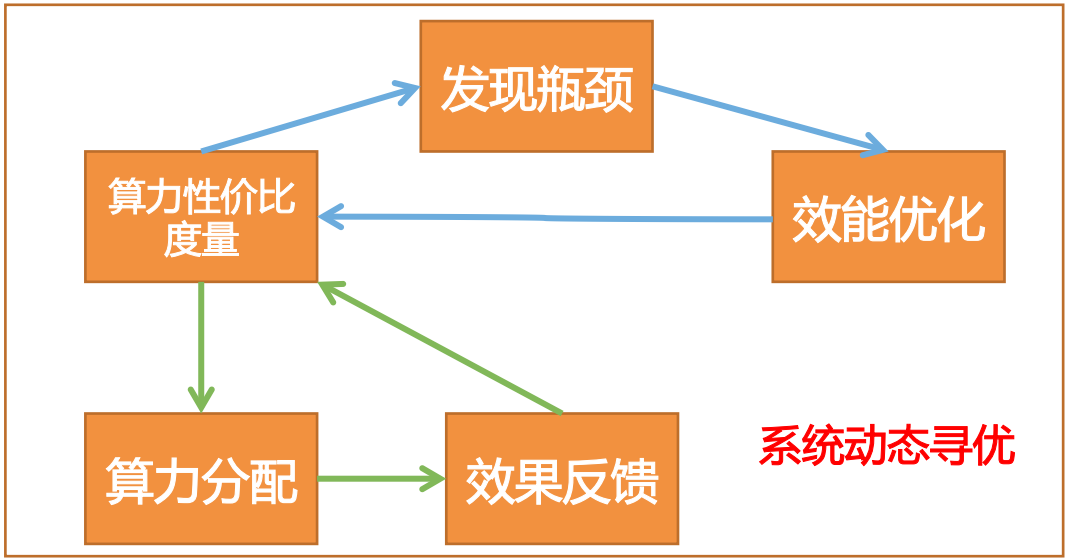
总结

- 效能优化从1.0发展到3.0，在多个层面上提高算力性价比
- 效能体系的搭建和运作并非一朝一夕，而是需要不断与整体系统共同迭代

1.0 数值精度/模型尺寸/Op算力性价比

2.0 结合业务特点提升算力性价比

3.0 流量/模块间基于算力性价比的分配



Thanks