开发者社区 × 飞天大数据平台 APSARA BIG DATA PLATFORM | 飞天AI平台 APSARA AI PLATFORM

阿里巴巴大数据及AI实战

阿里经济体大数据及AI典型场景最佳实践全揭秘

> “云时代的数据智能，可以真正处理海量的数据，实时地进行数据分析，真正把人工智能和大数据完美结合，提炼数据的内在规律。”

——阿里云智能计算平台事业部总裁　贾扬清

阿里云

Alibaba Group
阿里巴巴集团

阿里技术

扫一扫二维码图案，关注我吧

阿里云实时计算

MaxCompute 开发者交流钉钉群

DataWorks 开发者交流钉钉群

扫码关注阿里技术

序言

贾扬清
阿里云智能
计算平台事业部总裁

2020 年我们如果问企业 IT 最大的趋势是什么，我觉得云计算必然会排在前列。今天，上云是 IT 基础设施继续向企业提供能力升级的必然趋势，通过稳定、快捷、高性能和高弹性的底座，帮助企业迅速实现已有业务的数字化，以及推动现有数字信息的实现业务价值。

IT 的基础设施上云只是一个开始。云的最大价值，用一句话来说，就是“数据让应用智能化”。从阿里巴巴经济体的角度来说，未来数据智能技术发展的两大方向，一是实时化的大数据能力，二是人工智能技术。云时代的数据智能，可以真正处理海量的数据，可以真正实时地进行数据的分析，也可以真正把人工智能和大数据完美结合，提炼数据的内在规律。

在阿里云提供的统一技术平台上，阿里巴巴的各个业务部门沉淀了很多优秀的方法论。我们非常高兴用这一本实践手册作为给开发者社区和企业用户的献礼。通过这些最佳实践的分享，我们希望能够和企业，和开发者一起探索，进一步推动数据智能领域的创新和落地。

目录

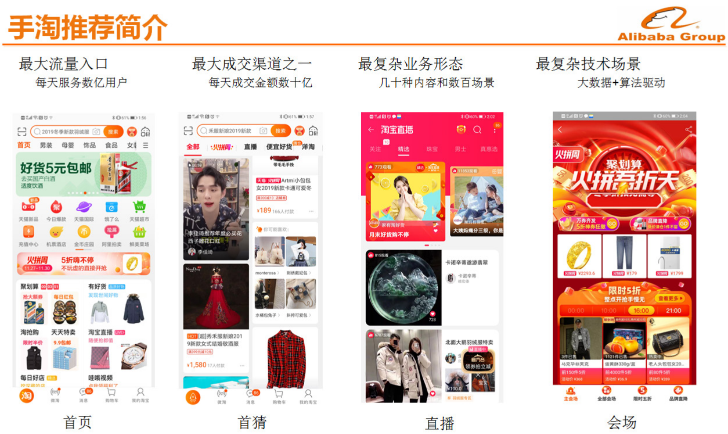
解密淘宝推荐实战，打造“比你还懂你”的个性化 APP

作者：欧文武（三桐） 阿里集团 淘宝事业群 资深算法专家

简介：如今，推荐系统已经成为各大电商平台的重要流量入口，谁才能够做到比用户更懂用户，谁占据了新零售时代的主动权。手机淘宝的推荐更是淘宝最大的流量入口和最大的成交渠道之一，其背后是最为复杂的业务形态和最复杂的场景技术，那么究竟如何打造手淘背后的推荐系统呢？本次首席技术官大数据专享会上，阿里巴巴搜索推荐事业部资深算法专家欧文武（三桐）为大家解密了淘宝的推荐实战。

手淘推荐简介

手淘推荐的快速发展源于 2014 年阿里“All in 无线”战略的提出。在无线时代，手机屏幕变小，用户无法同时浏览多个视窗，交互变得困难，在这样的情况下，手淘借助个性化推荐来提升用户在无线端的浏览效率。经过近几年的发展，推荐已经成为手淘上面最大的流量入口，每天服务数亿用户，成交量仅次于搜索，成为了手淘成交量第二大入口。



今天的推荐不仅仅包含商品，还包含了直播、店铺、品牌、UGC，PGC 等，手淘整体的推荐物种十分丰富，目前手淘的整体推荐场景有上百个。推荐与搜索不同，搜索中用户可以主动表达需求，推荐很少和用户主动互动，或者和用户互动的是后台的算法模型，所以推荐从诞生开始就是大数据 +AI 的产品。

手淘推荐特点

相比于其他推荐产品，手淘推荐也有自身的如下特点：

1. **购物决策周期**：手淘推荐的主要价值是挖掘用户潜在需求和帮助用户购买决策，用户的购物决策周期比较长，需要经历需求发现 , 信息获取，商品对比和下单决策的过程，电商推荐系统需要根据用户购物状态来做出推荐决策。
2. **时效性**：我们一生会在淘宝购买很多东西，但是这些需求通常是低频和只在很短的时间窗口有效，比如手机 1~2 才买一次但决策周期只有几小时到几天，因此需要非常强的时效性，需要快速地感知和捕获用户的实时兴趣和探索未知需求 , 因此，推荐诞生之初就与 Flink、Blink 实时计算关系非常紧密。
3. **人群结构复杂**：手淘中会存在未登录用户、新用户、低活用户以及流式用户等，因此需要制定差异化的推荐策略，并且针对性地优推荐模型。
4. **多场景**：手淘推荐覆盖了几百个场景，每个场景都独立进行优化显然是不可能的，而且每个场景的条件不同，因此超参也必然不同，无法依靠人工逐个优化场景模型的参数，因此需要在模型之间进行迁移学习以及自动的超参学习等，通过头部场景的迁移学习来服务好尾部场景。
5. **多目标和多物种**。

手淘推荐特点

1. 购物决策周期

➢ 需求探索->信息获取->购买决策->下单(流失)

➢ 根据用户购物状态采取相应的推荐策略

2. 时效性

➢ 相同需求非高频激活，生命周期短

➢ 实时计算和在线推理，实时捕获和预测用户兴趣

3. 人群结构复杂

➢ 未登录，新用户，低活用户，流失用户等

➢ 分人群建模和分人群推荐策略；

4.多场景

➢ 上百个推荐场景

➢ 高频场景深度优化，通过技术迁移平台化支持中低频场景

5. 多目标

➢ 点击率，停留时长，成交等

➢ 多目标联合优化

6. 多物种

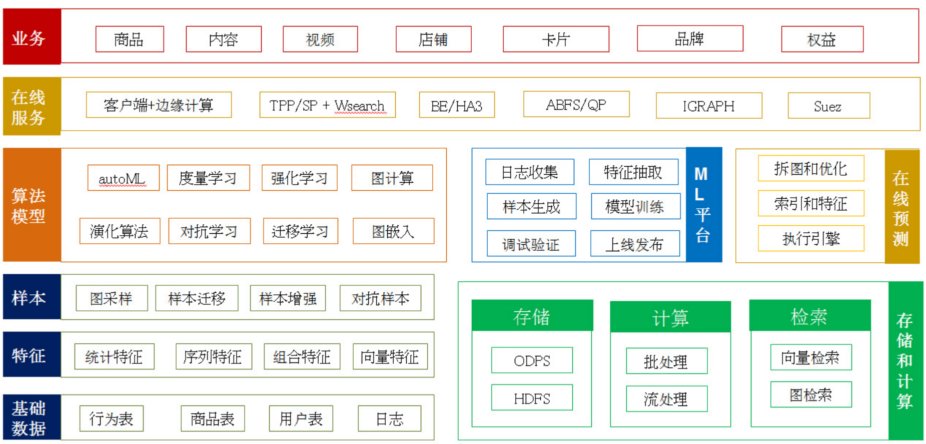
➢ 商品、视频、直播、内容等

➢ 多模态学习，异构图表示学习

推荐技术框架

如下图所示的是手淘推荐的技术框架。2019 年双 11，整个阿里巴巴的业务全部实现上云，因此手淘推荐的技术架构也是生长在云上的。推荐的 A-B-C 包括了推荐算法和模型、原始日志和基于日志加工出来的特征和离在线计算及服务能力，比如向量检索、机器学习平台、在线排序服务等。除了云，今年我们通过把深度学习模型部署到了端上，实现了云和端的协同计算。

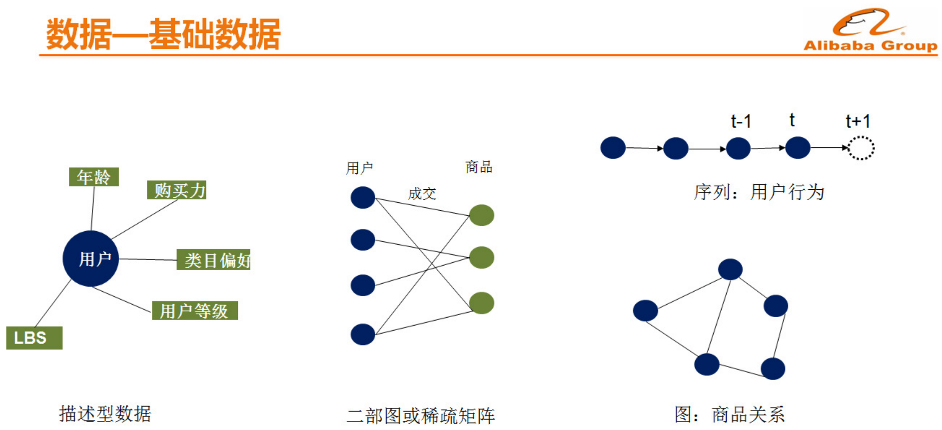
推荐技术框架



接下来将主要围绕数据、基础设施以及算法模型进行介绍。

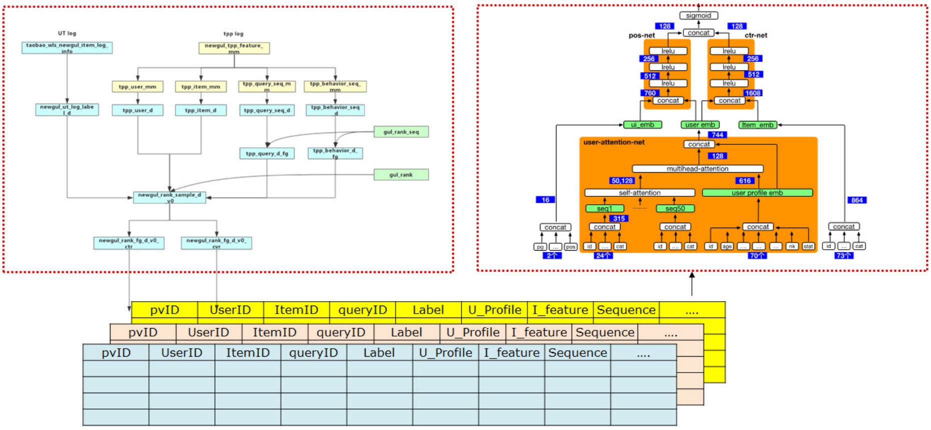
数据 – 基础数据

手淘的推荐数据主要包括几种，即描述型数据比如用户画像，关系数据比如二部图或稀疏矩阵，行为序列和图数据等。基于用户行为序列推荐模型在手淘商品推荐应用最为广泛，图模型则是近两年发展较快的模型，因为序列通常只适合于同构的数据，而在手淘里面，用户的行为有很多种，比如看视频、搜索关键词等，通过 graph embedding 等技术可以将异构图数据对齐或做特征融合。



数据 – 样本

数据样本主要包含两部分元素，label 和特征。label 一般在手淘推荐中有几类，比如曝光、点击、成交以及加购等。特征则比较多了，比如用户自己的特征、用户上下文特征、商品本身特征以及两两组合特征等。根据用户的特征和行为日志做 Join 就形成样本表，这些表格存储的时候就是按照稀疏矩阵方式进行存储，一般而言是按天或者按照时间片段形成表格，样本生成需要占用很大一部分离线计算资源。



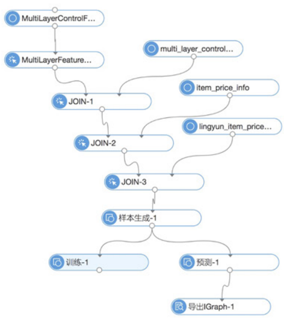
离线计算 – 计算模式

离线计算主要有三种模式，即批处理、流处理和交互式查询。批处理中比较典型的就是 MapReduce，其特点是延迟高但并行能力强，适合数据离线处理，比如小时 / 天级别特征计算，样本处理和离线报表等。流计算的特点是数据延迟低，因此非常适合进行事件处理，比如用户实时点击，实时偏好预测，在线学习的实时样本处理和实时报表等。交互式查询则主要用于进行数据可视化和报表分析。

离线计算—计算模式

Alibaba Group

1. **Batch data processing （批处理）**
   - 延迟：分钟到小时（任务驱动，有始有终）
   - 应用：数据合并、统计和离线预测
   - 平台：ODPS/ BLINK/SPARK/HADOOP
2. **Streaming data processing（流处理）**
   - 延迟：毫秒到秒级 （消息驱动，无始无终）
   - 应用：事物处理，实时统计，机器学习和实时预测
   - 平台： BLINK/ SPARK/STROM
3. **Interactive query（交互式查询）**
   - 延迟：秒级
   - 应用：数据可视化，生产报表，复杂图查询
   - Hologres/Gremlin



离线计算 – 模型训练

模型训练也有三种主要的模式，即全量学习、增量学习和在线学习。全量学习这里是指模型初始化从 0 开始学习，如果日志规模比较小，模型简单并不需要频繁更新时，可以基于全量日志定期训练和更新模型，但当日志和模型参数规模较大时，全量学习要消耗大量计算资源和数天时间，性价比很低，这时通常会在历史模型参数基础上做增量学习，用小时 / 天日志增量训练模型和部署到线上，降低资源消耗和较高的模型更新频率。如果模型时效性非常强需要用秒 / 分钟级别样本实时更新模型，这是就需要用到在线学习，在学习和增量学习主要差别是依赖的数据流不一样，在线学习通常需要通过流式计算框架实时产出样本。

离线计算—模型训练

1. **Batch learning （全量学习）**
   - 特点：消耗大量存储和计算资源，更新慢
   - 时长：数天 （15 天 * 6 小时/天）
   - 应用：适合新模型从0开始学习，模型迭代升级
2. **Incremental learning（增量学习）**
   - 特点：数据从生产到消耗时延适中，节省资源
   - 时长：3~8小时/天
   - 应用：模型日常更新，日常生产
3. **Online learning（在线学习）**
   - 特点：数据从生产到消耗时延低
   - 时长：分钟级
   - 应用：强化学习，online LTR，在线策略优化，大促CTR

全量学习和增量学习：以信息流CTR 模型为例， 10B参数，日均10B样本

离线计算 – 训练效率

因为机器资源总是不够的，训练优化是如何用更快的速度，更少的计算和更少的数据训练出更好的模型，这里为大家提供一些加速训练的方式：

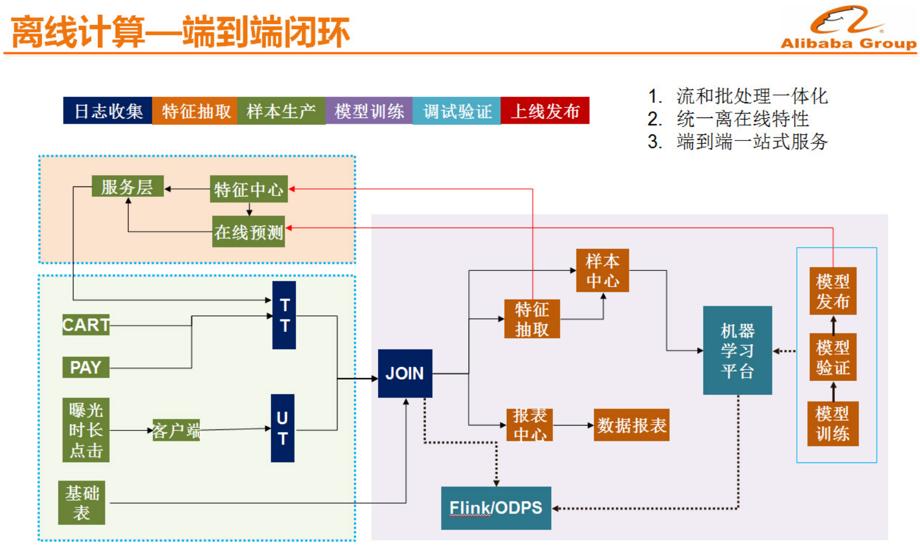
1. **热启动**：模型需要不断升级和优化，比如新加特征或修改网络结构，由于被修复部分模型参数是初始值，模型需要重新训练，热启动就是在模型参数只

有部分修改时如何用少量的样本让模型收敛。

2. **迁移学习**：前面提到手淘推荐的场景非常多，而某些场景的日志非常少，因此无法实现大规模模型的训练，这是可以基于样本较多的大场景做迁移学习。
3. **蒸馏学习**：手淘用来做级联模型学习，比如精排模型特征更多模型更加精准，通过精排和粗排特征蒸馏，提升粗排模型精度，除此之外也可以用来做模型性能优化；
4. **低精度、量化和剪枝**：随着模型越来越复杂，在线存储和预测成本也在成倍增加，通过这些方式降低模型存储空间和预测速度，另外是端上模型通常对大小有强要求；

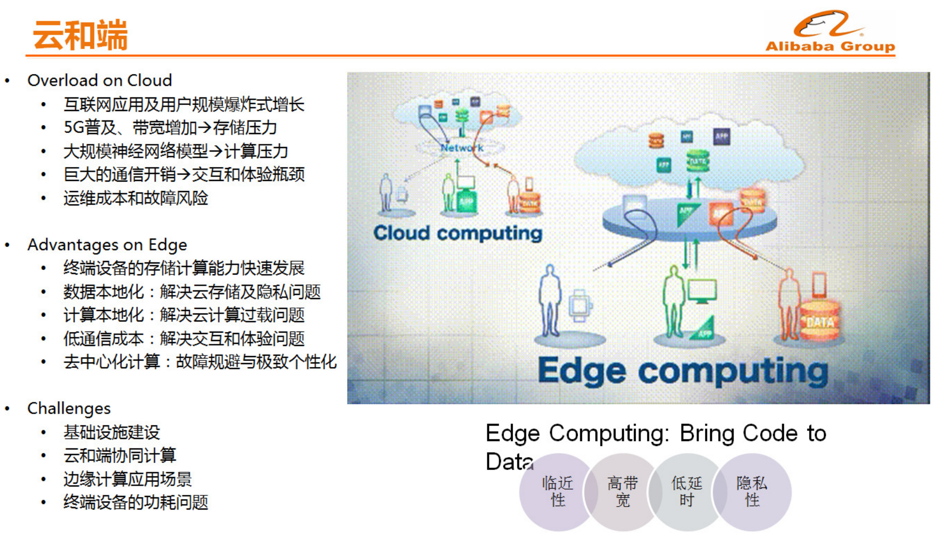
离线计算 – 端到端闭环

因为手淘推荐日志很大，特征来源很复杂，离线和在线的细微变动都可能导致样本出错或离线在线特征 / 模型不一致，影响迭代效率甚至造成生产故障，我们的解决办法是做一个端到端的开发框架，开发框架对日志，特征和样本做抽象，减低人工开发成本和出错的可能，并在框架嵌套 debug 和数据可视化工具，提高问题排查效率。目前手淘搜索推荐已经基本上做到了从最原始日志的收集、到特征抽取以及训练模型的验证、模型的发布，再到线上部署以及实时日志的收集形成整体的闭环，提升了整体模型的迭代效率。



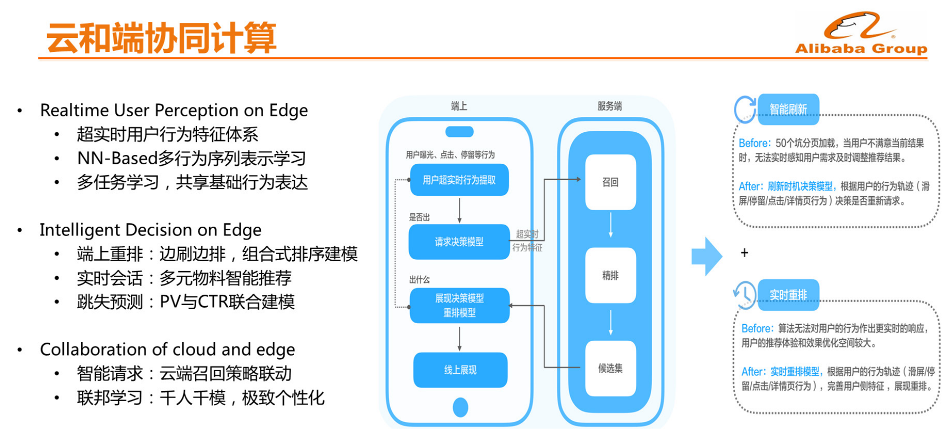
云和端

随着 5G 和 IOT 的发展数据会出现爆炸式的膨胀，将数据放在云上集中存储和计算，这样做是否是一个最合理的方式呢？一些数据和计算能否放在端上来做？端上相对于云上而言，还有几个较大的优势，首先延时低，其次是隐式性，各个国家对于隐私的保护要求越来越严厉，因此需要考虑当数据不能发送到云端的时候如何做个性化推荐。



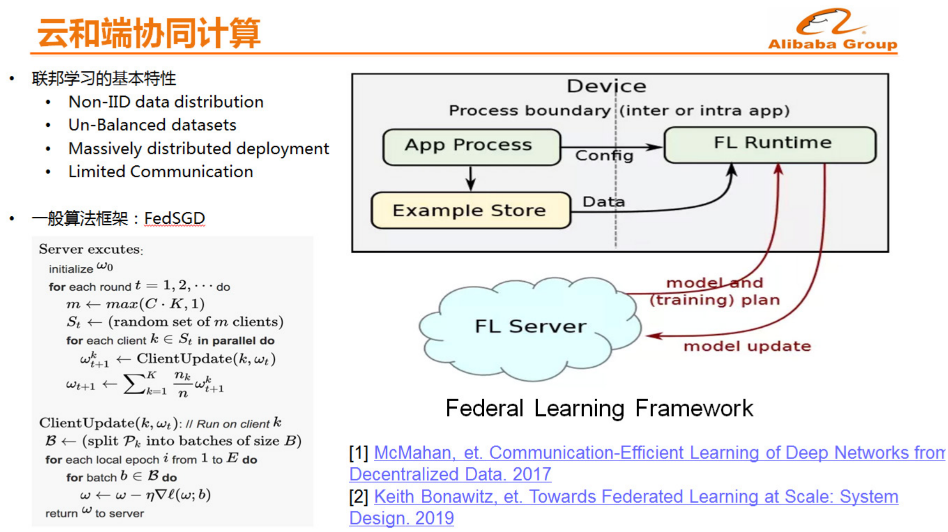
云和端协同计算

在云和端协同计算方面，阿里巴巴已经做了大量的尝试，比如云和端如何实现协同推理，这里包括几个部分，比如手机端上拥有更加丰富的用户行为如用户滑屏速度、曝光窗口时长以及交互时长等，因此第一步是端上的用户行为模式感知的模型。第二步就是在端上决策，比如预测用户何时会离开 APP，并在用户离开之前改变一些策略提高用户的浏览深度。此外，手淘还在端上做了一个小型推荐系统，因为目前云上推荐都是一次性给多个结果比如 20 多个，而手机一次仅能够浏览 4 到 6 个推荐结果，当浏览完这 20 个结果之前，无论用户在手机端做出什么样的操作，都不会向云端发起一次新的请求，因此推荐结果是不变化的，这样就使得个性化推荐的时效性比较差。现在的做法就是一次性将 100 个结果放在手机端上去，手机端不断地进行推理并且更新推荐结果，这样使得推荐能够具有非常强的时效性，如果这些任务全部放在云端来做，那么就需要增加成千上万台机器。



A Example diagram of EdgeRec

除了推理之外，还有云和端的协同训练。如果想要实现个人的隐私保护，云和端协同训练是非常重要的，只有这样才能够不将用户的所有原始数据全部加载到云上，大部分训练都在手机端完成，在云端只是处理一些不可解释的用户向量，从而更好地保护用户的隐私数据。



召回技术 – 动态实时多兴趣表达 (MIND)

早些年大家在做推荐协同过滤可能使用 Item2Vec 召回、标签召回等，比如像 Item2Vec 召回而言，确实比较简单，而且时效性非常好，在很长一段时间内主导了推荐技术发展的进程，后续才诞生了矩阵分解等。但是 Item2Vec 召回存在很大的问题，如果商品的曝光点不多其实是很难被推荐出来的，因此推荐的基本上都是热门的 Item。其次 Item2Vec 召回认为每个点击都是独立的，缺少对于用户的全局认知，此时需要做的是就是将用户的行为和标签进行全局感知并做召回。基于这样的出发点，我们提出了基于行为序列的召回模型，但这种方式存在的问题就是用户的兴趣不会聚焦在同一个点，单个向量召回通常只能召回一个类目或者兴趣点，因此如何通过深度学习做用户的多需求表达等都是挑战。这样的问题，阿里巴巴已经解决了，并且将论文发表在 CIKM 2019 上面。现在，淘宝所使用的是在线多向量化并行召回。

召回技术-动态实时多兴趣表达(MIND)

Alibaba Group

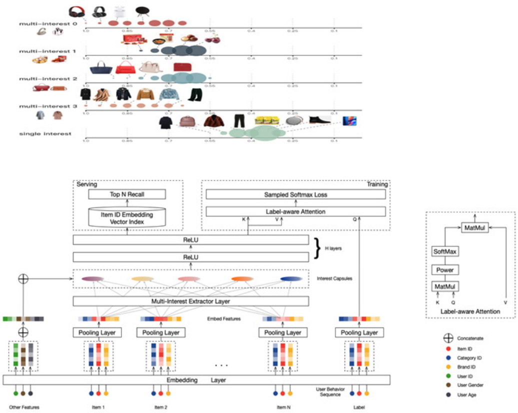
Motivation

- ➢ 多兴趣建模：用户兴趣是多峰分布的，单一表达很难精确刻画用户
- ➢ 兴趣的动态性：用户兴趣在不断变化，需要实时生成用户的兴趣表达
- ➢ 学习端到端的召回模型

方案

- ➢ 使用动态路由的机制对用户的兴趣多峰分布进行动态建模
- ➢ 通过 Label-Attention 加速训练
- ➢ 在线多向量化并行召回 ＋ GPU加速

CIKM 2019



CTR 模型

手淘推荐的 CTR 模型也经历了几个重要的变革，第一个模型是 FTRL+LR，其优点是模型简单，能够支持千亿级别特征。第二个模型是 XNN，对 LR 离散特征做 embedding，并引入多层神经网络，由于引入新的参数，模型学习能力更强。第三个模型是 Self-attention CTR，也就是基于图和用户行为序列实现的。

CTR模型

Alibaba Group

FTRL + LR
- 特征离散化和组合
- 千亿特征训练中自适应裁剪，预测约50亿特征

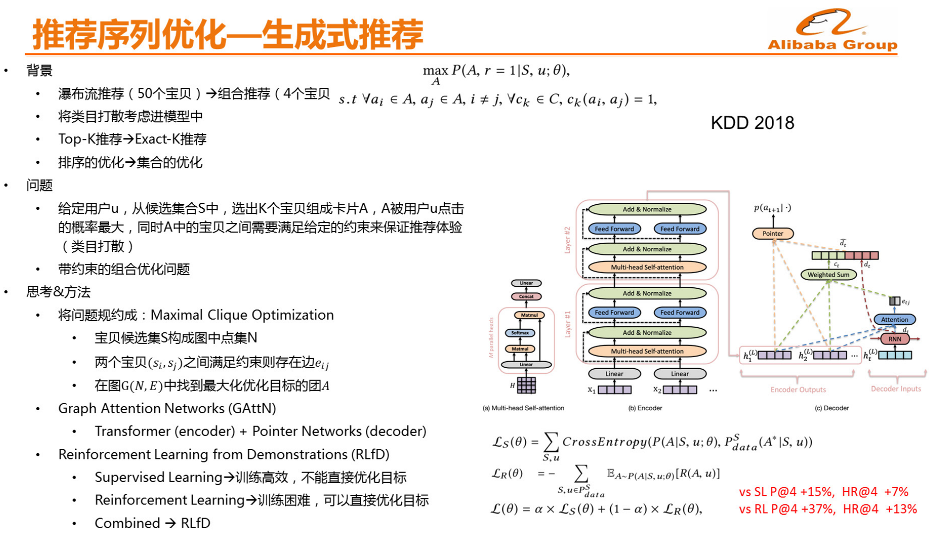
XNN
- 特征变长embedding和浅层MLP
- 增量学习节省存储和计算
- CTR CVR co-train

Self-attention CTR
- 双层multi-head表征用户行为序列
- 商品属性，停留时间和多序列引入
- 增量学习，万亿样本

- PAN CTR
  - 引入上下文和匹配等wide 特征
  - 在不影响原来模型基础上PAN 融合多个深度模型
- Target attention CTR
  - 结构化训练，加速比提升8倍
  - Target attention匹配历史兴趣点
- GNN CTR（ongoing）

推荐序列优化 – 生成式推荐

推荐一般都是基于打分的，打完分之后在做一个贪心排序和打散，这样的做法得到的结果其实并不是最优的，因为这样做并没有考虑结果与结果之间的依赖性，使用贪心算法得到的结果并不是最优的。推荐本质上应该是对于集合而不是序列的优化，因此手淘推荐是用的是生成式排序模型。更多可以参考我们在 KDD 2019 发表的论文。



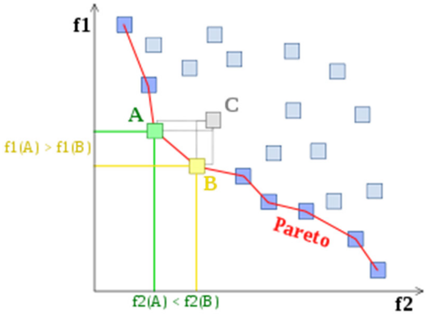
多目标均衡优化

在推荐时，大家往往会遇到多目标均衡问题，比如商品推荐的浏览深度，点击和成交，由于目标量纲不一致，不存在全局唯一最优解，需要同时优化多个目标或在多个目标之间做合理取舍，对此我们提出了基于帕累托的多目标优化排序模型。更多可参考我们发表在 RecSys 2019 的文章。

多目标均衡优化

Alibaba Group

- $Reward = a * Click + b *$ Pay + c * Price
- 帕累托解
  - 考虑最小化n个目标函数，当一个解无法在不牺牲一个目标从而获得其他目标的提升时，这个解达到帕累托最优。
  - 帕累托解可能为无穷个，所有帕累托解构成一个帕累托前沿。
- 效果兑换问题
  - 达到pareto最优之前，都有优化空间，不是兑换
  - 在pareto前沿曲线上调整目标，是最优兑换。
- 既要又要还要问题
  - 在现有模型能力的最优平衡解：想要一个目标涨，必然另一个目标跌



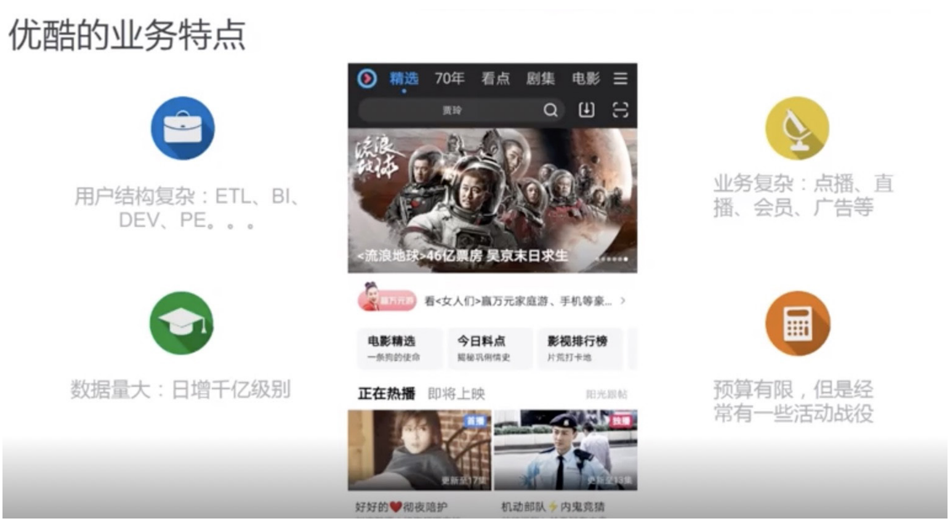
RecSys 2019

优酷背后的大数据秘密

作者：门德亮　阿里集团 新零售技术事业群 数据技术专家

在本文中优酷数据中台的数据技术专家门德亮分享了优酷从 Hadoop 迁移到阿里云 MaxCompute 后对业务及平台的价值。

大家好，我是门德亮，现在在优酷数据中台做数据相关的事情。很荣幸，我正好见证了优酷从没有 MaxCompute 到有的这样一个历程，因为刚刚好我就是入职优酷差不多 5 年的时间，我们正好是在快到 5 年的时候，去做了从 Hadoop 到 MaxCompute 的这样一个升级。这个是 2016 年 5 月到 2019 年现在的 5 月优酷的发展历程，上面是计算资源，下面是储存资源。大家可以看到整个用户数，还有表的数据，实际上是在呈一个指数式增长的。但是在 2017 年 5 月，当优酷完成了整个 Hadoop 迁移 MaxCompute 后，优酷的计算消耗，还有储存的消耗实际上是呈下降趋势的，整个迁移得到了一个非常大的收益。



下面说一下优酷的业务特点。

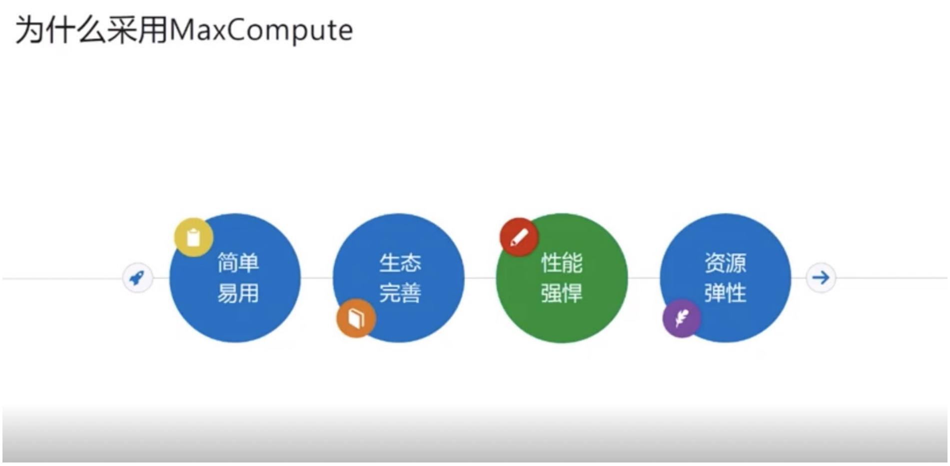
第一个特点从大数据平台整个的用户复杂度上面，不止是数据的同学和技术的同学在使用，还会包括一些 BI 同学，测试同学，甚至产品运营都可能去使用这个大数据的平台。

第二个特点就是业务复杂，优酷是一个视频网站，它有非常复杂的业务场景，从日志分类上，除了像页面浏览，还会有一些播放相关的数据、性能相关的数据。从整个的业务模式上，有直播、有会员、有广告、有大屏等这样一些非常不一样的场景。

第三个特点，就是数据量是非常巨大的，一天的日志量会达到千亿级别，这是一个非常庞大的数据量，而且会做非常复杂的计算。

第四个是比较有意思的，不管是小公司、大公司，对成本的意识是非常高的。优酷也是有非常严格的预算，包括在阿里集团内是有非常严格的预算系统的，但是我们也经常会去做一些重要的战役，像双十一战役，像我们暑期的世界杯战役，还有春节也会搞各种战役。这样的话，其实对计算资源的弹性要求是非常高的。

基于上面的优酷的业务特点，我整理了 MaxCompute 可以完美的支持我们业务的几个特点。



第一个，简单易用。

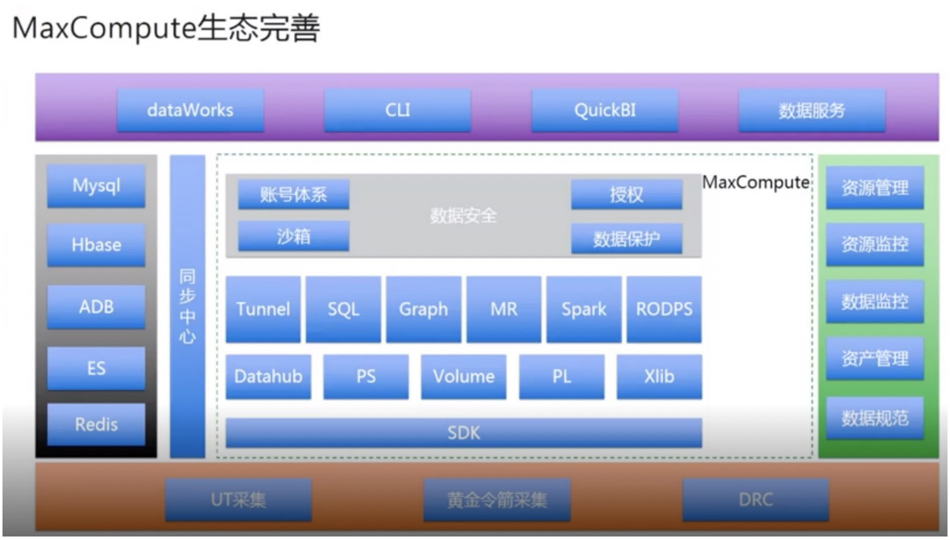
第二个，完善的生态。

第三个，性能非常强悍。

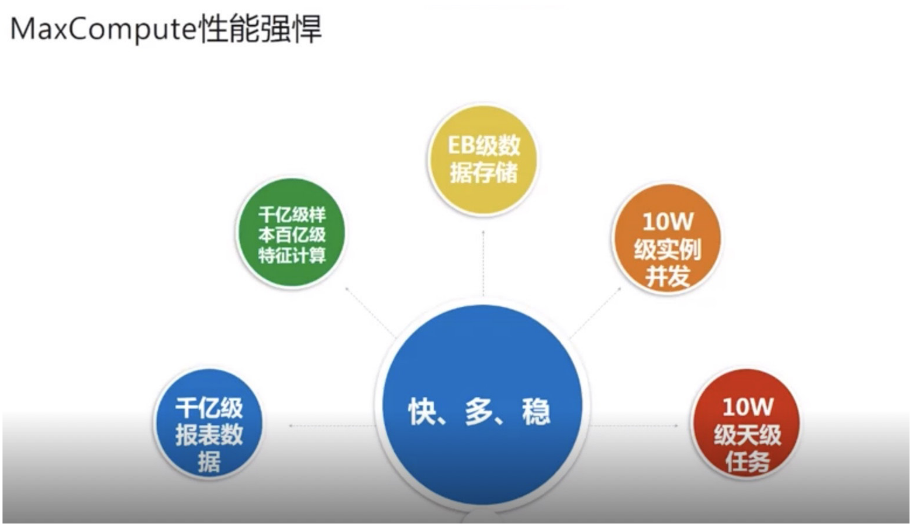
第四个，资源使用非常弹性。

第一个特点，简单易用。MaxCompute 有一个非常完整的链路，不管是从数据开发，还是数据运维，包括数据集成，数据质量的管控，还有整个数据地图，数据安全。当年优酷从 Hadoop 迁到 MaxCompute 之后，我们最大的体会是自己不用半夜经常起来去维护集群了，不用去跑任务了，写一个任务，别人之前提一个需求过来，我可能要给他排几周，而现在我可以告诉他，我给你马上跑一下，就可以出来了。包括之前像分析师 BI 还要登录客户端，写脚本，自己写调度，经常会说我的数今天为什么没出来？包括高层看的数，可能要到 12 点钟才能出来。而现在基本上所有重要的数据都会在 7 点钟产出，包括一些基本的业务需求，其实分析师或者产品，他们自己都可以实现了，不需要所有需求都提到数据这边。

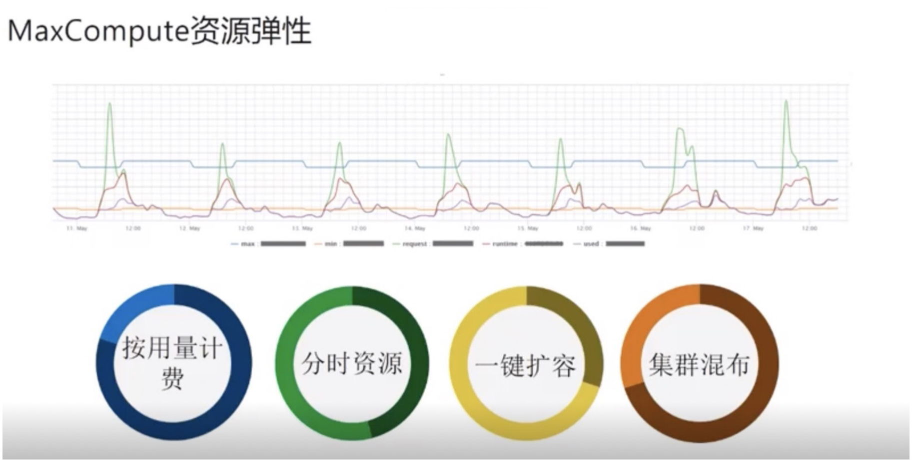
第二个特点，完整的生态。优酷在 2017 年之前是完全基于 Hadoop 的生态，迁到 MaxCompute 之后，是基于阿里云提供的 Serverless 大数据服务的生态。大家可以在开源上看到的组件，在整个的 MaxCompute 上都是有的，而且比开源的要更好用、更简单。从架构图上可以看到，我们中间是 MaxCompute，左侧依赖的 Mysql、Hbase、ES、Redis 这些都是由同步中心去做一个双向的同步。右侧会有资源管理、资源监控、数据监控，包括数据资产，还有一些数据规范。我们下层的数据输入，包括一些集团的采集工具，再往上边，有提供给开发人员用的 DataWorks，包括一些命令行的工具；有提供给 BI 人员用的 QuickBI 及数据服务。



第三个特点，强悍的性能，MaxCompute 支撑了优酷 EB 级的数据存储，千亿级的数据样本分析，包括千亿级的数据报表，10W 级实例的并发、任务。这些在之前维护 Hadoop 的时候，是想都不敢想的。



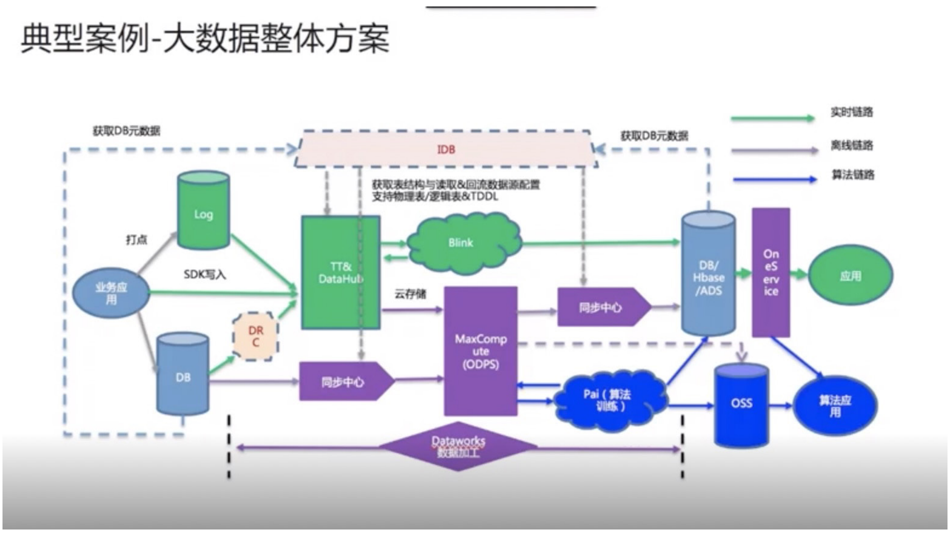
第四个特点，资源使用的弹性。我们在 2016 年迁移之前，其实优酷的 Hadoop 集群规模已经达到了一千多台，这个当时还是一个比较大的规模。当时我们遇到了很多问题，包括像 NameNode 这种内存的问题，机房没有办法再扩容的问题，当时是非常痛苦的，包括一些运维管理上面的问题。我们不断的去问运维要资源，运维告诉说，说你们已经花了多少多少资源，花了多少多少钱。我们面临的问题是计算资源如何按需使用，夜里的时候作业很多，到了下午之后，我的整个集群都空下来了，没有人用，造成了浪费。其实 MaxCompute 完美的解决了这个问题。



第一个，它是按用量计费的，不是说给你多少台机器，然后就收你多少钱的，真的是你用了多少资源收多少钱的，这个在成本上来说，比自己去维护集群，可能是一个砍半（降 50%）这样的收益。

第二个，实际上 MaxCompue 计算资源是可以分时的，比如说生产队列，凌晨的时候会调高一些，保证报表能够尽快出来。到白天时候，让开发的计算资源高一些，可以让分析师、开发去临时跑一些数据，会更顺畅一些。

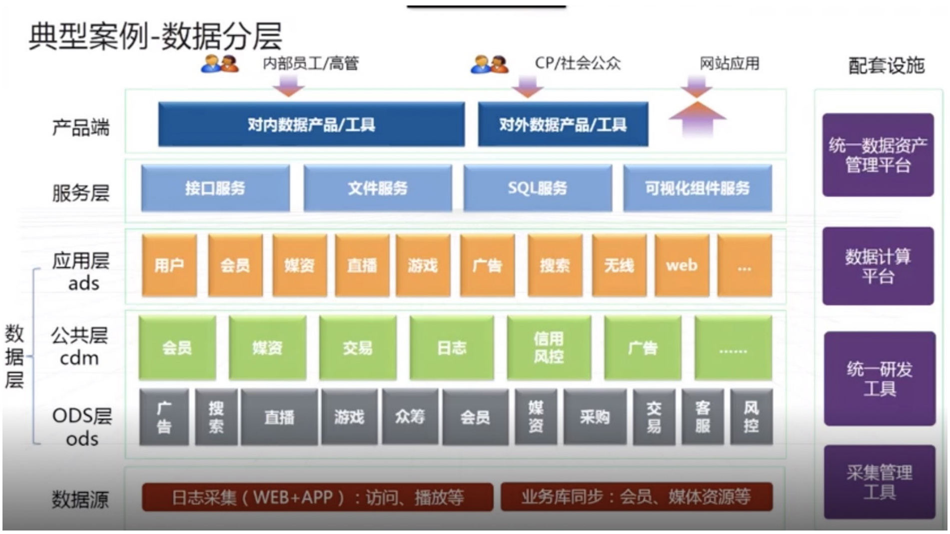
第三个，MaxCompute 快速的扩容能力，比如说突然有一个比较强的业务需求，发现数据跑不动了，计算资源不够，所有的队列都堵死了，这个时候其实可以直接跟运维说一声，帮忙一键扩容，他两秒钟敲一个命令就搞定了。这样的话，所有的资源可以迅速的消化下去。



上面是优酷为什么采用 MaxCompute，下面是在优酷的业务场景下，我们一些典型的方案、应用。这张图实际上是优酷，包括可能现在阿里集团内部一些非常典型的技术架构图。中间可以看到，MaxCompute 在中间核心的位置，左侧主要是一个输入，右侧是一个输出的趋向，绿色的线是一个实时的链路，包括现在我们从整个的

数据源上，比如 DB 也好或者服务器的本地日志 Log 也好，我们通过 TT&Datahub 存储到 MaxCompute 上面做分析。当然现在非常火的 Flink 实时计算，其实是作为一个实时处理的链路。

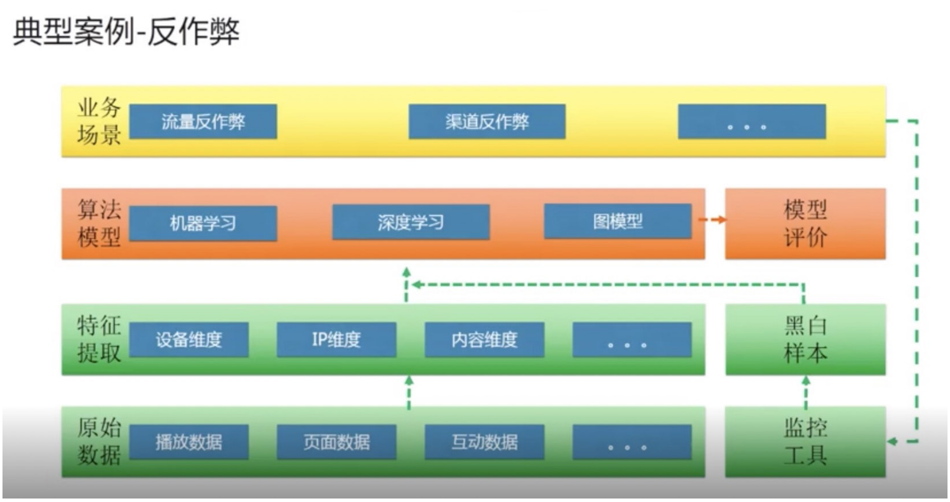
包括 DB 的同步，除了实时的链路，DB 也会去通过按天 / 按小时，把数据同步到 MaxCompute，数据计算结果也可以同步到 Hbase、Mysql 这种 DB 上面。再通过统一的服务层对应用提供服务。下面这个是机器学习 Pai 做的一些算法训练，再把训练的结果通过 OSS 传到一个算法的应用上面去。



这张图可能也是业界比较流行的一个数仓分层的图，因为我们这边是数据中台，所有的数据都是统一从 ods 层 cdm 层，然后 ads 层，去一层一层的往上去做精细，再到最上面，通过接口服务、文件服务、SQL 服务，去提供多样化的服务。再往上面，提供对内的一些数据产品，对高管、对小二，可能还有一些对外的，比如说像优酷的播放数，包括热度这些对应用的数据。

这张图其实就是我们从 Hadoop 迁到 MaxCompute 平台上以来，两个非常经典的案例。我们通过数据中台对不同场景的用户打通，来去赋能到两个不同的场景，提升业务价值。

第二个，可能是内部的，我们通过优酷，还有集团内部的一些 BU 去做换量，我们通过统一的标签去做样本放大，把优酷的量导给其它的 BU，把其它 BU 的量导给优酷，这样去达到一个共赢的效果。



这张图大部分互联网公司不太会涉及到，就是关于反作弊的问题。这个是我们在 MaxCompute 做的一个反作弊的架构，通过原始的数据去提取它的特征，然后再通过算法模型，包括机器学习、深度学习、图模型去支持流量反作弊、渠道反作弊等等。再通过业务场景上反作弊的监控工具，把监控到的作弊信息去打一个黑白样本，再把这个黑白样本跟特征一起来不断的迭代优化算法模型。同时针对算法模型，做一个模型的评价，不断来完善反作弊体系。

最后一点，其实还是跟成本相关，在日常使用中，一定是有小白用户或者一些新来的用户去错误的使用或者不在乎的使用一些资源，比如经常会有一些实习生或者是非技术的同学，如分析师，一个 SQL 消费比较高，这个其实是非常浪费资源，而且可能他一个任务，让其他所有人的任务都在这儿等着排队，实际上我们会去对整个的资源做一个治理。

从节点的粒度上，通过大数据来治理大数据，我们可以算出哪些表产出来之后，多少天没有被读取的，包括它的访问跨度可能没有那么大的，我们会去做下线或者去做治理，有一些业务场景可能并不是非常的重要或者它的时间要求没有那么高，比如一些算法训练，可以去做一些错峰的调度，保证水位不要太高。从 MaxCompute 任务的角度，可以算出哪些任务有数据倾斜、哪些数据可能会有相似计算，哪些任务需要去做 MapJoin，哪些任务需要去做一些裁剪，然后来节省它的 IO。还有哪些任务会去做暴力扫描，扫一个月、扫一年的数据，哪些数据可能会有这样一个数据膨胀，比如说它做了 CUBE 之类的这种复杂计算，一些算法模型的迭代；我们通过数据计算出来的这些迹象，去反推用户，来去提高它的这样一个数据的质量分，来去达到我们降低整个计算资源的目的。

在计算平台的角度，我们也持续的在使用 MaxCompute 推出的一些非常高级的用法，比如我们这边的 HBO、Hash Cluster、Aliorc；

第一个，HBO 就是我们基于一个历史的优化，这样避免了用户不知道怎么调参，我可能为了自己任务快一点，就调一个特别大的参数，这样的话，对集成的资源是非常浪费的。通过这个功能，用户就不用去调参数，集群自动调好，用户就写好自己业务逻辑就好了。

第二个，可能就是最近两年推出的 Hash Cluster，当时在使用 Hadoop 的时候经常会出现，两个大表 Join 的时候计算不出来，这个 Hash Cluster 其实是一个优化的利器。大表跟小表 Join，可以做一些分发，做一些优化。大表跟大表就涉及到一个排序的问题。这个 Hash Cluster，实际上就是提前把数据排好，中间省掉很多计算环节，来达到效率提升的目的。

第三个，Aliorc，在一些固定的场景上面，可以稳定的提升 20% 的计算效率。

第四个，Session。对一些比较小的数据，直接就放到 SSD 或缓存里面，一个节点下游有 100 个叶子场景，是非常友好的，因为低延迟秒出结果。同时，优酷也在使用 Lightning 解决计算加速，这个是在一个计算架构方案上的优化，它是一个 MPP 的架构。

最后一页是存储的优化，因为像一些关键的原始数据或者是需要审计的数据是不能删的，永久不能删的。实际上就会造成我们数据存储的趋势是一直往上不减的，计算会在某一个时间点达到一个平衡。当前用这么多的计算资源，再往后，其实应该也不会再大涨了，比如说旧的业务逻辑下掉了，会换新的业务逻辑，这样会保持在一个相对平稳的波动上面。但是储存，因为它有一些历史的数据是永远不能删的，可能会出现一直在增长，而且是指数级的。所以我们也会持续关注存储的情况，还是通过大数据来治大数据，去看哪些表的访问跨度比较小，来去做生命周期的优化，来去控制它的增速。还有刚才提到的 Aliorc，实际上也是做压缩的。我们会去做一些大字段的拆分，来提高压缩的比例。

OK，这个是优酷在 MaxCompute 中的一些应用场景，感谢大家的聆听。

阿里集团风控大脑关于大数据应用的探索与实践

作者：丁明峰（山蜂） 阿里集团 新零售技术事业群 高级数据技术专家

简介：2019 年双 11 阿里风控保护了约 388 亿消费者的操作行为，同时挡住了约 22 亿次恶意攻击。在首席技术官大数据专享会，阿里巴巴新零售技术事业群高级数据技术专家丁明峰为大家介绍了阿里风控大脑关于大数据应用的探索与实践，即风控领域如何应用大数据来构建风控体系？并详细介绍风控架构以及链路。

本次分享主要围绕以下三个方面：

一、阿里风控大脑整体介绍

二、近线引擎

三、离线引擎

一、阿里风控大脑整体介绍

1. 阿里风控大脑是什么？

阿里的风控主要分为两大块。一块是金融领域，主要业务是支付宝，另一块是非金融领域，如新零售、高德、大文娱等，我们负责的主要是非金融领域。阿里风控大脑的含义较为丰富，可以有不同的解读，但基本上代表了几个方向。首先，阿里风控大脑是“大中台小前台”战略，由于阿里风控管的风险业务很多，领域非常杂，所以允许不同的领域、不同的风控场景可以有自己独特的交互，有自己的 console，但是用到的底层引擎必须是中心化的，由风控引擎做统一计算和处理。第二，阿里风控大脑代表高智能，后续会有深度学习和无监督学习模型大量上线，防控策略及防控方式都会更加智能化。如下图所示，右侧是目前阿里风控覆盖的主要业务和防控的风控场

景，如黑客攻击、消费者保护、商家保护等。左侧是阿里风控 2019 年双 11 的部分数据，保护了约 388 亿消费者的操作行为，同时挡住了约 22 亿次恶意攻击。

2. 典型防控链路

用户通过阿里的 APP 或网站访问阿里的业务会产生大量操作。这些操作进来之后大概会经过如下图所示的七层防控环节。首先会是端上防控，主要在应用层，比如应用的加固，应用的代码混淆等。然后是端上安全策略。第二层是在网络层，在网络层做流量清洗和流量保护。

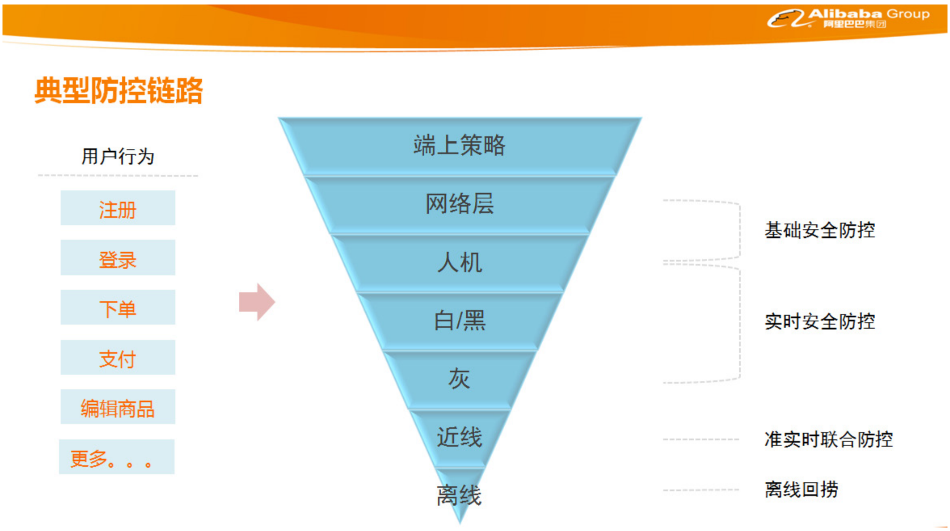
基础安全防控：网络层之后会有人机判断。人机部分在风控领域占比非常大，网络层 + 人机的防控方式和下面几层差异比较大，主要针对基础流量做防控，不会加入具体的业务逻辑，所以称其为基础安全防控。

实施安全防控：人机比较复杂，一部分与流量相关，另一部分偏业务。其中偏业务的部分与下面几层称为业务防控范围。人机之后，在业务防控侧做白 / 黑判断，主要是出于成本考虑。如果能先判定用户行为的白 / 黑，后面则不需要做太多进一步判定，可以节约大量成本。然后是比较复杂的灰的判定，需要从多个维度来识别风险。

准实时联合防控：近线是一种准实时联合性防控，将多种流量或者多种行为放在一起监控。

离线回捞：离线主要是一种离线回捞，针对历史数据做大量回捞。

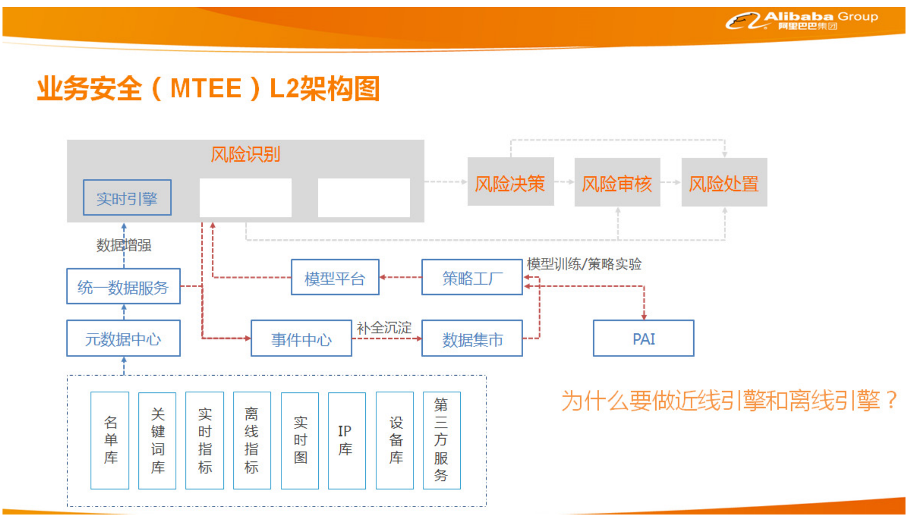
不是所有业务都会走近线或离线，业务按照自己需求自行选择。



3. 业务安全（MTEE）架构

如下图所示，业务侧安全防控可以分成风险识别、风险决策、风险审核、风险处置四大块。风险识别主要是风险行为的判定，当检测到用户的某些行为有风险，如果业务比较简单而且识别准确度很高，那么此行为可以直接流入风险处置做处置。如果识别出的行为没法确定或识别准确率不太高，会将其放到风险审核通过机审或者人审做进一步判定，判定之后才进行处置。还有一些业务非常复杂，可能需要进一步的综合判定，那么会将其放到风险决策。比如一些风险不论在一段时间内触犯多少次，只能对其进行一次处罚，但是它在不同环节或是不同行为可能会被识别多次，即会多次被认为有风险，需要在风险决策中对这种风险进行统一去重、收口。

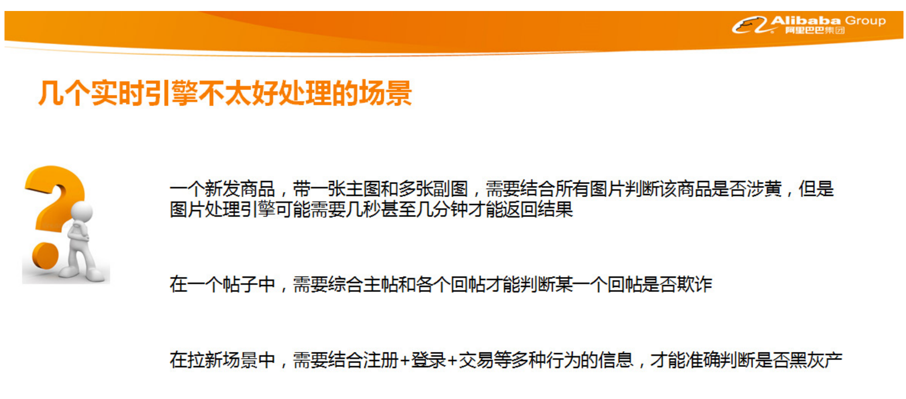
其中最复杂的是风险识别环节。风险识别会用到前端的业务系统，比如淘宝 APP、天猫 APP 传过来的各种业务数据。但是仅仅通过这些业务数据做风险防控是远远不够的，所以阿里会做很多大数据的应用，比如名单库、关键词库、还有很多的指标以及实时图、IP 库等。这些数据会通过元数据中心做统一定义和管理，最终通过统一数据服务来给风险识别做数据增强。另外，通过事件中心、策略工厂、模型平台，构建了策略 / 模型快速实验和上线的过程。事件中心把实时引擎或者近线引擎的数据补全完整后写入 MaxCompute，然后在策略工厂中，会和 PAI 打通，由策略工厂准备数据后，再通过 PAI 做模型训练。最终在策略工厂里面将新的策略、新的模型部署到线上，如此便形成了快速的数据 + 训练 + 上线的链路。



二、近线引擎

1. 几个实时引擎不太好处理的场景

阿里在做近线引擎之前内部讨论了很久，因为近线引擎的边界和实时引擎比较接近，有时很难区分。很多时候在近线引擎里面做的事情在实时引擎里也可以做。那么为什么要做近线引擎？阿里发现有很多场景虽然可以在实时引擎里做，但是需要很多定制化的开发，需要按照场景专门找开发人员去实现。模型大规模推广之后，发现这样的应用场景越来越多，所以希望有更好的方式解决这些问题。比如在商品新发时，需要结合商品图片信息和商品其他信息做综合判断该商品是否涉黄，对于图片信息，大部分公司可能会使用图片识别引擎，但图片识别引擎本身处理能力时快时慢，所以返回时间不一定。这种情况通过实时引擎等待返回是不可能去做的，所以需要做很多个性化的开发去实现整个链路的异步化。还有一些场景比如一个帖子有很多回帖，某些回帖可能是垃圾回帖或带有欺诈行为，大部分情况下是无法通过单个消息或者回帖判断其是否有欺诈行为，而要综合从发帖到回帖各个环节来判断，所以需要把时间跨度很长的很多消息放在一起来处理。这在实时引擎中也不好处理，因为实时引擎本身就是基于事件消息触发的。还有一些非常复杂的业务场景，比如拉新场景，需要结合注册 + 登陆 + 交易等多种行为来判断是否有薅羊毛等黑灰产行为，需要将很多事件放到一起去综合判定，在实时引擎中也不太好做。所以阿里最终决定做近线引擎来对上述问题进行更好的抽象和处理，希望有一种更好的解法来解决这些问题。



2. 近线引擎的定位

基于阿里场景，要求近线引擎至少满足三个要求，如下图所示，第一时效性不能太差，即允许有延时，但不能太久，因为如果延时太久，没有返回风险结果，业务侧就会认为这种行为是正常的，容易造成漏防。第二要支持多事件综合处理的能力，在流计算中称为支持多流的 join 处理。并且需要非常灵活的数据处理能力，大部分算法里面会有很多非常灵活的数据加工，需要算法同学自己实现。第三希望近线引擎能和阿里现有的风控体系无缝融合，因为阿里本身原本的风控体系非常庞大，上下游环节非常多，如果近线引擎孤立在外，可能需要做很多重复造轮子的事情。

3. Why Blink？

流计算的快速发展，让阿里选择了流计算平台作为近线引擎的底层核心。在对比了市面上几个比较受欢迎的流计算平台之后，最终选择了 Blink。选择 Blink 有几点原因，如下图所示。首先 Blink 是阿里内部定制版的 Flink，在公司内部已经搭建了性能非常好的流计算平台，平台开放性、扩展性非常不错，兼容成本也非常低。另外 Blink 有一套完整的 SQL 语义支持，这一点非常重要。因为阿里希望业务方尽量使用 SQL，SQL 使用成本较低，上手速度也会非常快。Blink 团队会持续优化 SQL 性

能，使用 SQL 也可以持续享受到这个福利。

4. 近线引擎功能框架

近线引擎的主要功能是把风控逻辑转换成 Blink 能够执行的任务。近线引擎会把自己需要的数据需求给到事件中心，事件中心通过统一数据服务做数据增强之后流到 Blink 里面做计算。为什么要把数据补全放到前面去做？因为 Blink 是按照任务分别计算，同一个事件或同一个流量可能会在不同的任务里面计算多次，如果把数据增强放到 Blink 里面做，可能会存在多次补全。另外数据补全体量非常大，成本消耗很大，如果放到 Blink 里面做，会对 Blink 造成非常大的压力，并且不好做性能优化。

近线引擎从功能上主要分成四个模块。

业务组件：对风控元素进行封装。在近线风控链路中有很多风控元素，比如事件中心的数据源、对应的下游风控决策或过程中需要用到的模型、算法、策略等。对这些元素进行组件封装，从而使用户使用近线引擎时可以快速使用这些风控元素。

Security-SQL：语法和 Blink SQL 类似，Blink SQL 中会要求写具体的物

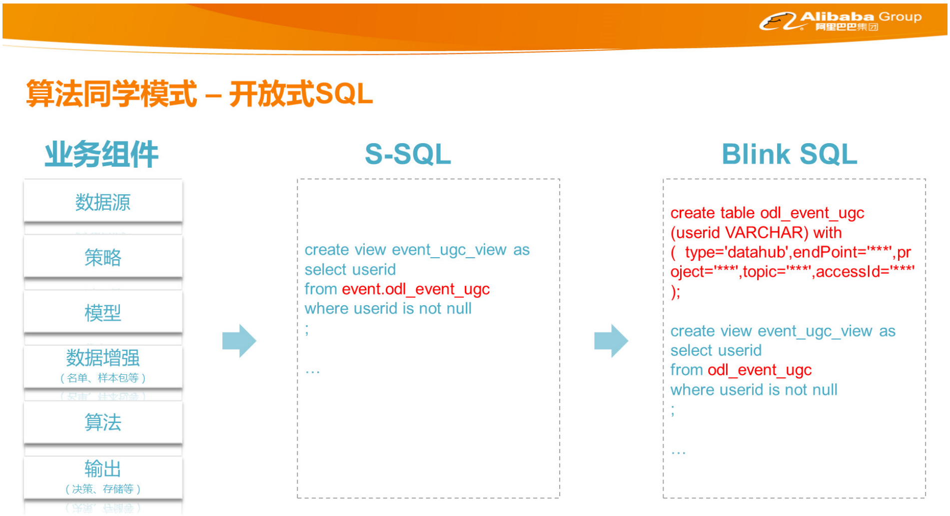
理实现，阿里希望用户不需要关注这些实现细节，而只关注业务逻辑，所以设计了S-SQL。

语法转义：将 S-SQL 翻译成 Blink SQL。

Blink 任务管理：包括任务的上下限、安全生产相关的，做灰度、做测试等。

5. 阿里在近线引擎为同学提供的两种模式

算法同学模式—开放式 SQL：算法同学模式是开放式 SQL。因为算法同学具备较强的数据能力和开发能力，并且经常会针对一些业务场景写个性化很强的算法，如果将其限制的太死反而不方便，所以支持完全用 S-SQL 来写业务逻辑。下图所示案例是从数据源取出一个字段。左侧是对过程中需要用到的业务组件的封装，中间是 S-SQL。可以看到 S-SQL 写法跟 Blink SQL 完全不一样，只需要写 event.odl_event_ugc。event 是数据源的特殊名词，即一个系统保留关键字。用 S-SQL 来写根本不用关注 event 是怎么来的等影响研发效率的信息，因为在系统、业务组件里有一套约定告知 event 从哪里来。



运营同学模式—通用能力：运营同学可能有一定的数据能力，但没法自己去研发算法，但运营同学也希望能用上不同的算法来实现自己的业务需求。算法同学会按照不同业务场景开发一些通用能力，比如音频类，视频类，图片类，或文本类的，有基本的，也有具体业务的。每一个通用能力会转换成一个 Blink 任务。业务同学可以通过交互式的方式配置自己的策略，其中可以引用通用能力用来做风险识别。当业务流量进来，首先进行数据预处理，然后按照引用的通用功能把流量转发到各通用能力对应的任务作相应计算，然后将原始数据和通用数据进行合并，之后再执行自己的策略，并最终将数据分发给下游，比如风险决策系统。整个处理过程拆分的比较细，主要是因为在同一个 Blink 任务里面，如果代码量特别大或者是任务特别长，性能和运维会是非常大的问题。将任务拆的比较细便于管理运维，性能也会更好。

另外，任务之间基本通过两种方式进行数据传递。第一种是 MetaQ，上游任务会通过 MetaQ 分发到下游。第二种是通过 HBase，因为 HBase 的多列加上 HLog 可以很方便地把多条消息整合到一条消息里面，所以数据合并基本是用 HBase 来做。

6. 业务效果

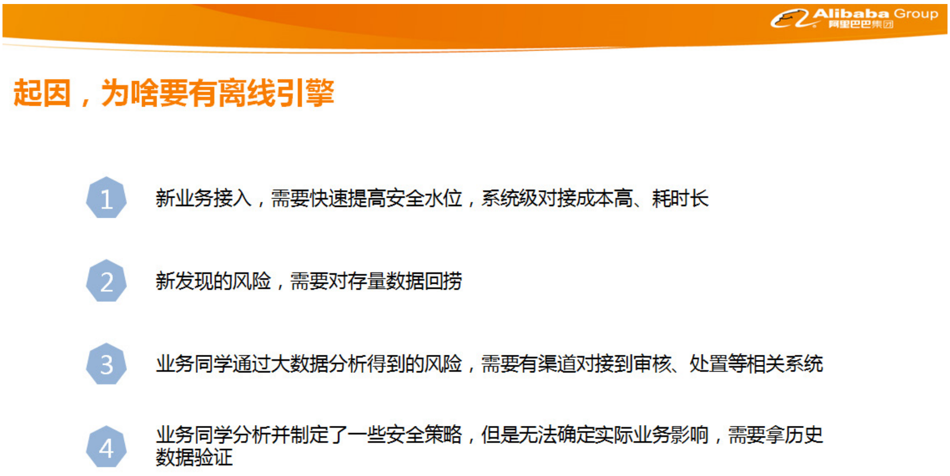
目前近线引擎用了约 2000CU 资源，日均处理事件量约 300 亿，主要覆盖的场景有商品、内容、直播、拉新等多个领域，风险识别在风控领域占比约 10%。相信随着模型和算法的进一步发展，产品的进一步完善，比重也会大幅上升。

三、离线引擎

1. 为何需要离线引擎

离线引擎基本是和近线引擎同一时间考虑的，起初是发现有很多离线数据会批量导入到实时引擎中处理，非常不利于实时引擎的稳定。随着深入探索和研究，发现很多场景确实需要批处理能力进行风险识别。离线引擎起初是为了解决以下几个问题。第一是新业务的接入，阿里集团规模最近几年发展越来越快，覆盖了非常多的业务领

域。大部分新业务的安全水位很比较低，需要接入风控体系。原来会让新业务走实时引擎做对接，对接成本较高，对接速度比较慢。新业务方和安全小二都希望有一种更方便、更快速的对接方式。第二是很多新发现的风险，或在当前时间点漏过的变异风险，在发现之后需要对历史数据进行回捞，需求量很大。第三是很多业务同学在离线做大数据风险之后得到的一些结果，需要有渠道流入到审核或处置等后续环境。第四是业务同学会做策略变更，需要用历史数据来验证其实际影响，否则上线过程会变得非常慢。



2. 离线引擎的功能框架

语义转译 SQL

离线引擎底层主要依赖于 MaxCompute，主要过程是将风险语义转换成 MaxCompute 任务去执行。离线引擎和近线引擎有些地方非常像，比如语义转换和任务管理部分，区别只是近线引擎基于 Blink，离线引擎基于 MaxCompute。

仿真模拟

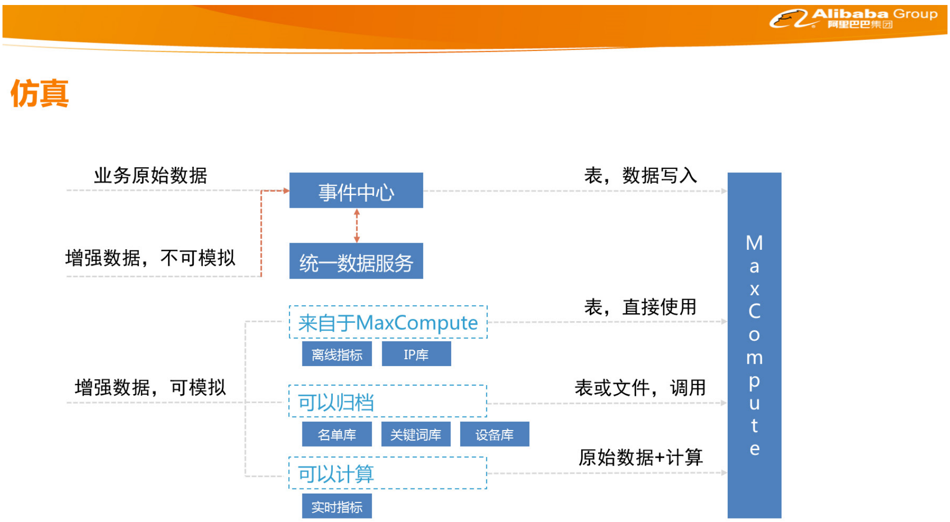
离线引擎的独特之处是需要对历史数据进行全面处理。一个很大的问题是新特征不能通过事件中心对历史数据进行补全，所以做了仿真模拟，即尽可能在离线上复现风控在实时引擎中用到的特征。按照如何去做将仿真分为三类。

业务原始数据：业务原始数据即业务发过来的数据，按照目前策略，业务原始数据会通过事件中心全量写到 MaxCompute 中。事件中心使用 DataHub 中间件，事件数据会通过 DataHub 写到 MaxCompute。这类数据默认全部都有，不需再做过多操作。

不可模拟的增强数据：第二类是不可模拟的增强数据。比如调用了一个第三方的服务，完全不清楚第三方服务的逻辑是什么，只知道第三方服务给出的数据。因为调用的第三方服务比较多，所以不可能逐一去看，这类数据基本暂时是不可模拟的。目前对于这种数据要预先配置在事件中心里面去做补全，其缺点是如果要在新的事件上做补全，已经属于事后的事情了，那么历史的那部分数据是没办法获取的。所以不可模拟的增强数据目前还有缺陷。

可模拟的增强数据：第三类是可模拟的增强数据，按照模拟方式的不同又分为三

种。第一种数据来自 MaxCompute，因为很多数据，如离线指标、IP 库原来就在 MaxCompute 上做计算，计算完成后同步到线上，给线上应用使用，对这种数据只需在做 SQL 翻译时直接采用数据源表即可。第二种是可归档数据。很多数据应用是在自己或周边团队建设的的，如名单库、关键词库等等，非常清楚整个数据逻辑，可以按约定做好数据归档。归档方式多种多样，如直接回流到 MaxCompute 上，或将其转成文件，在 MaxCompute 上读取。归档数据体量不会特别大，数据量最多 TB 级。第三种基本指实时指标，线上几千个实时特征每时每秒产生的数据量都非常大，这些数据全量回流到 MaxCompute 的成本很高。但好的地方在于，实时计算用到的原始数据基本都是实时引擎流出的，而且这些数据事件中心都会接入，所以 MaxCompute 上也都有这些原始数据。而且实时指标的逻辑都维护在系统里面，是非常清楚的，所以可以基于原始数据及指标的计算逻辑，在 MaxCompute 上写一套模拟任务去模拟。阿里写了一套尽可能仿真的仿流式计算的任务模板，结果数据和流计算基本一致，最多会有一分钟或者更小时间窗口的误差。通过这一整套模板，就可以在离线引擎上提供很多与线上一致或接近一致的指标供用户使用。

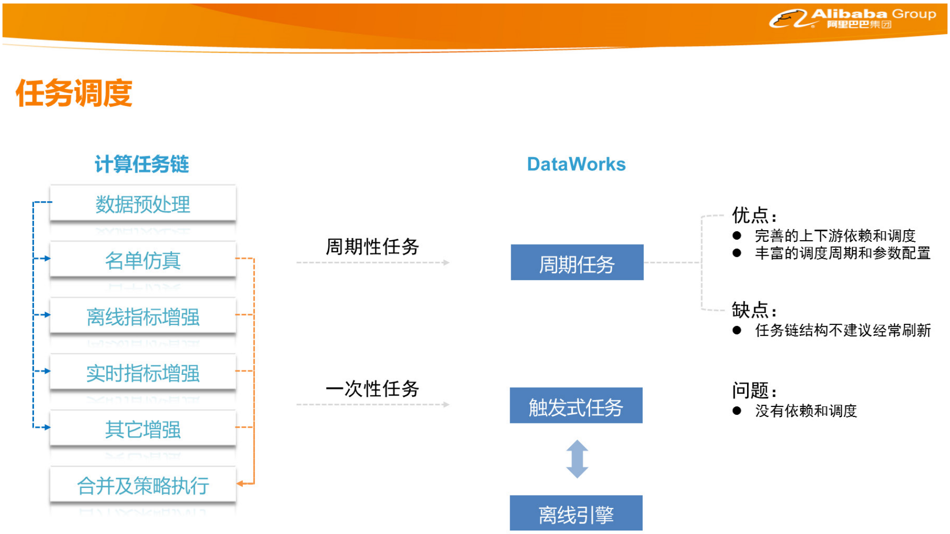


任务调度

Blink 无需太多任务调度，流量来了就处理，但离线批处理需要有任务调度。离线引擎的任务调度很大一部分是用 DataWorks 本身的调度，但也有一些场景没办法使用。目前离线引擎的任务分为两种。

周期性任务：用户需要周期性的对一些增量或者全量的历史数据进行风险识别。周期性任务借助 DataWorks 的周期任务，因为它的周期任务管理已经做得很好，首先有完善的上下游依赖和调度，其次有丰富的调度参数配置，可以通过很多参数来控制调度。DataWorks 周期性任务的缺点是任务结构不建议经常刷新，如果经常刷新任务的上下游结构，可能导致任务调度出问题。比如昨天的任务今天还未调度，如果把调度链路改了，任务就可能有问题甚至报错。但在离线引擎中，为了让一次风控计算任务性能更好，需要将一个任务拆成多个任务，即多个 DataWorks 周期性任务来处理。比如会先用一个任务做预处理，然后多个任务并行做各种数据增强，之后再进行合并，最后做策略执行，如此一来时效性会很好，执行速度会更快。但是周期任务中这种任务链会在策略变更时需要经常去刷新，为了避免刷新带来的问题，现在将增强数据固定分成了几类，比如无论这一次离线任务里是否使用名单，先将名单增强任务生成好，将任务节点永远保留在任务链中，那么无论怎么刷新，最多是刷新其中的逻辑，而不刷新任务结构，从而让离线引擎任务可以随时更新。

一次性任务：需要对历史数据做一次性回捞，不需要跑多次。一次性任务是利用 DataWorks 中的触发式任务。触发式任务最大的问题是只支持单个任务做调度。因为一次性任务时效性很重要，用户做一次回捞不可能等几个小时，所以需要对任务进行更细致的分拆，分拆完成后在离线引擎里面自己实现调度，通过周期性轮询任务状态，自己完成任务依赖、任务调度等工作。



四、总结

阿里目前有三个引擎，实时引擎、近线引擎和离线引擎，其好处是用户能做的事情变得更多，能力变得更强，坏处是引擎增多，概念增多，用户理解和使用成本会变得更高。所以阿里在引擎设计之初坚持的原则是用同一套元数据，即引擎的元数据都是一样的，可以在最大层面上避免用户对引擎理解的不一致。其次，在很长时间甚至到现在，一直在做的事情都是尽量让引擎用到的是同一套数据。未来希望所有引擎有同一套风控语言，例如 S-SQL 或比 S-SQL 更成熟、更抽象的一种语言。这种语言可用于解释风控场景中的各种语义。如果有这样一套风控语言，风控用户对风险的描述、风险场景的落地会更加直观清楚。

可闭环、可沉淀、可持续的企业级数据赋能体系——友盟云数据中台产品实践

作者：林鸣晖　友盟 + 首席产品官

简介：对于所有企业来说，数据决定了基于算力、算法等能做出哪些场景和应用。在本次首席技术官大数据专享会上，友盟 + 首席产品官林鸣晖围绕业务数据化，数据资产化、资产应用化、应用价值化构建属于企业的可闭环、可沉淀、可持续的数据赋能体系进行分享，基于智能数据采集（U-SDC），用户数据平台（U-CDP），数据开放平台（U-DOP）探讨如何建立企业的数据银行。

本次分享主要围绕以下两个方面：

一、构建可闭环、可沉淀、可持续的企业级数据赋能体系的背景

二、开发者数据银行

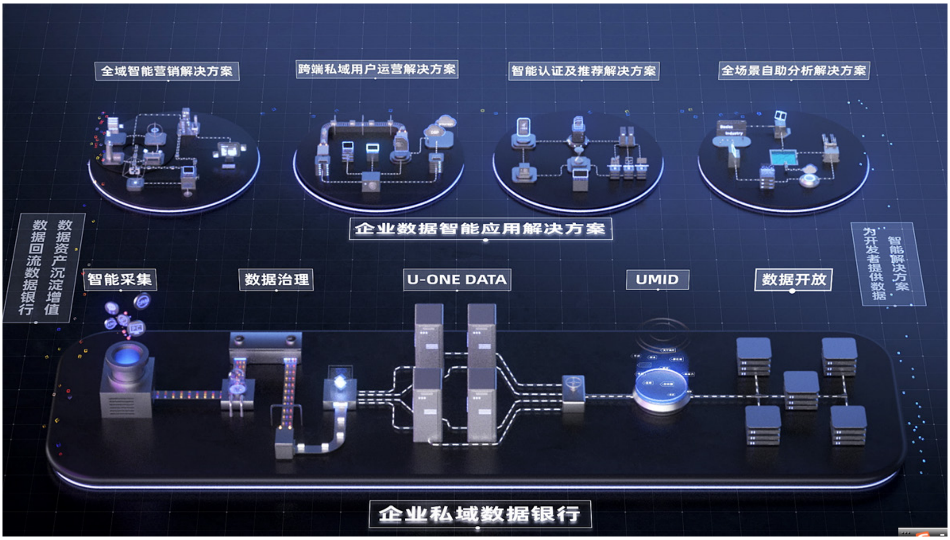
一、构建可闭环、可沉淀、可持续的企业级数据赋能体系的背景

1. 数据“四化”

如何让属于企业自己的不同触点的数据快速形成一个闭环，沉淀串联这些零散的数据能够快速应用去赋能业务？这涉及到四个关键词，一是业务数据化，企业所有触点是否为真，是否被打通。第二是数据资产化，能否可以像管理资产一样很好地管理数据。第三是资产应用化，企业的资产能否有效应用？如何借助数据资产赋能业务，最后是应用价值化。所有的应用最终一定是为增长、为获客而服务，必须要有价值。在这背后最重要的是场景必须可闭环，数据必须可沉淀，最终数据中台、数据能源才是可持续的。

2. 构建可闭环、可沉淀的数据赋能体系的意义与价值

下图展示了一套可闭环、可沉淀、可持续的企业级数据赋能体系是如何构建的。下图友盟 + 会推出一个面向企业的数据银行。数据银行和业务是一种什么样的协作关系？开发者数据银行会基于云基础设施，如 MaxComput 等，不断帮助企业采集各种场景、触点的数据，做相应的数据治理、提纯、模型加工、形成各种应用服务，基于 UMID 打通能力，多账号归一，多端归一，支持不同的终端数据打通（移动客户端、服务端、客户端不同的平台），帮助开发者完成全场景、全触点的数据资产沉淀及应用的管理。



关于跨端用户运营会涉及两个问题，首先，公司每一次在媒体外投的数据是否已经回流？回流后是否能够对数据进行第二次应用？第二，通过你的营销是否将用户沉淀至用户池，跨端的用户是否有效运营起来了？其实除了营销，企业会有很多用户触点，如头条号、微博、抖音号等，用户资产的数据必须打通后才能发挥真正的价值，如果你在做你的搜索推荐，那么除了先进的模型算法之外，你的公司是否有数据底座，是否在收集回流归一各个触点的用户行为数据，并喂养给你的搜索引擎让它越来越智能；比如：此前投过广告的数据下次进行搜索时，你就应该推荐客户之前看过相

关广告的内容。

二、开发者数据银行

每一家公司都需要构建属于自己的数据银行。比如在阿里巴巴的生态体系内，阿里在双 11 当天有上百万商家卖货，很多品牌商家都在阿里构建数据银行。同样，友盟 + 在数据智能服务领域已深耕九年，凭借服务百万家互联网企业的经验，面向开发者推出开发者数据银行，与 MaxCompute 形成一套核心解决方案服务用户。数据银行需要解决几个问题：第一，数据银行解决数据资产的管理和应用的问题，可以用采、建、管、用四个字来表达。首先是业务数据化和数据资产化，如何采集数据，并快速将端的数据形成数据资产。其次是资产应用，形成多种消息的推送，营销的拉新，包括 App 的推送，各种运营推荐，都是在数据银行上能够提供的服务。

数据银行包括三类产品，从三个角度帮助用户解决问题。如下图所示，第一个产品是智能数据采集（U-SDC），第二个用户数据平台（U-CDP），帮助企业沉淀数据资产，高效服务业务部门、运营团队、市场等团队。第三个是数据开放平台（U-DOP），将采集到的数据通过友盟云之上与业务数据进行融合、分析，更全面的洞察用户，更场景化的应用数据。

1. 智能数据采集（U-SDC）

无论 AI 或者智能引擎产品，本质是数据生产和采集。采集是数据质量的根本，数据采集的效率质量和效益都至关重要。数据采集工作需要关注是否全面掌控了公司的数据埋点？是否清楚某个场景应如何埋点？埋点后会产生什么样的数据？所埋的点是否正确有效？埋点是个长期运行的动作，需要不断验证埋点是否健康，最后一点回归到根本性的问题，如果埋点是错的，那么叠加的 AI 智能等所有内容也都会是错的。

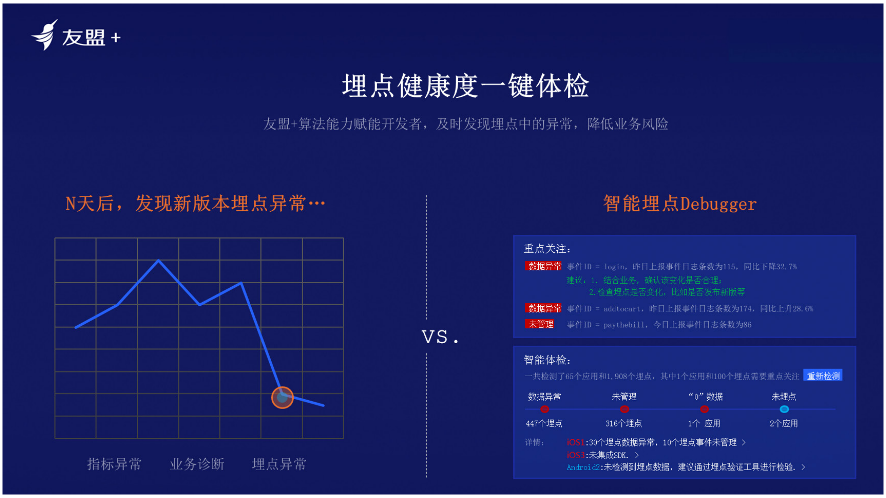
管理埋点：埋点在大数据领域属于脏活累活，很多人不愿意做。常见的情况往往到了产品上线，需要使用数据的时候开始催促埋点。所以一家公司的埋点是否有人搞清楚？是否清楚这么多的埋点中哪些埋点正确，哪些异常？很多企业是不清楚的，这是一个残酷的现实。这是一个非常实际的问题，如果公司长期不清楚自己的埋点问题，便是在错误的数据上长期持续经营业务，越走越错。

埋点智能方案推荐：某家视频行业领域的公司的有两个团队，分别负责直播不同频道的业务，两个团队都会定义一些公司的埋点规范。但是数据规范性在两个团队不一致，如视频播放开始，A 团队定义埋点全局参数叫 Play，代表播放开始事件，B 团队将其定义为 Start。两个团队并不知道两个数据定义都不一致。案例中的问题看似不严重，但后续会发现公司数据不可持续，此时不论利用什么工具都不能解决问题。对于公司数据的管理一定要基于对业务场景的深刻理解，对业务场景进行标准、规范的定义。友盟 + 通过更多标准化的场景，包括为不同行业提供标准的埋点方案推荐来解决用户问题。友盟 + 聚合了非常多比较优秀的企业的实践，告诉用户如何埋点，埋点后能够解决哪些场景问题，同时会提供各种各样埋点智能推荐，针对技术团队沉淀公司基于场景的埋点解决方案的知识图谱。

智能埋点与智能验证：开发做埋点是通过 SDK 代码，上报数据，后台打印日志。但并不意味数据上报则完成了埋点。如开发者直接将一个启动的日志埋在登录页面，突然有一天发现登陆数高于页面访问数近一倍。原因是该点同时被埋到了退出页面的加载进程中。即开发者错将一个点埋到两个位置。友盟 + 希望能够提供各种智能验证工具，比如当埋点上报时，会为开发者提供一个服务，如果埋点命名为“启动”，会有一系列的智能检测该埋点上报时所在的页面截图是否为正确的业务场景位置。智能埋点及其验证测试是非常重要的，友盟 + 会通过视觉切图计算验证埋点的正确性，为技术团队大幅减轻工作成本和压力。

埋点健康度一键体检：当埋点全部完成，公司要做埋点健康度的验证，检查埋点是否符合规范，是否有异常点。埋点健康度是公司数据采集准确性的底座保证。数据团队和做客户端的开发团队经常会因为埋点问题产生矛盾。数据团队觉得数据有问题时一般归责为埋点问题，开发团队也会认为是数据团队配合问题。埋点的 KPI 就是先让埋点可视化，看到是由谁埋了哪个点，运行情况是否出现问题，是否按照规范埋点。如果埋点的规范度没有达到一定程度，团队是否应该承担责任？因此需要从管理角度、从组织层面以及产品能力层面解决公司埋点和采集的核心问题。

数据银行采集平台（U-SDC）会重点解决以上几个核心问题，使用户埋点可见、可控、可管，为用户埋点推荐合适的优秀方案，使用户埋点能够智能调试和验证，大幅降低埋点采集的成本，从而最终达成数据质量的根本性提升，使最终保存的数据资产有价值有质量。



2. 用户数据平台（U-CDP）

数据采集之后，最重要的是解决用户资产问题。首先，用户资产管理一定要解决的问题是可信和归一。数据做了很多触点，每个请求在访问 APP，其中很多是作弊的或受欺诈的流量，如何保证设备是可信的？基于 UMID 打通能力，多账号归一，多端归一，支持不同的终端数据打通（移动客户端、服务端、客户端不同的平台）的流转换关系洞察，归一完成后形成自动的标签生产库，使得私域的标签生产保持高效，并且能够赋能到业务团队，快速做标签、洞察、圈人，并且最终形成对客户的运营动作。

是否清楚自己的用户资产：用户数据平台（U-CDP）支持多源数据如何在很短时间一键接入平台，如移动客户端、服务端、客户端等源头。U-CDP 保证可信识别和多端归一，通过全域数据识别，帮助用户做数据归一和提纯，过滤垃圾，反作弊。识别打通后最终形成用户资产可视化，清楚公司触点来源，了解多少私域用户被沉淀下来。清楚上述问题再分析需要建哪些触点，需要增强哪些触点。最终沉淀下来的才真正是自己的私域数据资产。沉淀私域用户资产的一个前提是可运营，若不可运营、不可见，那么数据是无用的。

用户的标签管理库，配置即生产：业务团队总是对技术团队不满意，当运营团队要做一个活动，需要按照业务场景准备物料，准备活动的页面，还要再按照规则圈到一群想要触达的内存，然后对其进行运营。上述需求需要先和产品经理提需求，产品经理再去和算法、技术团队沟通然后写 PRD，再等待几天将活动开发上线。往往流程特别长，完全无法满足运营团队快速迭代、快速试错、快速运营客户的诉求。而运营团队的需求并没有那么复杂，如运营团队只是想给最近 30 天访问过 APP、看过小程序，同时这两天被广告命中的那部分人一个红包，但是很多企业面临技术排期。

运营团队感到不满，技术团队也缺少成就感，因为每天的工作基本是跑 SQL 等繁琐零碎的事情。企业需要思考的问题是如何高效解决上述生产场景。**友盟 + 希望数据银行提供预置私域标签的生产，不需要技团队做过多事，只需要将埋点做好。所有产品要去支持运营，能够在平台上面快速配置，快速生产，赋能业务团队，预置私域标签，配置即生产。此外，友盟 + 数据银行会提供一个不同的能力，既全域标签。私域标签只对客户进行圈选和洞察，友盟 + 会额外加持全域标签，告知不同用户的兴趣方向，从更多的维度去洞察和圈选用户。友盟 + 未来计划与其它企业联合建立一个标签实验室，贡献双方不同的数据，通过融合计算得出更好的标签效果以服务不同企业。**

预置分析模型，自定义报告结构：运营团队只需要预置分析模型，做交、并、差的各种组合，做各种洞察，洞察完成后保存自己的人群包，即可快速复用到每一个业务的运营和活动之中。自定义私域人群细分体系埋点完成后，在友盟云上采用 MaxCompute 数据仓库的方案，自动汇聚成一个人在多个端每一天的行为，自动形成用户的档案序列，自动配置完成。只要埋点是正确的，运营团队马上可以完成私域人群细分。友盟 + 希望把上述的轻量方案应用到解决实际生产中的各种各样支撑的问题。

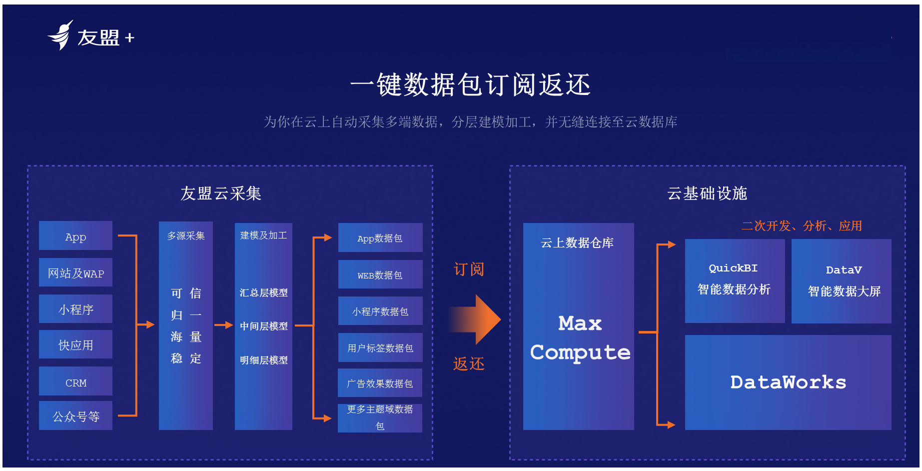
多种组合模式，找到想找的人：如某装修建材公司，有一个 Web 网站，起初是通过 Web 网站以及 QQ 与客户联络。后面该公司又发展了 APP 和小程序的团队。客户可能同时出现在三处，问题时数据不互通，并且组织是分开运营的。其实本质问题是能否在 APP 端快速发现小程序的客户，再去客户端做投放，运营和回流。友盟 + 结合多种模式，无需等排期，帮助运营能找到合适的人。

多种通道触达、互动效果追踪：U-CDP 支持多种通道，无论是短信、EDM、还是 APP 的消息等都可以对接，所有的运营效果都可以实时可见。友盟 +CDP 本质上是和技术团队一起赋能业务团队，解决业务团队的效率问题，并增强业务团队运营能力，并沉淀下来用户数据资产。

3. 数据开放平台（U-DOP）

友盟 + 采集数据后将采集的数据与客户的数据进行融合，通过与 MaxCompute 进行云端的无缝对接，支持更大力度的开放返还。

一键数据包订阅返还：如下图所示，友盟云采集帮助客户快速采集移动客户端、服务端、客户端不同的平台等数据。如果客户自行加工单一的上述事情，处理时间会非常就且最终质量难以保证。基于 UMID 打通能力，多账号归一，多端归一，支持不同的终端数据打通，友盟 + 帮助客户做好加工，生成不同的数据包，只要客户使用 SDK，数据包自动生成，自动将数据传送到 MaxCompute 中。然后可以借助 DataWorks、DataV、QuickBI 与客户的数据做数据融合，极大地降低成本。客户使用的不再是原始数据，而是经过友盟 + 加工处理过的数据。之后，用户就可以专注于业务产品的开发，业务场景的赋能，把精力放到业务创新而非原始的加工工作上。



友盟 + 和 MaxCompute 云上数据仓库无缝对接，不仅可以提升处理性能，还可以增强使用的简易和便利性。友盟 + 会为用户预置好所有模型包、模型表，并且打通数据，数据即开即用。

QuickBI 智能数据分析展现：下图是一位客户做的友盟 + 和 QuickBI 智能数据分析展现。数据融合、返还后，结合 MaxCompute+QuickBI，做业务人员自助分析，**拖降式自助分析**和在线表格的分析，与原来其它的割裂数据放到一起做大量工作，由此分析师团队的效率获得了极大的提升。

总结：无论企业有多么强大的容器、数据库和算法，或者要做多么智能的场景应用，一定要先回到四个关键词：第一是业务数据化，管理好采集和数据质量。第二是数据资产化，让管理层清楚的看到用户资产的具体情况，涉及多少个端，多少个触点，每天产生的数据，沉淀下多少用户。第三是资产应用化，沉淀下来的数据能够快速变成哪些应用去服务业务团队，使业务团队认为技术、数据是在促进帮助业务团队做创新，而不是业务团队等待资源去赋能。其中最根本的一套理念是必须让所有的触点和业务行为的环节能够产生场景和数据的闭环，让场景和闭环能够沉淀数据资产，只有这样才能使一个企业的数据中台可持续，数据赋能可持续，数据能源才会越用越厚，越用越好。

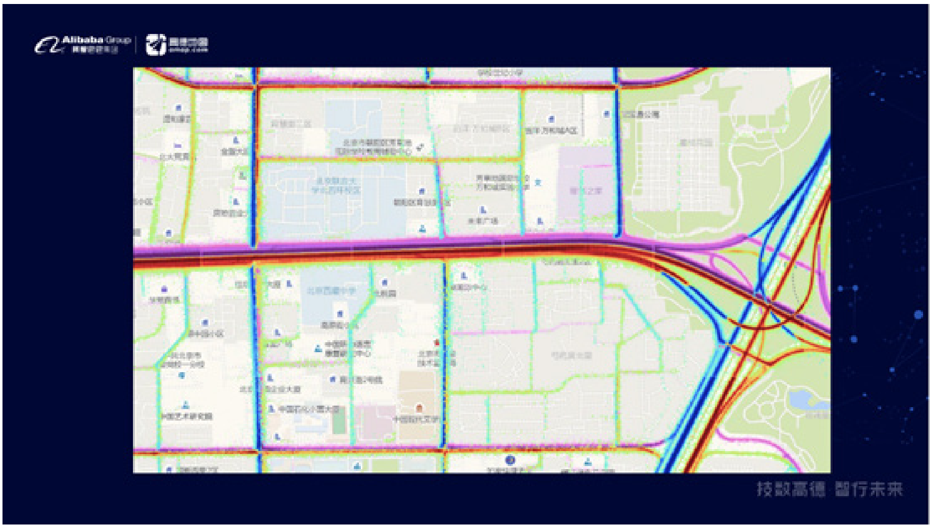
MaxCompute 在高德大数据上的应用

作者：苗翌辰（喵鹿） 阿里集团 高德事业群 数据技术专家

摘要：2019 年 1 月 18 日，由阿里巴巴 MaxCompute 开发者社区和阿里云栖社区联合主办的“阿里云栖开发者沙龙大数据技术专场”走近北京联合大学，本次技术沙龙上，高德数据技术专家苗翌辰为大家分享了高德如何应用 MaxCompute 来管理数据架构，开发易用、高效以及弹性的高德应用，为用户提供更优质的出行服务。

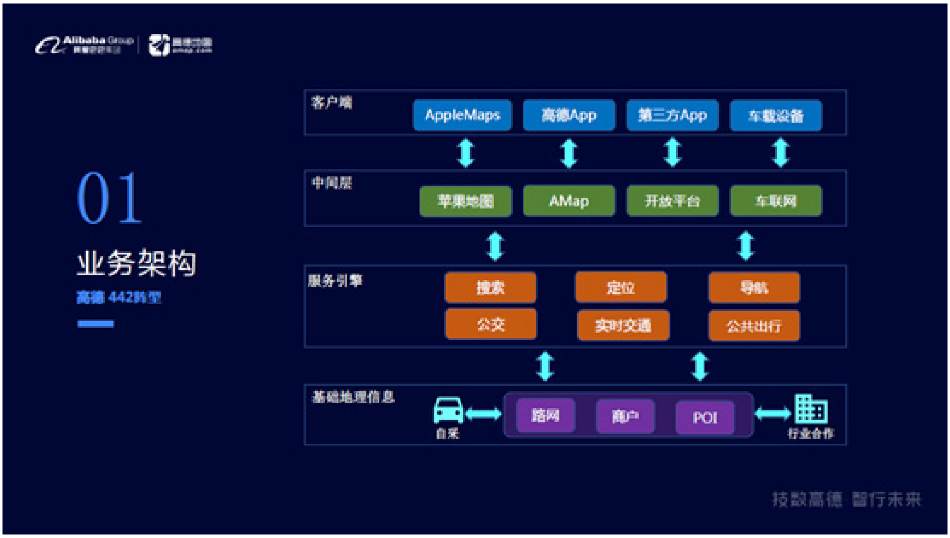
一、高德的业务和数据

地图描绘需要很多支撑数据，包括现实中的道路信息、路形以及路况等。下面的轨迹热力图展示了高德地图显示的北京联合大学的周边路况，描绘了点、线和面三种信息。通过地图信息和轨迹数据叠加形成区域热力。其中，不同颜色的轨迹展示了该区域一天内不同时间段的路况。



下面展示了高德的一些业务场景。第一个场景是大家日常使用的高德 APP。高德地图是苹果中国的战略合作伙伴，第二个场景展示了高德为苹果提供的出行服务。高德向整个互联网行业开放了其生态能力，第三个场景是高德为 APP 应用开放者提供的位置服务接口，目前使用该接口进行开发的移动应用包括手机淘宝、今日头条和小米运动等。另外，第四个场景是高德为车载设备提供的完善的位置服务方案。

高德地图的业务架构可以用“442 阵型”来形容，即分为客户端、中间层、服务引擎以及基础地理信息等 4 层，同时包含 AppleMaps、高德 App、第三方 App 以及车载设备等 4 个业务入口。另外，数字“2”是指高德地图依赖于两个数据源，即自采数据、行业合作数据组成的基础地理信息和轨迹数据、行车数据等服务引擎产生的数据。“442 阵型”的业务架构给高德的发展带来了质的飞跃。



下图是高德总裁刘振飞先生正在庆祝高德十一 DAU 突破一个亿的历史时刻。十一期间，高德为全网用户提供导航的总里程超过 135 亿公里，相当于在太阳和地球之间往返 45 次。高德提供海量服务的背后是高德强大的大数据计算能力、超过数千台的高德集群节点和承载超过百 PB 数据的集群存储容量。

二、如何管好数据

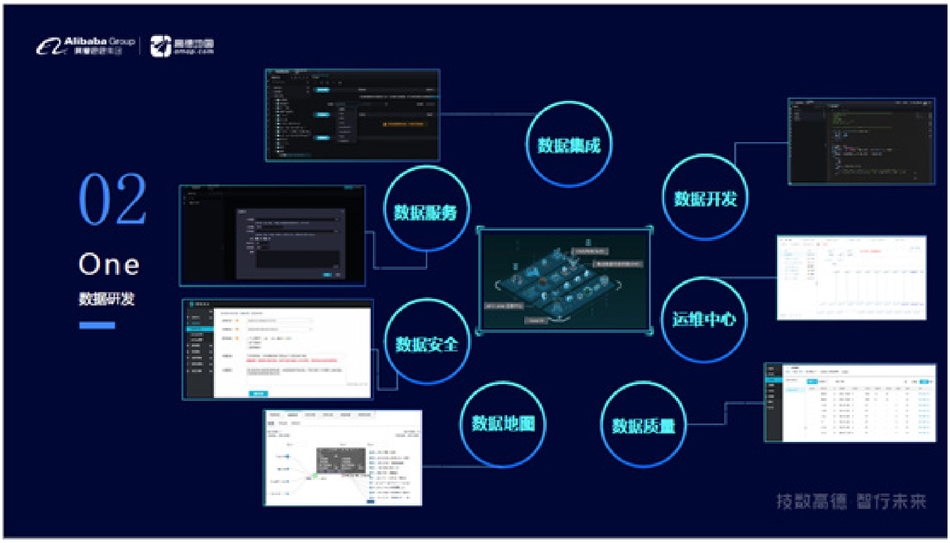
SPA 架构

高德的数据架构称为“SPA 架构”。“S”指代 Source，即数据源层，收容了高德内部所有的位置、地图和图像数据。“P”指代 Platform，即数据平台，提供了数据仓库、数据适配和数据挖掘能力来支撑上层的数据应用层，也就是 Application(“A”)。在“SPA 架构”中，高德最关注获取数据的权限，即要求所有数据操作都应该符合安全规范。此外，高德还要求所有部门都明确其开发目标，并且使用统一的平台工具进行开发。

数据研发

数据研发的全链路过程包括数据集成、数据开发、运维中心、数据质量、数据地图、数据安全以及数据服务等。高德对数据平台的要求不仅仅是以上全链路都 All in One，还希望都能以可视化的方式进行用户交互，以提高开发效率。以运维中心为例，希望所使用的工具能够将调度节点可视化，并方便进行不同时间粒度的任务依

赖。同时，我们还希望拥有可视化的数据地图用于管理元数据信息，方便上下游即时查看。MaxCompute 正是符合高德数据业务诉求的给力产品。



MaxCompute 平台特点

高德使用的 MaxCompute 平台具有以下三个特点：

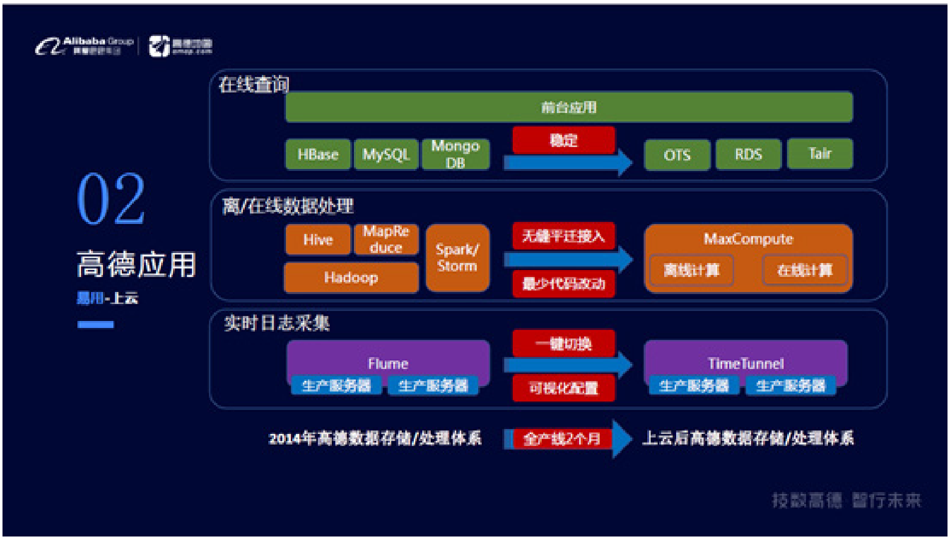
第一，易用性，具有零学习成本和完善的 IDE 等优势。

第二，效率，高德内部迄今为止最大的公共项目“魔方”就是运用阿里云和 MaxCompute 实现的。

第三，弹性，高德于十一期间的流量远远超出想象。

易用 – 上云

2014 年，高德的数据架构依赖 Flume 进行数据采集，依赖一个仅含几百台机器的 hadoop 集群和 Hive 等软件实现数据处理。2014 年 9 月份高德内部提出“上云”，即将数据迁移到阿里云，使非流程化的作业得到流程化的管理。与其他复杂的数据迁移工作相比，2014 年高德实现了“一键”上云，将源数据的同步从 Flume 切换成 TimeTunnel，后续再可配置化地切换数据。此外，迁移还伴随着代码修改，2014 年高德“上云”仅修改了非常少的代码，比如修改老版本 M2 中的接口等。上方的数据存储层将数据介质替换成 OTS 等云端产品，以支持更加稳定的前台应用。高德将所有集群数据都迁移到“云上”仅花了两个月时间。



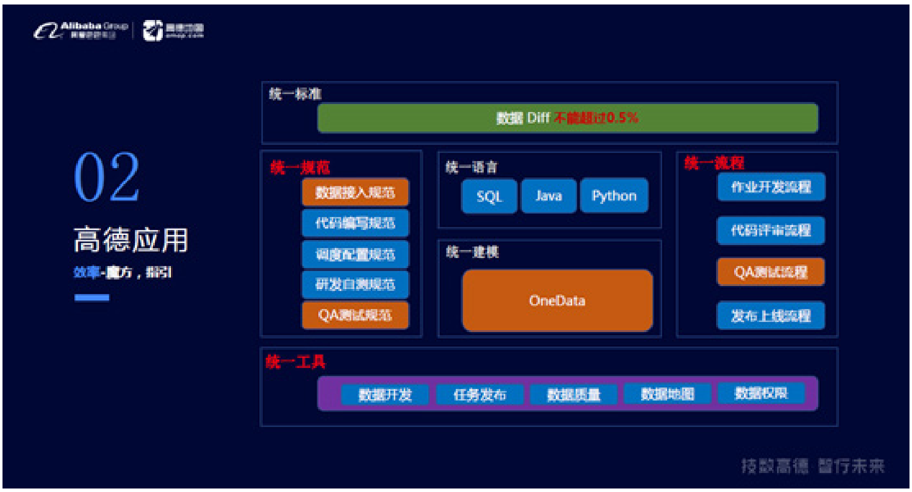
“上云”为高德带来的收益不可估量。图 1 展示了“上云”后由云端管理所有代码；图 2 展示了一键式运维管理；图 3 展示了可计量的计算资源管理，量化地显示各个任务的资源使用情况；图 4 展示了流程化的可视化安全审批操作。从 2014 年“上云”到如今 2018 年，高德经历了飞速的发展，同时也暴露出了一些问题。

效率 – 魔方

烟囱过多是数据仓库中比较麻烦的问题，高德同样存在该问题。数据使用者可能需要花费一个月寻找数据所在部门、数据的相关产品负责人以及研发人员以索要数据。2017 年高德盘点数据仓库时发现，高德内部存在二十个数据仓库项目，并且各个数据仓库间的数据冗余度高达百分之三十，严重影响了团队工作效率。此外，高德数据仓库还存在高时延缺点，核心数据无法保证每天“7 点产出”。基于以上两个问题，高德发起了“魔方”项目，将二十个仓库合并成一个以实现全集团的数据治理。

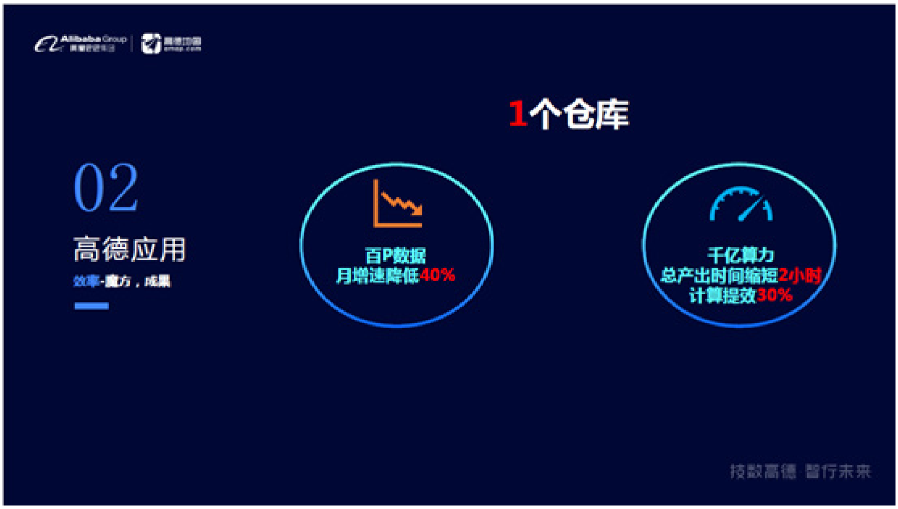
显而易见，要实现全集团的数据治理项目存在严峻的挑战。首先，数据量非常大，“魔方”项目要求实现百 PB 级数据的全域数据治理。其次，参与人员众多，“魔方”项目涉及到高德全产线的所有数据开发人员，项目团队超过百人。最后，排期紧，为了使数据架构升级不影响正常业务，高德要求“魔方”项目的主体开发工作应在两个半月内完成。此外，数据迁移工作在越短时间内完成对企业的收益就越大，因此高德要求“魔方”项目应在尽量短的时间内完成。应对这几个挑战的主体思路就是引入高效率的研发工具，在规范化的流程中实现协同开发，提高团队的工作效率。

为此，高德首先统一了工具平台，引入了 MaxCompute，下图蓝色部分皆为 MaxCompute 为我们带来的业务收益。在数百人的团队中统一规范无疑是困难的，而 MaxCompute 提供了代码编写规范、调度配置规范以及研发自测规范等规范化模块。其中，代码编写规范模块使用 SQL Scan 工具自动化地检查代码是否符合规范，调度配置规范模块提供了完善的用户手册及各种模板辅助开发人员完成配置。统一流程要求实现定制化地管理数据开发流程，包括研发测试、开发自测、调度测试、QA 测试以及最后的上线部署等流程。此外，统一建模和语言、统一数据核定标准也非常重要。



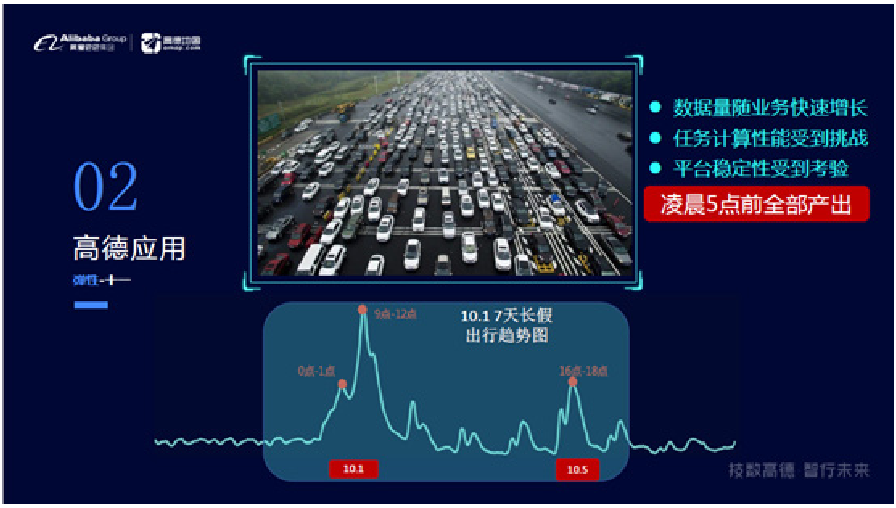
阿里云提供了一些优秀工具以构建规范化流程。第一，提供了数据血缘可视化工具，帮助数据开发团队及时地跟踪源数据、数据的上游和下游等信息。第二，提供了开发 / 测试流程并行的能力，以支持完善的协同开发和高效运转的工作。第三，提供了代码云端版本管理工具，允许实时查看代码变更、代码管理状态并支持回滚。第四，提供了一键数据探查工具，允许数据开发人员通过简单的配置探查海量数据的字段空值率，有效值率，表重复率等信息，极大地提高了数据开发人员的工作效率。

在规范化的流程以及众多效率工具的帮助下，高德在规定时间内完成了“魔方”项目开发，得到了一致好评。高德最终统一了数据仓库，将内部所有百 P 级数据的月增速降低了 40%，同时数据计算效率提升了 30%。即使在 2018 年十一的流量轰炸时期，高德仍然实现了核心数据的“5 点产出”目标（5 点到 7 点需完成核心数据计算任务）。



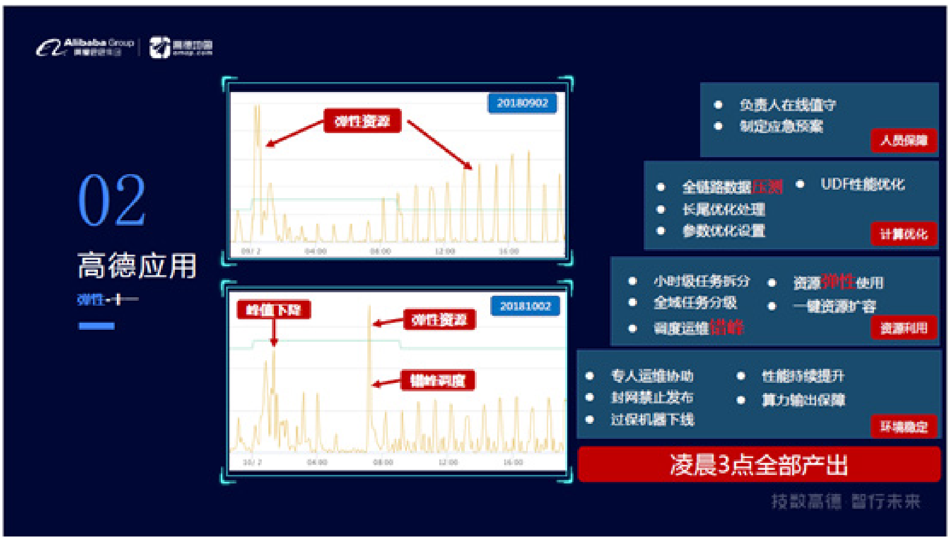
弹性 – 十一

2018 年十一期间，高德的数据处理量随业务快速增长，数据计算任务的性能和平台的稳定性都受到了极大的考验。



数据血缘可视化工具允许数据开发人员可视化地查看系统资源配置，下图展示了高德在 2018 年 9 月 2 日的系统实际使用计算水位，其中，蓝线是系统配额水位，黄

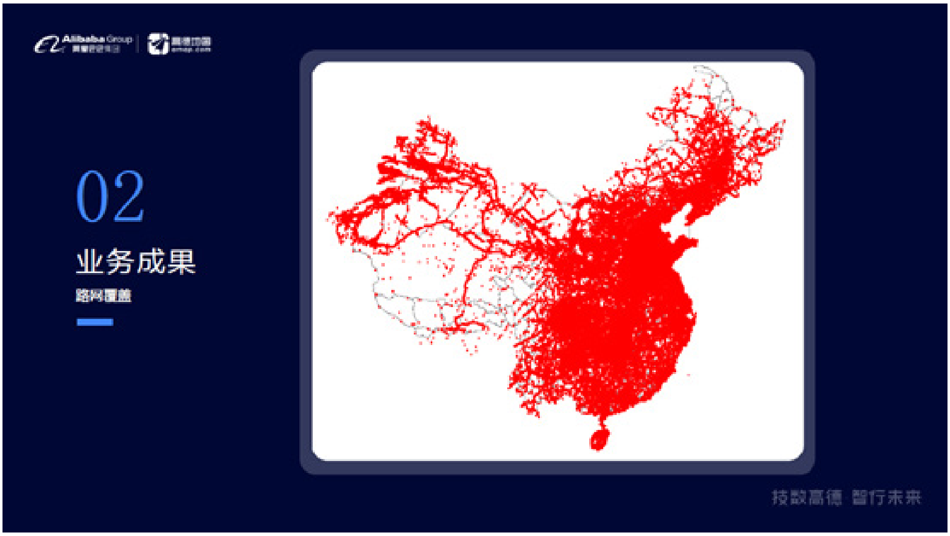
线是系统的实际计算水位。阿里云提供的弹性计算能力允许在一定弹性数据量范围内保证系统资源的正常计算和输出。此外，阿里云还提供了稳定的计算环境，保证计算任务高效地运转，同时避免资源竞争问题。另外，为了更好地利用系统计算资源，高德团队提出了“提高蓝线、打散黄线”方案，申请扩大集群资源配额提升算力空间，通过调度错峰打散实际资源水位。在扩容方面，MaxCompute 为高德带来了一键资源扩容能力，使得集群扩容在小时级别的时间内完成。最后，高德还实现了计算优化，提供了人员在线值守等保障。下图同时展示了高德在 2018 年 10 月 2 日的系统计算水位，蓝线代表的“系统配额水位”远高于 9 月 2 日的，说明集群扩容工作已顺利完成。同时，黄色代表的“实际资源配额水位”已完全被蓝线 cover，更好地保障了资源计算任务。此外，黄色高峰被明显打散，一些重要非核心数据被错峰调度到 7 点，说明计算资源的错峰调度工作也已顺利完成。阿里云提供的一键运维调度工具能够保证系统方便的进行调度错峰，节省人力。MaxCompute 为高德带来的弹性能力使得高德于 2018 年 10 月 2 日实现了核心数据“3 点产出”的骄人成果。



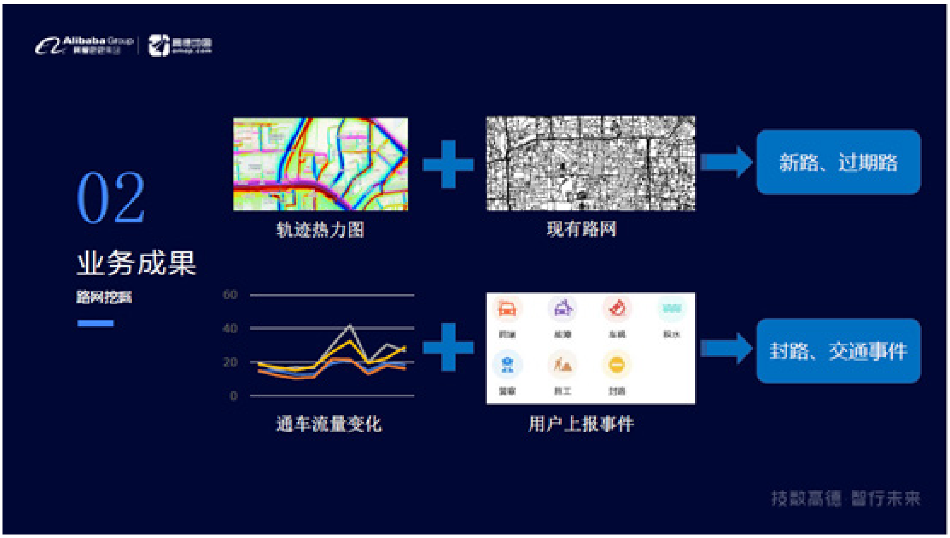
业务成果

下图展示了中国的路网覆盖图，华东、华北和华南地区基本实现了道路全网覆

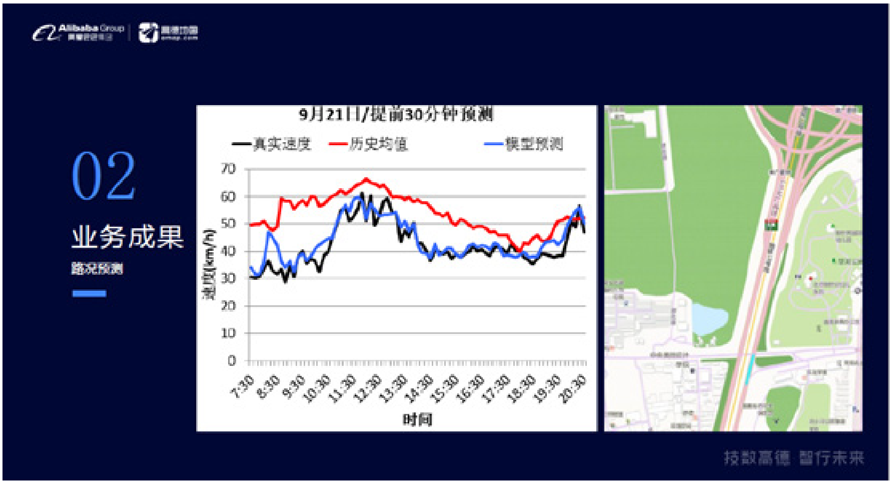
盖，西部等偏远地区的道路交通还不够发达，许多道路仍在建设。路网覆盖对高德而言非常重要，高德需要用尽可能少的成本自动地发掘新路和过期路。



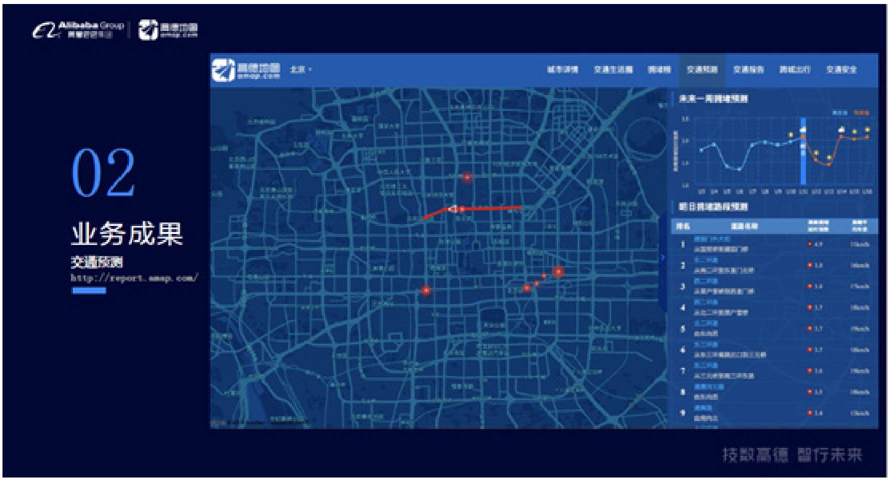
高德将轨迹数据资产和地图建设能力结合起来构建轨迹热力图，辅以现有路网和数据挖掘算法以自动化地发现新路和过期路。此外，高德还结合区域通车流量和该区域相关的用户上报事件来动态地发现封路、交通事件，更好地实现路网挖掘。



路况预测是高德的另一个重要业务，即实时预测道路的通车状况、道路是否拥堵等。左图展示了高德对右图路段从早到晚的平均车速预测，红线表示数据累计得到的历史均值，蓝线表示模型预测值，黑线表示真实数据值。蓝线和黑线基本重合，有力地说明了高德应用的数据挖掘能力和统一数据仓库建设取得的成果。



此外，高德还面向全网用户开放了一个城市级数据产品，允许用户随时查看城市的道路拥堵状况和城市拥堵指数等相关数据，该产品可在 http://report.amap.com/ 页面访问。下图显示了该产品给出的北京北二环某路段一周内的拥堵状况。



三、未来展望

技“数”高德表达了高德的未来规划。作为一家数据公司，高德致力于丰富数据源、优化 SPA 数据结构，得到更强大的数据平台，为用户提供更优秀的出现工具。从企业责任出发，高德希望能通过自身努力，帮助社会智能化地解决交通拥堵问题。从应用生态出发，高德希望能够帮助生态建立更加场景化的 LBS 服务，共同打造更专业的位置服务应用。

高德致力于在未来继续联手阿里云，实现“连接真实世界，让出行更美好!”的行业愿景。

MaxCompute 在阿里妈妈数据字化营销解决方案上的典型应用

作者：梁时木（载思） 阿里集团 阿里妈妈事业群 技术专家

摘要：大数据计算服务（MaxCompute）是一种快速、完全托管的 GB/TB/PB 级数据仓库解决方案，目前已在阿里巴巴内部得到大规模应用。来自阿里妈妈基础平台大规模数据处理技术专家向大家分享了 MaxCompute 在阿里妈妈数据字化营销解决方案上的典型应用经验。首先介绍了广告数据流，分析了 MaxCompute 是如何解决广告的问题；然后通过阿里妈妈内部的应用经典场景来介绍其如何使用 MaxCompute；最后介绍了 MaxCompute 提供的高级配套能力以及在计算和存储方面的优化。

本次的分享主要分为两部分：

一、题外话：该部分主要介绍在听完之前的分享后，嘉宾的几点个人感受 。

二、广告数据流介绍：该部分主要是对没有中间商赚差价的广告数据流的基本介

绍以及为什么 MaxCompute 能解决广告的问题。

三、典型应用场景：该部分主要通过阿里妈妈内部的应用经典场景来介绍其如何使用 MaxCompute，包括数据分层体系、报表和 BI、搜索引擎索引构建和算法实验。

四、高级功能和优化：该部分主要就阿里妈妈在五六年的 MaxCompute 内部使用和管理过程中的相关高级功能和优化进行公开分享，包括 MaxCompute 提供的高级配套能力以及在计算和存储方面的优化。

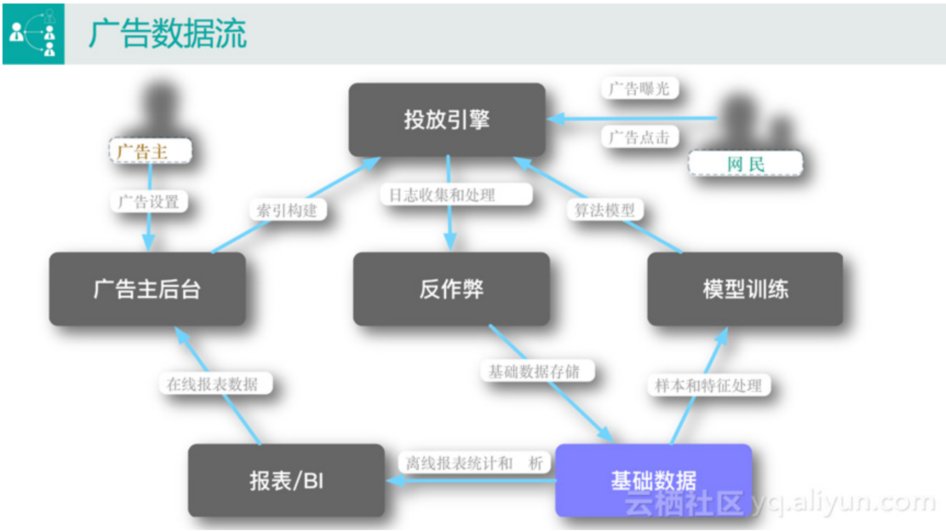
一、题外话

开始之前先聊一点题外话，是我在听完刚才的分享之后的几点感受。其实我从 2013 年就开始在内部项目中使用 MaxCompute, 阿里妈妈作为第一批登月用户，踩了很多坑。听了刚才的分享，我的**第一感受**是，MaxCompute 有好多功能我们都没有关注过，作为一个资深用户，很多功能都不知道听起来好像有点不太合格，但转念一想这可能是 MaxCompute 作为一个平台、一个生态解决方案应该具备的一种能力。就是说它让一个终端用户只需要看到你所关注的东西，有一些你不太关注的，而对于整个生态链上又必须有的东西它可能潜移默化的帮你做了，在我看来这其实是一种很强的能力。在听完分享后的**另一个感受**就是我还是蛮庆幸在阿里这边做这些事情的，一些中小公司在基础设施服务这块很需要一些第三方平台的支持，因为你会发现它们如果从头搭建的话，成本会特别高，而 MaxCompute、阿里包括钉钉这样的产品，撇开商业化这个因素，它们更多的其实是帮助我们推动互联网技术的进步，帮助我们做一些基础上的事情，从而让我们有更多的精力去关注与自己业务相关的事情。

二、广告数据流

聊完题外话，下面正式开始我的分享。首先向大家介绍一下广告数据流。下图是广告数据流的一个简化版本，因为我们今天的主题是大数据相关的计算，所以我们站在整个数据流的角度简化介绍传统广告，如搜索广告和定向广告。在图中可以很清楚

的看到，从角色上来说，比较受关注的是广告主和网民两个角色。



广告主后台

一般来讲，广告主首先会在广告主后台进行相关的广告设置，举个最简单的例子，一个广告主要做一次广告投放，首先需要设置一些基本信息，如将广告投给“来自北京”的“男性”，或者当用户搜索“连衣裙”的时候将广告投给他(她)。

投放引擎

广告主做完设置之后，会用到最重要的一个服务，即投放引擎。它的作用是当广大网民在百度或淘宝键入搜索信息后想其投放相关的广告。比如淘宝搜索“连衣裙”，结果页里面显示的一部分是自然搜索结果，还有一部分是带有“广告”图标的结果，这些结果就是通过投放引擎投放给广大网民的。

与网民最相关的两个行为是广告曝光和广告点击。当网民在使用某个平台进行搜索的时候，比如淘宝，投放引擎里面会用到两个很关键的模块来决定将什么广告投放给网民，即索引和算法：

- **索引构建**。当用户键入某个搜索关键词后，如淘宝搜索“连衣裙”，会有上万条商品满足该关键词，不可能将所有的商品都投放给用户。 这个时候的解决方案是首先从广告库中查询有多少广告买了该商品，这个数据需要广告主在后台设置，即哪些广告可以投放给该关键词。
- **算法模型**。假如有一万个广告候选集，相关的评分服务会根据一系列的特征和后台训练出来的模型对所有的广告进行打分排序，最终按照一定的规则展现给用户。

反作弊

上面介绍的是网民看到广告的过程，相应的在系统后台会产生一些日志，最典型的是广告的曝光和点击日志，会用到反作弊的模块。因为很多时候网民的行为不一定人操作的，有些是通过 API 或工具等恶意的手段进行竞争，反作弊模块的作用就是对这些行为进行判断并给出相应惩罚决策，如扣费。

基础数据

经过反作弊模块的数据我们认为是可信赖的，会将其存储在基础数据平台中，此处的基础数据平台是一个抽象的概念，最终其实就是 MaxCompute。

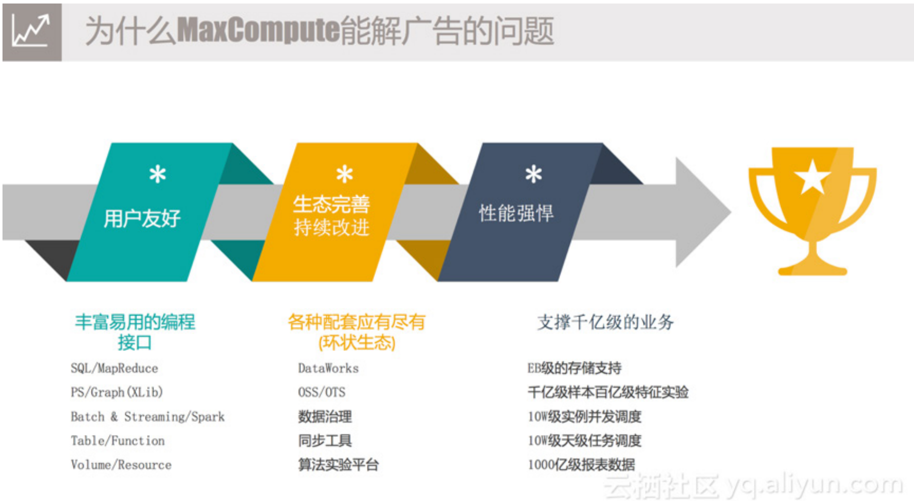
报表 /BI

有了数据之后，最常见的两个应用场景是做报表 /BI 分析和模型训练。

- 报表 /BI 分析。主要有两种情况，一种情况是广告主在设置了广告投放之后，想要了解广告的投放效果，此时会有相应的统计数据到广告主后台；另一种情况下 BI 运营的人员也会对这些数据进行分析，如某些广告位的表现情况。
- 模型训练。前面已经提到，投放引擎第一步只能拿到一个很大的广告候选集，如何筛选用户预估分最高的广告并投放给用户是模型训练这个模块要做的事情。该模块需要用已经存储的原始基础数据去跑各种各样的模型，从最传统的逻辑回归到现在的深度学习，跑完的数据再推送到投放引擎，这个时候就可以

实现广告的在线评分功能。

以上是广告的简要数据流的介绍，接下来分享为什么我们这么久以来一直坚持用 MaxCompute。总结一下主要有三点，即数据友好、生态完善持续改进和性能强悍。



1) **用户友好**。从刚才的数据流介绍中，或多或少能看到一点，我们的应用场景有很多，比如反作弊的场景，再比如报表和 BI 分析的场景，针对此 MaxCompute 提供各种各样的计算能力和丰富易用的编程接口。最传统的是 SQL 的表达支持，如果 SQL 表达的语义不满足要求，加 UDF 仍然解决不了问题，MaxCompute 还支持用户自己写一个 MapReduce，提供原始数据用户可以自己去加工；还支持用户做一些图计算像 Deep Learning；另外 MaxCompute 本身也是支持 Batch 和 Streaming 两种功能，包括之前提到的 Spark Streaming；还有一点也是我比较喜欢的，在 Hadoop 生态圈中，大家其实更多的看到的是 HDFS 文件路径，那在 MaxCompute 中，我们更多的看到的是一堆一堆的表，表对用户来讲有 Schema，比空洞的文件更容易理解一点，针对这些表 MaxCompute 提供 API 层面的操作支持，另外也提供相应的 Function，包括 UDF、UDTF 类型的支持；同时 MaxCompute 还提供半结构化类型的支持，如 Volume，支持用户操作相

关的 Resource。上述介绍的功能为用户的开发提供了便利。

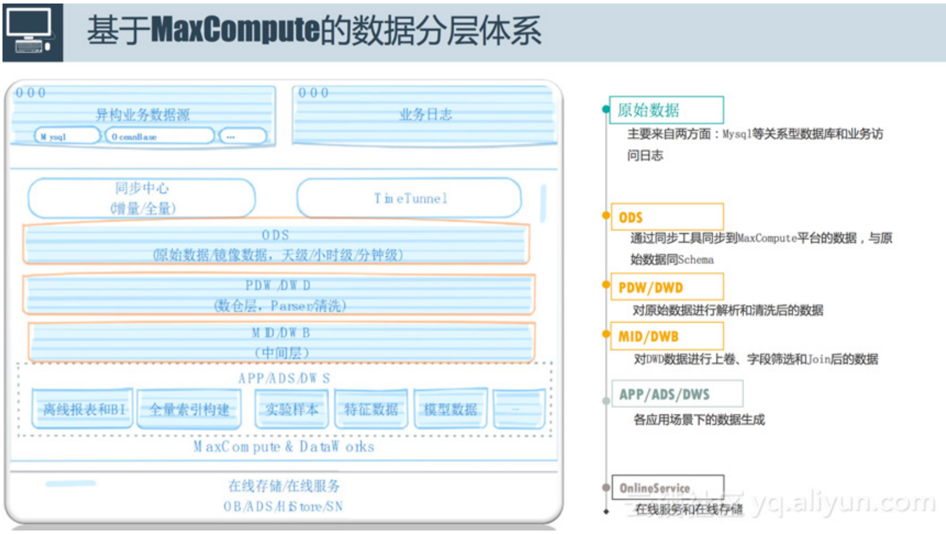
2) **生态完善持续改进**。MaxCompute 是一个平台，一个生态系统。在体验过 MaxCompute 整套系统之后，我们发现其可以应用到我们开发、运维管理的整个过程中。从最开始数据产出之后，如果要加载到 MaxCompute 平台中，可以通过“同步工具”来完成；数据同步之后，如果想要做数据处理，可以通过“DataWorks”跑一些的简单的模型来做数据分析和处理；复杂的数据处理可以通过“算法实验平台”来完成，目前支持 TensorFlow 上的一些功能；数据处理完后，传统的做法是只看数据是否正确，但这对于系统管理人员来讲是远远不够的，还需要看结果好不好，是否有优化的空间，以保证投入产出比，比如传统的离线任务，分配资源的方式是基于 plan 的模式，用户需要预先预估一个 instance 需要多少 CPU 和 Memory，但是会存在两个明显问题，一个是依靠经验的估计是不准确的，另一个是现在的数据量是在不停变化的，无法很好地估计。针对这个需求，“数据治理”会给用户相应的反馈。

3) **性能强悍**。阿里妈妈作为业界数字化营销的厂商来说，数据量非常大。目前使用 MaxCompute 已经可以完成 EB 级别的数据存储；在具体的场景中，可以完成千亿级样本百亿级特征的训练实验；跑一个 MapReduce 或 SQL 的 Job，MaxCompute 可以实现十万级实例的并发调度，后台远远超过十万实例的并发度；阿里妈妈一个 BU，目前一天之内跑在 MaxCompute 的 Job 数已经达到十万级别；最后是我们的报表数据，这其实也是最常见的一个场景，目前我们在 MaxCompute 的报表数据已经到千亿级别。

三、几个典型的应用场景

介绍完为什么使用 MaxCompute 之后，再给大家分享一下阿里妈妈的几个典型的应用场景中是如何使用 MaxCompute 的。

数据分层



前面介绍了广告数据流，我们针对 MaxCompute 也对数据进行了划分（如上图所示），主要划分为六层 。

1) 第一层是原始数据层，原始数据的来源一般有两个，一个是我们的业务数据库，比图 MySQL 或 Hbase, 另一个是我们的业务访问日志，如刚才提到的广告的曝光和点击日志，这些数据是放在我们的服务器上面的。

2) 第二层是 ODS 层，即通过同步工具同步到 MaxCompute 平台的数据，与原始数据同 Schema。原始数据要做离线处理的时候（包括 Streaming 处理），我们内部使用同步中心平台进行全量和增量同步，同时也会使用 TimeTunnel 进行整个服务器日志的采集。最终同步到 MaxCompute 平台的数据与原始数据是同 Schema 的，但是它能以天级、小时级、分钟级实时或准实时的将数据同步到离线平台里面。

3) 第三层是 PDW/DWD。有了同步的数据后，大家知道，数据的格式是千奇百怪的，以日志为例，我们线上回流的日志是遵循一定的协议的，想要把数据真正用起来还需要经过一系列的操作。第一步会进行数据清洗，包括上面

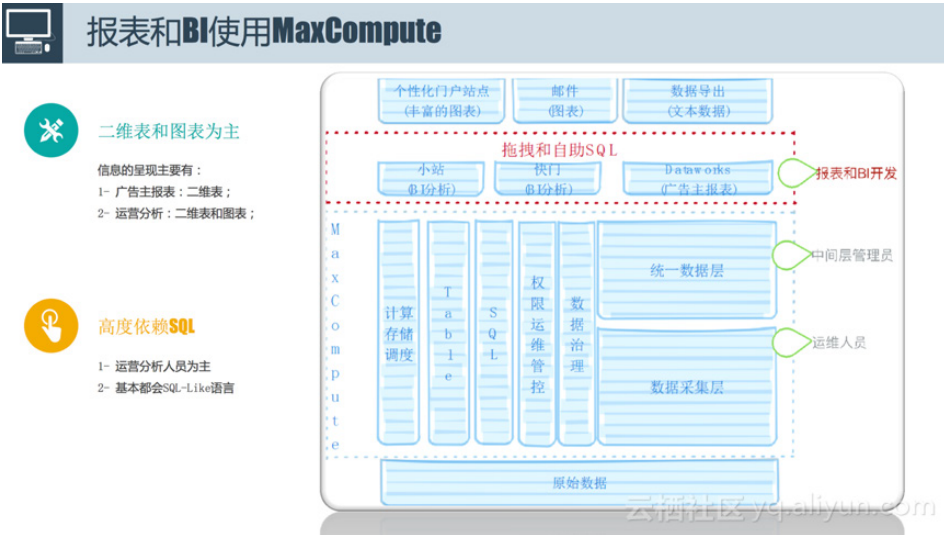
提到的反作弊也是一种清洗的方式；然后会对数据进行简单的拆解，将其拆解成可以理解的字段。

4) 第四层是中间层 MID/DWB。数据量过大的情况之下，比如阿里妈妈一天产出的业务数据高达几十亿，这个数据量根本无法实现直接处理分析，所以我们的做法是使用中间层，对 DWD 数据进行上卷、字段筛选和 Join，后续业务的应用基本上是基于中间层来做的。

5) 第五层是各种应用场景生成的数据层 APP/ADS/DWS。具体的场景包括离线报表和 BI、全量索引构建、模型训练，后面会从这三个方面的的场景来具体介绍以下如何使用 MaxCompute。

6) 最后是在线服务和在线数据存储层。

报表和 BI



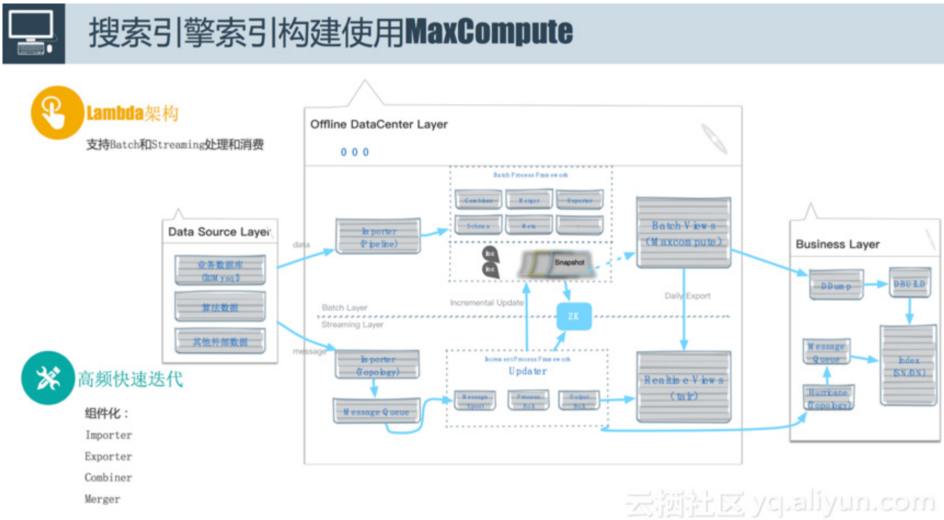
首先介绍一下报表和 BI 是怎么使用 MaxCompute 的。对于一般的用户来说，我们只需要了解两部分内容，包含什么 Table，用 SQL 怎么处理它们。报表和 BI 具备以下两个特点：

1）**二维表和图表为主**。对于广告主来讲，信息的呈现主要通过二维表来完成，通过过滤排序就能看到想要了解的结果；而对于运营人员来讲，除了二维表之外，可能还需要一些图表的具体分析。我们会提供这样一种能力，这种能力就是，比如想要给广告主来看的话，提供数据导出功能，将数据直接导到线上，供广告主在后台直接查看效果；其次在部门内部发送支持邮件；再次我们提供类似小站的功能，即个性化门户站点，后面会通过简单的 demo 进行展示。

2）**高度 SQL**。上述介绍的所有功能都是高度依赖 SQL 的，大部分情况下是不需要做一些 Java 开发的，也不需要去写太多的 UDF，用户在报表和 BI 中只需要去写 SQL，有些甚至只需要拖拽几下就可以得到想要的结果。

下图是报表和 BI 使用 MaxCompute 的 demo 截图。用户输入简单 SQL 表达之后，通过简单的预处理就能看到想要的数据，这个数据不仅可以在系统中查看，还可以通过邮件的方式发送，或者推送的线上进行展示。

索引构建



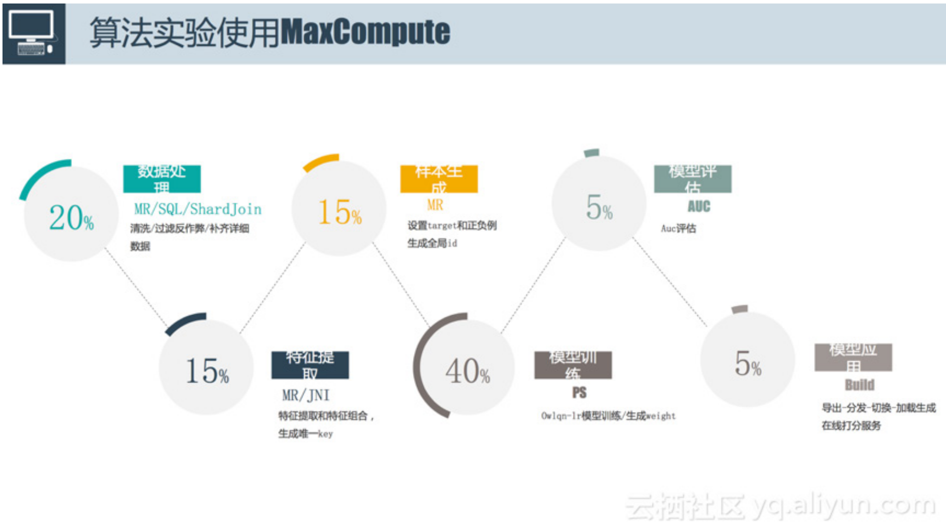
广告主在后台更改投放设置之后，一旦数据量达到百万、千万甚至上亿级别的时候，需要针对在线查询做专有的引擎服务。阿里妈妈广告搜索引擎索引构建使用 MaxCompute 如上图所示，使用的是 Lambda 架构，支持离线和在线，可以使用 Batch 和 Streaming 处理和消费。使用该架构的背景是当时阿里妈妈在做索引更新的时候，每天伴随着各种各样的实验来查看效果，常常会加很多字段，而且这种情况下并行的需求很高，所以我们对系统的要求是必须支持高频的快速迭代，当时我们定的目标是加一个字段要在半天或者一天之内搞定，并将结果推上线，同时要支持多人同时做这个事情，为了实现该需求，我们当时也做了一些类似于组件化的工作。

对于整个索引构建服务，由于时间关系在此只展示业务层，业务处理过程中需要面对各种异构数据源，图左侧数据源层（Data Source Layer），如业务数据（来自 MySQL）、算法数据和其他外部数据，最终将其沉淀到业务层（Business Layer）引擎的索引中，使其支持各种各样的查询。数据从数据源层到业务层需要经过离线数据中心层（Offline DataCenter Layer），分为上下两部分，上半部分是批量层（Batch Layer），下半部分是 Streaming 层（Streaming Layer）。数据源的接入方式有两种，

一种是全量的方式，意思是将 MaxCompute 上面的一张表直接拖拽过来，然后跑一个离线的索引；但是还有一种情况，比如说广告主做了一次改价，这种更改需要快速地反映到索引中，否则索引中一直存放的是旧信息，将会造成广告主的投诉，因此除了全量流之外，还提供增量流，以将用户的更改实施反馈到索引中。

- 离线部分。我们提供一个类似同步工具的服务，叫做 Importer，它是基于 MaxCompute 来实现的，大部分功能是跑在 MaxCompute 上的，因为这里面我们进行了组件化，需要进行一系列的类似于数据 Combine、Merge 的操作，还涉及到源数据的 Schema 和数据的多版本管理。离线数据存入 ODPS 中，通过 Maxcompute 的 Batch views 来查看。
- 在线部分。简单来讲，比如拿到一条 MySQL 增量，通过解析将其直接流入消息队列中，然后通过相应的平台包括 Storm、Spark 以及 MaxCompute 的 Streaming 等，利用和离线部分类似的组件跑索引。接下来通过 Realtime views 可以查到最新的数据，目前通过 tair 来实现。实时部分的数据每隔一定时间进行 Merge，就会形成多版本的数据。它的作用有两个，一个是将这些数据直接批量往在线部分去灌，尤其是在线上数据出问题、走增量流程很慢的时候；另一个是在做离线索引构建的时候，为了避免索引膨胀的问题，需要定期做一次离线全量，为了保证数据实时更新，需要有一条增量流在此期间往全量部分注入数据，为了避免因为服务宕机导致的效率低下，我们提供了多个版本增量数据的保存。

算法实验



接下来介绍一下算法实验使用 MaxCompute 的场景。不仅仅是算法实验，包括我们每天往线上推我们的性能模型的时候，都是下图这套流程。整个流程的输入是线上日志，比如哪些用户浏览和点击了哪些广告，输出是对用户的浏览和点击分析后抽取的特征进行在线评分。中间大致可以抽象为六个步骤：

1) **数据处理**。数据处理除了前面提到的清洗和过滤反作弊之外，做的最简单的是将多份数据合并成一份数据，这里面除了用到 MapReduce 和 SQL 之外，还用到了 ShardJoin，是阿里妈妈和 MaxCompute 合作，为了应对在离线数据进行 Join 的过程中，两边数据都特别大时效率低的问题而开发的。原理很简单，就是将右表拆成很多小块，使用独立 RPC 服务去查。数据处理在整个过程中的时间占比约为 20%。

2) **特征提取**。经过第一步之后输出的结果是一整个不加处理的 PV 表，包含一系列的属性字段，然后在此基础上进行特征提取，常用的是跑一个 MapReduce，最重要的是有 JNI 的操作，实现特征提取和特征组合，生成唯一的 key。比如我想要把 UserId 和 Price 联合算出一个新的特征。原始

的特征可能只有几百个，但经过交叉、笛卡尔积等操作之后，特征可能会达到几百亿，这个时候前面所提到的 MaxCompute 支持千亿级别样本、百亿级别的计算能力便得到很好地发挥，这对于调度包括整个计算框架具有极其重要的意义。特征提取在整个过程中的时间占比约为 15%。

3）**样本生成**。一条样本出来后一般需要设置 target 和正负例，针对每一个特征会生成一个全局 id，最后进行序列化。之所以进行序列化是因为每个计算框架对于输入样本会有格式要求，序列化实际上是对输入样本进行相应的格式转换。样本生成在整个过程中的时间占比约为 15%。

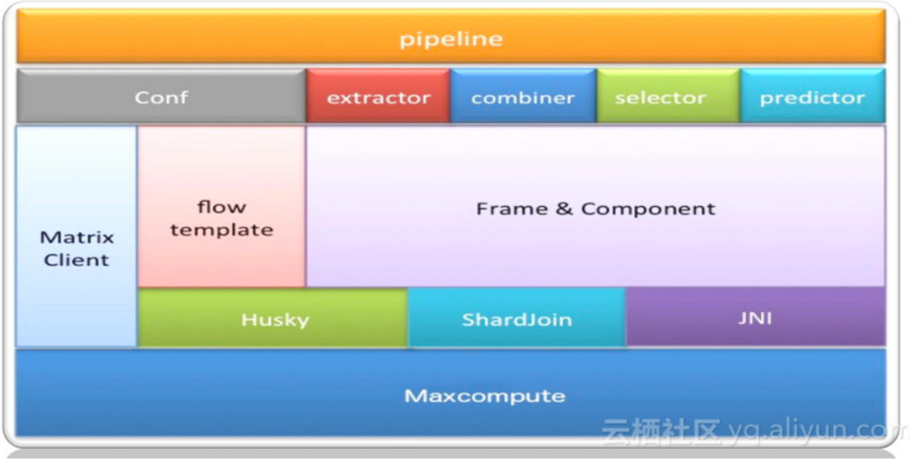
4）**模型训练**。模型训练的输入是上一步产生的千亿级别的样本，输出是每一个特征的权重。比如“男性”这个特征的权重，购买力是一颗星对应的特征的权重。模型训练在整个过程中的时间占比是 40% 左右，这个时间和模型复杂度有关，比如说是运行了简单的逻辑回归或者复杂的深度学习，时间是不同的。

5）**模型评估**。有了训练后的模型，接下来要进行评估，使用 Auc 评估训练模型的效果。一般在样本生成的时候会对样本进行分类，分为训练样本和测试样本，使用测试样本对训练好的模型进行评估。模型评估在整个过程中的时间占比是 5%。

6）**模型应用**。模型评估达到一定标准之后就可以将训练好的模型推到线上，这个过程比较复杂，包括数据导出、数据分发、加载、切换生成在线打分服务。模型应用在整个过程中的时间占比是 5%。

以上介绍的六个步骤和 MaxCompute 最相关的是数据处理（20%）、特征提取（15%）、样本生成（15%）和模型训练（40%），时间占比百分之九十以上的操作都是在 MaxCompute 进行的。

算法实验使用MaxCompute



为了支撑算法实验，我们基于 MaxCompute 搭建了算法实验框架（如上图所示）。整个模型训练不需要开发太多的代码，一般来讲只需要做两个方面的改动，传统的逻辑回归需要增删改特征，深度学习中需要更改各种网络，整个流程是高度一致的。因此我们将这个过程抽象成为 Matrix 的解决方案。这套解决方案对外来讲是运行了一个 pipeline，串联一系列任务，这些任务最终运行在 MaxCompute 上面。对外提供 Matrix Client，用户大部分情况下只需要进行配置文件的修改，比如设置特征抽取的方式，知道原表的 Schema 抽第几行第几个字段；抽取的特征怎么做组合，如第一个特征和第二个特征进行叉乘生成新的特征；包括特征选择方式，如低频特征进行过滤。框架将上述功能组件化，用户只需要像拼积木一样将需要的功能拼接起来，每一个积木进行相应的配置，比如输入表是什么。样本输入模型之前，样本的格式是固定的，在此基础上我们实现了调度框架 Husky，主要实现 pipeline 的管理，实现任务的最大化并行执行。其他功能由于时间关系在此不多做介绍。

四、高级功能和优化

高级配套能力

因为阿里妈妈在 MaxCompute 上曾经的资源使用量占比达到三分之一，从计算到存储。因此我们根据自己的经验，接下来分享一下大家有可能用到的 MaxCompute 的一些高级功能和优化。

MaxCompute 提供了我们认为比较重要的四个功能有：

1）**实时 Dashboard 和 Logview**。Logview 前面已经介绍过了，通过它用户可以不用再去扒日志，可以很快速地查看任务情况，查找问题原因，另外还提供各种实时诊断的功能。当集群出现问题的时候实时 Dashboard 可以从从各个维度帮用户分析当前运行集群的 Project 或 Quta 或任务相关信息，其后台依赖于一系列的源数据管理。当然，MaxCompute 能提供的功能远不止实时 Dashboard 和 Logview，但是这两个功能在个人、集群管理过程是被高度依赖的。

2）**强大的调度策略**。主要有三种：

- 第一种是**交互式抢占**。传统的抢占方式比较粗暴，当用户提交一个任务的时候，比如分 Quta，无论是 Hadoop 还是 MaxCompute，都会分析是 minQuta 还是 maxQuta，这种情况下一定会涉及到共享与资源抢占的问题。如果不抢占，一个任务会跑很长时间；如果直接将任务停掉，已经运行起来的任务可能需要重新再运行，导致效率低下。交互式抢占比较好地解决了这个问题，其提供了一种协议，这个协议需要和各个框架之间达成，比如说要 kill 掉某个任务之前，会在一定时间之前发送 kill 的命令，并给予任务指定的运行时间，如果这个时间结束之后仍然运行不完，则 kill 掉。
- 第二种是**全局调度**。阿里云的机器已经达到了万级，当某个集群的任务跑的很卡的时候，如果发现其他集群比较空闲，全局调度策略便可以发挥作用，将任务分配到较为空闲的集群上运行，这种调度过程对于用户来讲是透明的，用户只能直观地感受任务的运行速度发生变化。
- 第三种是**兼顾 All-or-nothing 和 Increment 的资源分配方式**。简单来讲，比如前者跑了图计算的训练模型，后者跑了 SQL，这两种计算的资源分配方式有很大不同。对于 SQL 来讲，如果需要一千个实例来运行 mapper，不用等到一千个实例攒够了再去运行，可以拿到一个实例运行一个 mapper, 因为这种计算实例之间没有信息交互；但是模型训练是一轮一轮进行迭代的，第一轮迭代运行完之后才能开始运行第二轮迭代，因此注定需要所有的资源准备好了之后才能运行，因此阿里云的调度人员在后台做了很多兼顾这两方面资源分配的工作。

3）**数据地图**。帮助的用户描述数据、任务之间的关系，方便用户后续业务的处理。

4）**数据治理**。任务运行结束对于集群或者任务管理人员来讲并没有结束，还需要去看任务跑得好不好，这个时候服务治理就可以提供很多优化建议，比如某个数据跑到最后没有人用，那与其相关的链路是否可以取消，这种治理不管对于内部系统还是外部系统来讲可以节省很多的资源开销。

计算优化

接下来就上面介绍的数据治理向大家分享一下我们的经验。在数据治理计算优化方面，我们主要的用到了 MaxCompute 的以下四个功能：

1） **无用 / 相似任务分析**。这个很容易理解，它可以帮助用户分析出哪些任务是没有用的，哪些任务是相似的，这需要依托于数据地图的强大关系梳理能力分析出任务的有效性。

2） **HBO**(History-based optimization) + **CBO** (Cost-based optimization)。它其实解决的是优化的问题，在跑计算任务的时候，不管使用的是 SQL 还是 MapReduce，一定会预先设定 CPU 和 Memory 的值，但是预先设定往往是不准确的。解决的方式有两种，一种是先将任务运行一段时间，根据运行情况计算每个实例大概需要的 CPU 和 Memory，这种方式就是 History-based optimization；而第二种方式是 Cost-based optimization，解决的是基于成本的优化，时间关系在此不多做介绍，后期如果大家感兴趣的话，会组织相关的高阶分享。

3） **列裁剪**。它解决的问题是不用讲整个表中的所有字段都列出来，比如 Select

*，根据 SQL 语义，它可以实现十个字段只需要加载前五个字段。这对于整个任务的执行效率包括整个磁盘的 IO 有很大益处。

4）**Service Mode**。传统运行 MapReduce 的时候，会有 shuffle 的过程，这个过程会涉及到数据在 Mapper 和 Reducer 端的落盘，这个落盘操作是很耗时间的，对于一些中小型任务来讲（可能只需要两三分钟就运行完），是不需要落盘操作的。MaxCompute 会预判任务的执行时间，短小任务通过 Service Mode 的方式来降低任务的运行时间。

存储优化

MaxCompute 除了计算优化，还有存储方面的优化。存储优化主要体现在以下一个方面：

1）**无用数据分析和下线**。这个前面也已经反复介绍过了，可以帮助用户分析无用的数据并下线，和无用 Job 分析是类似的原理。这里的难点是“最后一公里”，即数据从离线平台产出之后导到线上，最后这一公里的元素是很难去追踪的，这依赖于工具和平台高度的标准化，阿里内部的好处是这一块已经做到了标准化。随着后续阿里云暴露的服务越来越多，这个难点将有希望被攻克，能够帮用户分析出来哪些数据是真的没有人用。

2）**生命周期的优化**。一份表到底要保存多少时间一开始是依靠人去估计和设置的，比如一年，但根据实际的访问情况你会发现，保存一天或者三天就可以。这个时候依托于 MaxCompute 的数据治理，会帮助用户分析出某张表适合的保存时间，这对于存储的优化具有极其重要的意义。

3）**Archive**。数据是有冷热之分的，尤其是分布式文件存储的时候，都是通过双备份的方式来存储数据，当然双备份尤其意义在，比如可以让你的数据更加可靠、不会丢失，但这样带来了一个问题是数据的存储将会变得大。MaxCompute 提供了冷数据策略，不做双备份，通过一定的策略将数据变成 1.5 备份，用 1.5 倍的空间达到双备份的效果。

4）**Hash Clustering**。这个属于存储和计算优化中和的事情。每次在做

MapReduce 的时候，中间可能需要做 Join 操作，每次 Join 操作的时候可能会对某个表做 Sort 操作，但是 Sort 操作没必要每次都去做，这样就可以针对 Sort 操作提前做一些存储上的优化。

下图展示了的阿里妈妈预估的 ODPS 存储消耗趋势，可以很明显的看到预期消耗随着时间推移几乎呈直线增长，但中间使用 MaxCompute 做了几次优化之后，明显感觉存储消耗增长趋势减缓。

介绍这么多优化的功能，最终目的是希望大家对 MaxCompute 有更多的了解和期待，大家如果有更多的需求可以向 MaxCompute 平台提出。

实时计算助力 1688 打造「实时挑货」系统

作者：葛圆根（水锋） 阿里集团 新零售技术事业群 高级开发工程师

一、背景

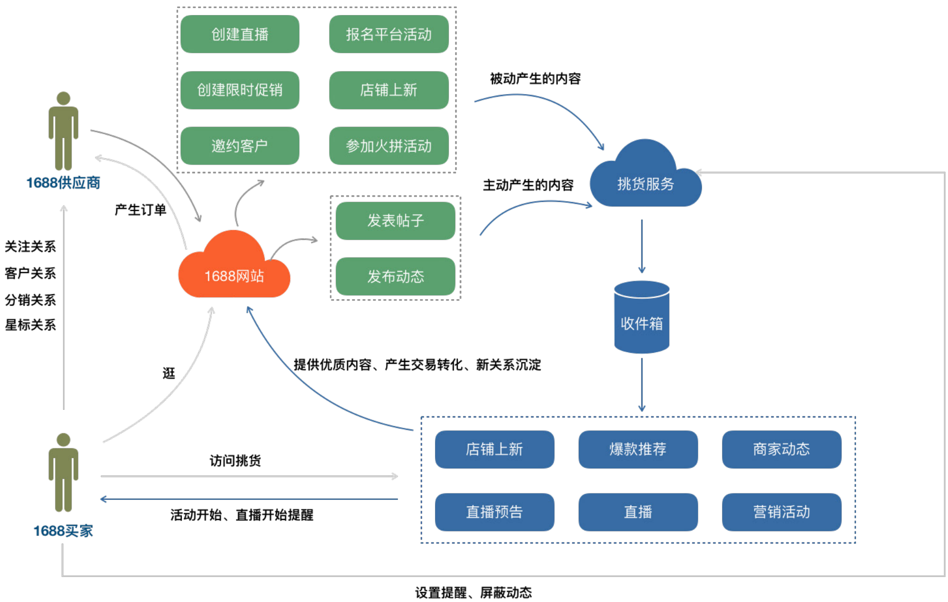
内容是一个电商 app 不可或缺的组成部分。越来越多的人会使用碎片时间浏览手机 app 的内容，包含导购的帖子、短视频、直播等。1688 挑货业务，打造了基于买家和商家之间老买卖关系的内容场。让商家通过内容维系老客户，挖掘新客户。让买家能第一时间获取到关注商家的上新、优惠、直播等信息，为自己的采购等决策提供帮助。

从宏观的背景分析，挑货业务有以下几个特点：

- 基于 1688 的老买卖关系：关注关系、客户会员关系、星标关系、分销关系
- 内容的产生源头是供应商，供应商在 1688 网站的各种行为，都可以转变成内容信息
- 内容的消费源头是供应商的老买家（与供应商有老买卖关系）
- 内容形式比较多样：帖子、营销活动、店铺上新、商家动态、直播、短视频等一系列商家产生的有效内容
- 内容产生形式：供应商的主动行为、供应商的被动行为

基于上面的业务特点，我们梳理下挑货整体的业务架构。

二、业务框架



供应商在 1688 网站可以直接将动态和文章发布到挑货。

供应商在 1688 网站的直播、店铺上新、报名营销活动、邀约客户等行为，会被挑货服务识别到，被动抓取到挑货。

挑货将供应商产生的内容推送到供应商老买卖关系的买家收件箱中。

买家访问挑货时，挑货服务会从买家收件箱中将内容渲染成各种业务卡片，呈现到客户手机端。

买家可以对部分有时效性的内容（直播预告、未开始的互动）设置订阅提醒，当直播开始、活动开始时，挑货服务主动通过消息告知买家。

买家可以通过挑货平台去到供应商的店铺和商品详情去购买商品，产生交易订单。

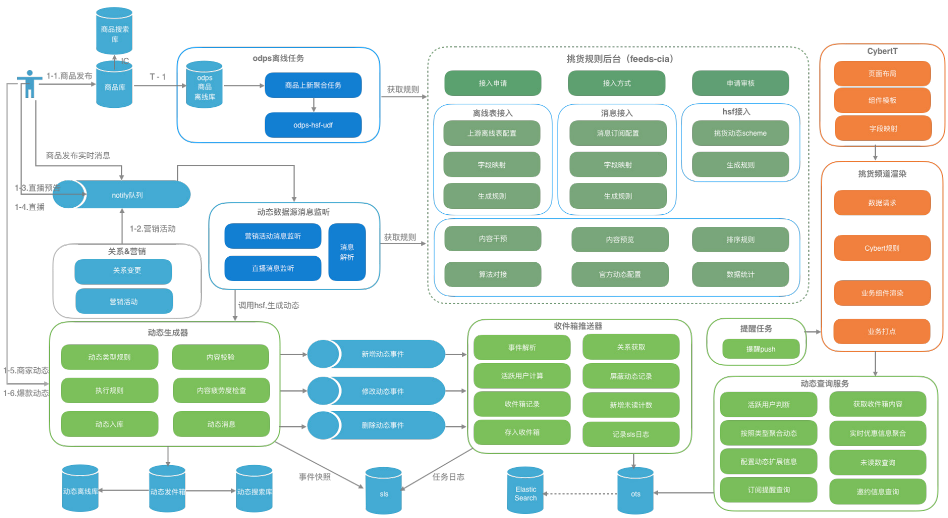
挑货中的优质内容，可以投放到 1688 网站的其他模块，新用户看到后，能有效的产生交易和关系转化。

新用户访问挑货时，挑货服务会推荐部分优质供应商给用户，用户可以关注供应商，看到供应商的挑货内容。

挑货的业务价值在于可以维系供应商的老买卖关系，提供买家快速获取供应商信息的内容，另外可以为供应商引入新的客户，并沉淀为老买卖关系。而挑货的优质内容可以投放到 1688 网站公域频道，一方面节省运营成本，另一方面为供应商引入更多的新用户。挑货在 1688 公域、私域流转上承担着枢纽的作用。

三、技术框架

为了实现挑货的业务架构，在 1688 无线领域孵化出多个技术方案。下面就其中关键的技术细节进行下描述。



挑货总架构图

- 业务对接方式：离线任务、消息对接、实时 rpc
- feeds 流的推拉引擎：feeds 动态生成器、feeds 的收件箱推送、架构图未包

含非活跃用户走拉的模式、feeds 流查询服务

- 动态组件的对接方案 Cybert：业务模型和组件 UI 分离、动态生成 native 组件
- 个性化算法对接方案（未包含）
- 异构系统的监控体系（未包含）：多种中间件平台实现同一个业务时，异构系统的监控成为一大难题。在挑货中，已经设计出了方案，目前正在对接中。

数据统计系统

1. feeds 流推拉结合的引擎

挑货的内容从供应商产生，到供应商的买家去消费，是典型的 feeds 场景。实现一个 feeds 流业界比较流行的主要是推模式、拉模式、或者是推拉结合的模式。

推模式：内容产生后，会为每一个订阅内容的粉丝复制一份内容，放入粉丝独有的收件箱中。

优点：粉丝读取内容非常快，只是一个表查询。在写的过程中可以做很多业务干预。

缺点：写的成本比较高，需要很多耗费存储，数据冗余。

拉模式：内容产生后，只会存入内容产生者的发件箱。读取时，需要粉丝读取每一个被关注用户的发件箱，实时将内容聚合到一起排序，最终展现给粉丝。

优点：存储耗费少，写入速度快。

缺点：查询都需要耗费很多计算资源做内容聚合，业务干预规则多时，会更加加重查询复杂度，查询性能会下降。

推拉模式：综合推模式、拉模式的优缺点使用的方案。

我们从挑货业务的特点和已有的技术积累两个方面，选取了推拉结合，以推为主、拉为辅的技术方案。

1688 有独立的关系团队维护了 1688 所有的老买卖关系数据，已经成为 1688 底层基础服务。而微淘的关系和微淘的内容之前是在一个团队孵化出来的。目前 1688 关系是 B 类的老买卖关系，和淘系的关系没有打通。基于老买卖关系的挑货不能直接复用微淘关系的技术。

微淘开发的 timeline 的拉引擎，使用多级缓存，很好的支持了微淘业务的发展，在 feeds 流拉模式上有很强的专业性。而集团搜索团队的 igraph 图数据引擎也能很好支持数据多维度拉取功能。igraph 有详细的文档、成体系化的接入和运维方案、专业的答疑。从对接的成本和后期运维角度，我们选择了 igraph 作为拉引擎。

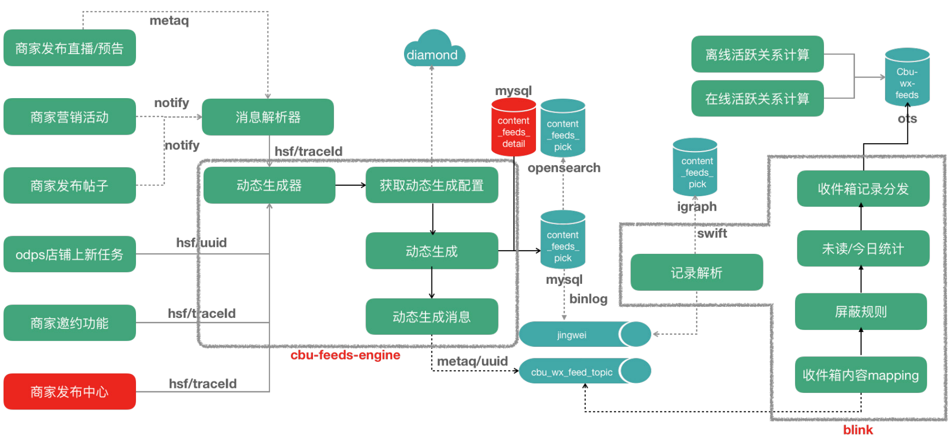
选择以推为主，拉为辅的策略的原因：

1) 数据规模比较小：1688 的老买卖关系总量在 4 亿，一个粉丝比较多的供应商的粉丝数量级在几十万，而在挑货频道活跃的粉丝，对应一个供应商量级在 5 万以下。更多的在几百到上千。所以这种量级的数据，使用推模式能很好的支持查询性能。

2) 集团的中间件平台非常成熟：在 blink 平台开发 feeds 流的收件箱推送功能，能根据数据规模扩缩容资源，有效的支持大规模数据的推送。tablestore 是一个单表支持 10PB 数据的 nosql，作为挑货买家收件箱非常合适。

3) 业务的规则干预比较多：需要支持屏蔽动态这样干预 feeds 展现的业务规则。

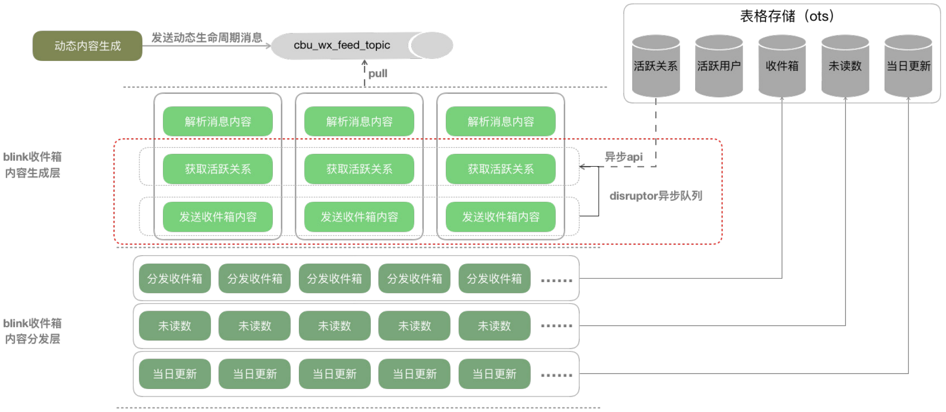
下面看下挑货 feeds 流引擎的架构图：



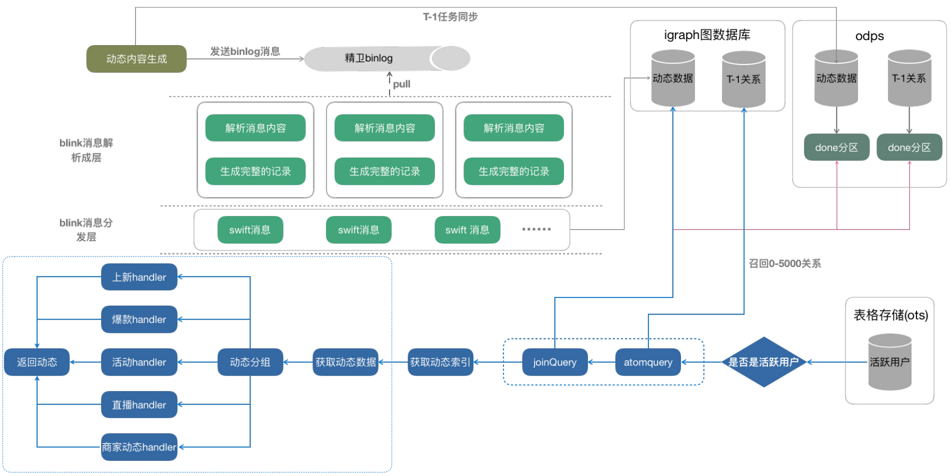
内容数据流入动态生成器，产生挑货 feeds，挑货 feeds 生成后发送 metaq 消息，被运行在 blink 上推送任务监听到，将动态索引写入到用户到收件箱。

下面 3 张比较细节的图：

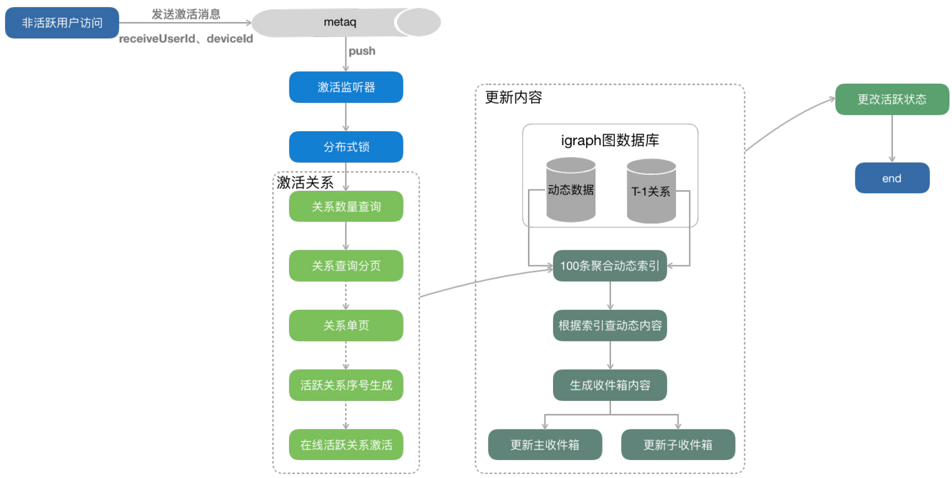
1）活跃用户 – 推模式



2）非活跃用户 – 拉模式



3）非活跃用户激活策略



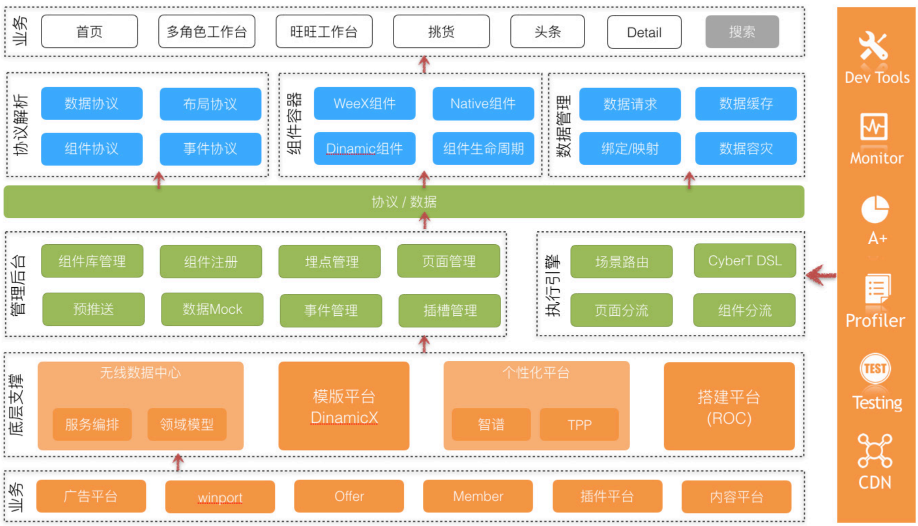
2. 动态组件的对接方案 Cybert

Feeds 列表原有纯 native 的技术解决方案，对业务的核心痛点就在于发版周期和覆盖率的限制；1. 半个月的发版周期严重限制了业务日常营销活动、用户活跃提升的运作方式，业务更希望营销类的活动卡片做到快上快下，不跟随发版节奏；2. 纯 native 卡片还是依托与发版，新的营销活动为了达到最佳效果，还期望于版本的覆盖率，虽然目前安卓、ios 的新版本覆盖周期已经越来越短，但不代表可以做到像 weex 一样立马全版本覆盖。

鉴于这些业务痛点，我们可以采用的技术方案有两种 1. 全 weex 化；2. native 动态化。先从 weex 化来说起，去年整个营销场在整体完成性能优化后，我们已经做到 ios95% 会场秒开，安卓超过 85%，这个秒开率对于挑货 feeds 来说已经足够。但当时没有采用全 weex 化的方案更多出于以下几点考虑：1. 交互或者后期的 tabview 支持，挑货 feeds 作为第二个主 tab 性能、交互、稳定性都必须去强保证，会场一些 tab 切换层面我们可以接受刷页面的方案，但在核心主 tab 我们还希望做到更佳的用户体验；2. 改造成本，原有 native 卡片化的方式，如果整体切换就意味这页面全部重做，这个改造成本不小。因此，我们考虑是否可以把应用在首页、工作台

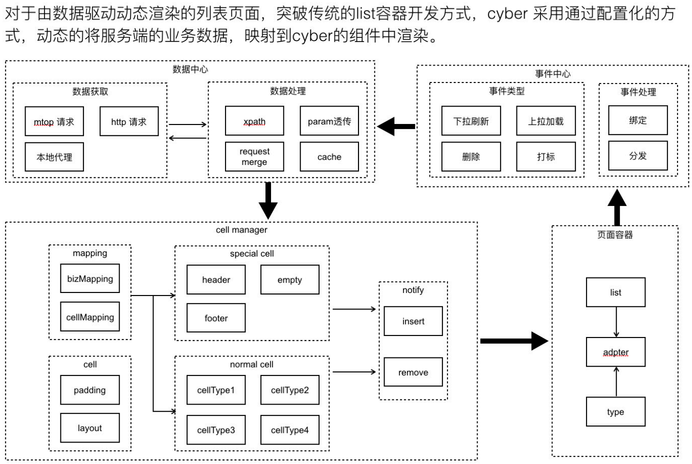
的 native 动态组件化方案做能力扩展，支持类似挑货这类列表组件由“数据驱动 ui”的渲染方式，老卡片仍然采用 native 方案，新组件采用 dinamic 组件插入，既做到老方案兼容，有做到新组件动态发布。在技术改造成本、动态性、业务支撑、交互体验多个点上做平衡。

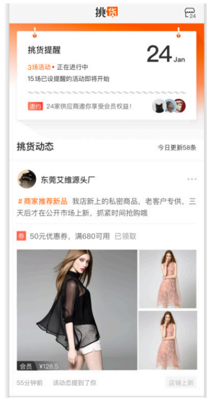
这里，我先简单介绍下目前首页 cyber 动态组件化方案的业务价值、技术架构和能力。CyberT 原生孵化于首页，核心点就是 native 组件化方案，后续因为业务对于动态性能力的要求，支持了 weex 组件、dinamic 组件的扩展，两个动态化组件化方案看 dinamic 组件在首页的性能从渲染耗时、内存开销上更优。由服务端下发页面布局协议、组件 ui 协议、服务端数据协议来驱动端上的渲染，同时在 ui 协议上打平后可以做到组件单端投入双端生效，后续也在进一步支持 h5 外投方案。



而对于 feeds 列表的场景，我们如何升级能力做到“数据驱动 ui”？这主要是升级 List 复杂容器的支持，list 组件的特点是由数据决定有多少个 cell。cybertron 的 layoutprotocl 支持 list 组件声明其 cell 的类型 ，根据服务端返回的 list 数据，结合 cellMapping & datamapping 规则，会生成一个个 cell 实例，并且在框架层支持

了常见的分页加载（page 模式和 offset 模式）。





挑货

这样，由搭建平台配置组件和映射协议，业务服务端下发数据，通过 cybert 容器的客户端 sdk 实现协议、数据解析进行页面的动态渲染。

3. 挑货个性化算法对接（实时特征计算）

优化内容排序：用户在 1688 手机阿里上各种访问行为，特别是在挑货的访问行为，会被抽象成用户的行为特征。通过这些行为特征，根据对应的算法，将用户收件箱中最感兴趣的内容排序到最前面，增加用户的阅读体验。

提取优质的内容：用户的对内容行为，通过一定的算法，可以给内容进行评分，提炼出优质的内容。这些优质内容可以投放到其他公域场景，为商家带来更多的价值。

4. 异构系统的监控体系

整个挑货系统，数据在多个系统中流转，一旦某个系统出现故障，很难排查和追踪问题。现有的监控体系对同构的应用的链路监控已经支持的很完善。一旦涉及到在线应用、离线任务、实时流计算多个异构的系统，就无法有效的做到串联。为了保证挑货的稳定性，我们设置了全局的 uuid，将整个系统的日志采集到云日志中。通过专

门的监控系统，去分析日志，做到链路预警和数据可视化。

5. 数据统计系统

挑货业务承接是 1688 整个买家的内容需求。从业务在挑货的展现的 PV、UV 到具体的业务转化率，再到新业务的 AB 效果对比，都离不开数据化支持。所以数据化是挑货系统的必不可少的组成部分。我们实现思路，从规范业务打点方式入手，到数据日志输出规范、数据统计模块、可视化系统，完成一个闭环的数据化系统。

四、未来发展

内容对接平台化（配置化、数据化）：打造一个内容对接流程、内容数据展现、内容投放的配置后台。

feeds 引擎的抽象，支持更多的业务。

实时计算在「阿里影业实时报表业务」技术解读

作者：向飞（飞大） 阿里影业 技术专家

需求背景

影业实时报表开始做法也是按照传统型报表做法一样，直接从阿里云 rds 写 sql 查询，随着数据量越来越大，这种做法已经没有办法满足业务扩张，带来的问题响应时间变慢，吞吐量低，我们急需要一种技术方案能满足未来 2-3 年随着影院增加，数据增长，而报表功能还能很好的满足客户需求技术方案。

业务目标

时间：平均 1 分钟，像销售报表最差 5 分钟返回结果。

数据准确性：数据不丢失并且跟业务库保持一直，数据正确性要 100%。

数据校对和回溯：数据不准确的时候，能够手动修复报表。

技术挑战

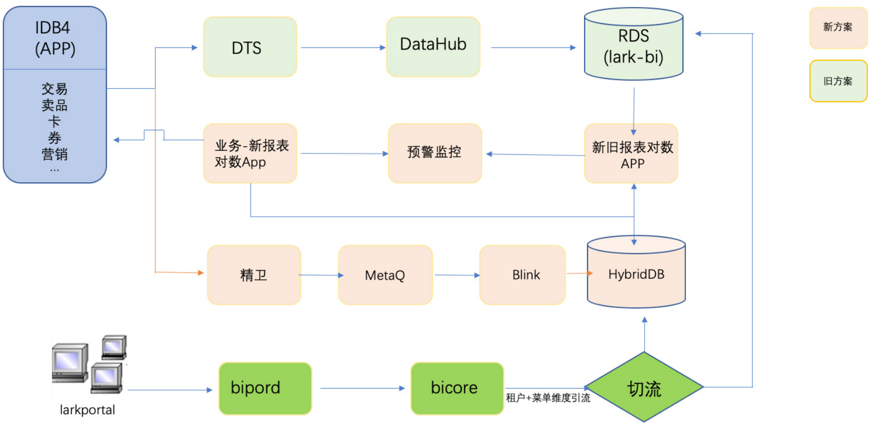
数据幂等：ETL（精卫 /Blink）如何保障业务明细数据变更时间顺序，即：对同一条记录进行 update，如何保障 update 的顺序是正确的；同样对 delete/insert 一样要求。

数据校对：目前试运行阶段，一方面采用新旧报表对比，另外一方面，采用“数据回溯”，如每隔 5 分钟从业务库同步最近 5 分钟的变动数据。

数据查询性能：HybirdDB for Mysql 存储中如何保障根据不同条件查询的性能，包括单表查询和多表关联查询。

技术方案

数据架构图



通过上图可以看出我们新旧报表方案，根据报表目标梳理出要处理的细节点：

- 集团内部有很多种 blink 的数据源，如 tt、datahub、metaq、notify，而我们在弹外使用了很久 metaq，并且实时性也没什么问题，所以就用了 metaq 作为 blink 的数据源。
- 精卫发送数据到 Metaq，而 Blink 订阅 Metaq 数据，整个解析模型现在 Blink 不支持，这个需要自己 Blink UDX 函数去转换，这个也需要去了解精卫发送到 Metaq 的协议，UDX 自定义函数。
- 精卫发送数据到 Metaq，如下图，目前只支持增量方式，并不支持全量到 Metaq，而全量支持只有 RDS 到 RDS，这个也需要去了解精卫发送到 Metaq 的协议。

整体配置流程

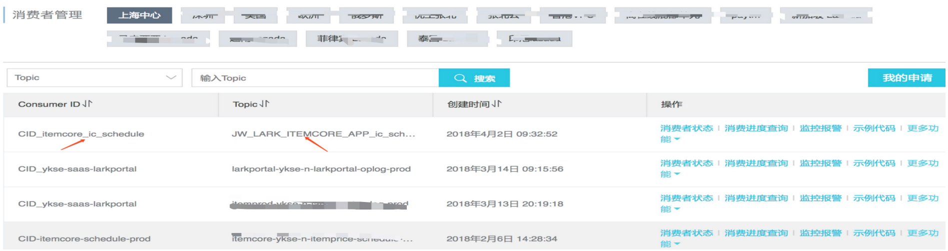
选择自主消费模式

选择 TDDL 数据源和配置表

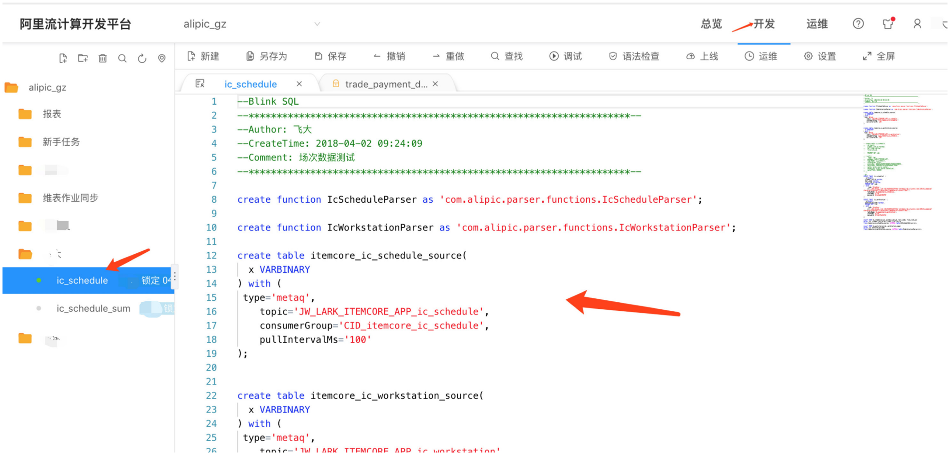
选择链路类型

精卫服务配置完成

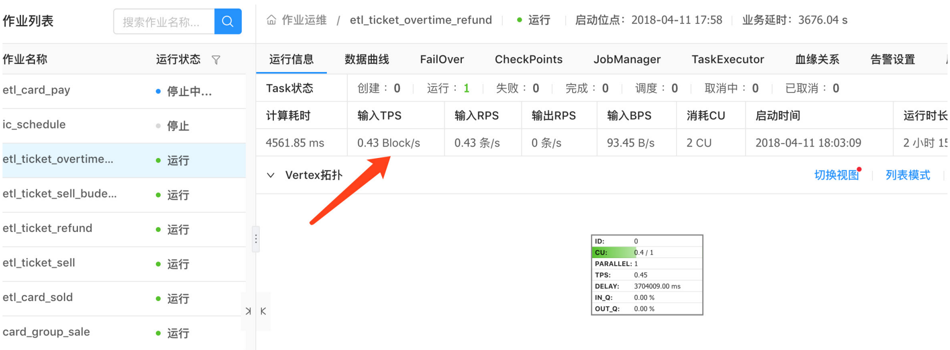
订阅 metaq



编写作业代码



上线运维作业



精卫和 metaq 消息传输协议

为什么要去了解 metaq 和精卫之间传输的协议，是由于目前精卫官方提供的功能，没办法满足我们的业务需求，他提供了全量功能是针对 db 对 db，没有办法 db 对 metaq，并且我们还要兼容 blink 脚本共用，要满足这个功能就要查看精卫现在是怎么做的，通过查看文档和代码 3.0.12 之前的版本是 dbsync 协议而这个协议本身基于对象是 DBMSRowChange，而之后的版本是 EventMessage，现在容器代码也是返回 DataMessage 其结构跟 EventMessage 是一样，但是通过读取代码发现核心的还是 DBMSRowChange 通过这里面的数据组装到 thirft 协议中去，核心代码如下：

精卫发送 metaq 关键代码

com.alibaba.middleware.jingwei.core.applier.BatchMetaq3Applier

```
MessageBuilder messageBuilder = new PartitionMessageBuilder();
ThriftHeader header = null;
List<Message> messageFlush = new ArrayList<Message>(messages.size());
for (com.alibaba.middleware.jingwei.externalApi.message.Message message : messages) {
    DbSyncMessage dbSyncMessage = (DbSyncMessage) message;
    DBMSRowChange dbmsRowChange = (DBMSRowChange) dbSyncMessage.getDbMessage();
    try {
        header = buildThriftHeader(dbSyncMessage);
        // 省去部分代码
        for (Message thiftMessage : messageList) {
            messageFlush.add(thiftMessage);
        }
    } catch (Throwable e) {
        throw new JingWeiException(builder.toString(), e);
    }
}

try {
    for (Message thiftMessage : messageBuilder.flush(header)) {
        messageFlush.add(thiftMessage);
    }
} catch (TException e) {
    String err = "Thrift serialization error: " + e.getMessage();
    logger.error(err, e);
    throw new JingWeiException(err, e);
```

```
}
msgSender.send(messageFlush);
```

代码主要是讲经过 binlog 的数据转换成 Message 对象，Message 把响应的数据丢到 Thrift 协议中去，最核心就是把数据转成 Thrift 然后丢到 metaq。

精卫解析 metaq 核心代码

com.alibaba.middleware.jingwei.client.util.MetaqToJingweiMsgParser

```
ThriftHeader thriftHeader;
try {
    thriftHeader = ThriftHelper.loadThrift(msg.getBody(), eventSet);
} catch (Exception e) {
    if (DynamicConfig.isSKIP_ALL_EXCEPTION(taskName)) {
        logger.warn("SKIP_ALL_EXCEPTION is true, will skip the msg : " + msg, e);
        continue;
    } else {
        throw new JingWeiException("ThriftHelper.loadThrift occur error" + ",
msg : " + msg, e);
    }
}

for (ThriftEvent event : eventSet) {
   // 省去部分代码
}
```

上面代码是精简出来的，解析 Thrift 处理过的 metaq 的数据，如果针对上面 ThirftEvent 感兴趣可以仔细看看他们内部怎么处理，通过测试这种协议反解析出来的速度很快，都是毫秒级别。

blink 解析 metaq 模型

解析 metaq 发送过来的核心代码

```
public static<T> List<T> parseMetaQMessage(byte[] bytes, Class c) throws Exception {
    List<T> sourceList = new ArrayList<>();
    List<ThriftEvent> eventSet = new ArrayList<>();
    ThriftHelper.loadThrift(bytes, eventSet);
```

```
        // 协议解包
        for (ThriftEvent event : eventSet) {
            // 省去部分代码
        }
        return sourceList;
    }
```

配置流程

可以参考上面整体配置流程。

精卫回溯方案

为什么不自己开发一套代码，直接把消息丢到 metaq 不就行了吗？ 是因为精卫是按照任务纬度来划分，一个任务有多个表，如果自己再开发一套就需要把现有的任务跟表的关系关联在一起，这样造成很大的工作量，并且通过查看官方文档和 vone 咨询那边的开发，他们现在没有办法支持全量同步到 metaq，只能通过精卫容器模式自己编写代码来实现。

全量发送 metaq 核心代码

```
Metaq3ApplierVO metaq3ApplierVO = new Metaq3ApplierVO();
metaq3ApplierVO.setCompressionType("NONE");
BatchMetaq3Applier outerClass = new BatchMetaq3Applier();
outerClass.setMetaq3ApplierVO(metaq3ApplierVO);
BatchMetaq3Applier.PartitionMessageBuilder messageBuilder = outerClass.new
PartitionMessageBuilder();
ThriftHeader header = null;
String topic = entryMessageData.getKey();
List<Message> messageFlush = new ArrayList<Message>(messages.size());
for (DataMessage message : entryMessageData.getValue()) {
      DBMSRowChange dbmsRowChange = null;
      List<DBMSColumn> columns = new ArrayList<>();
      int columnIndex = 0;
      // 省去部分代码
}
```

基于上面分析的协议，而现在返回 DataMessage 对象是精卫封装后的，为了兼容一套 Blink 协议转换，需要把转换后的对象再组装成 Thrift 协议，在组装过程有

很多依赖，有些依赖没有考虑全面报了很多错误，然后去适配，还有个细节由于每个任务对应不同的表，而不同的表是有不同的 topic，所以在代码上把接受到数据做了一层分组，这样按照精卫生成的 topic，就能一套代码运行 N 个服务。

回朔配置流程

选择精卫自主消费

选择据链路类型

上传代码到容器

petadata 技术

petadata 百万查询都是毫秒级别，不过不能分页过多，因为页数过多多从多个分区拿数据，然后再组装很耗时。

总结

精卫发送 metaq 的数据带毫秒的，格式如：2018-03-10 12:12:12.0，而回溯全量方案获取的 dataMessage 是直接从数据库拿的值，没有带毫秒。格式如：2018-03-10 12:12:12。

petadata 不支持 group by 后再 order by。

幂等顺序问题通过 LAST_VALUE 和 group by 来解决。

blink.state.ttl.ms（默认一天）和 state.backend.rocksdb.ttl.ms（默认 1 一天半），一般是按照这个规则设置两个参数 blink.state.ttl.ms <= state.backend.rocksdb.ttl.ms，由于我们报表业务特殊性，是凌晨 6 点到第二天凌晨 6 点为一个周日，所以这两个参数最少要设置为 2 天，才能保证准确性。

有 join 操作，先过滤、去重，再 join。

如果 in/outcpu 过高，则从 core、memory、parallelism，cu 几方面扩大调优。

传输数据过多，可以增大 blink.miniBatch.size，以减少在计算过程中对 IO 的操作，但只能对 group by 生效。

精卫 bug：当带上自定义查询条件，如果 id 是主键，且是 varchar 类型，精卫拼装 sql 就会报错，这个已经反馈给精卫开发，正常 id 都是 long，由于我们在弹外的业务是字符串，所以复现这个 bug，后续的表一定要按照集团规范，不然使用其他中间件估计也会有问题。

Alibaba Group
阿里巴巴集团

阿里技术

扫一扫二维码图案，关注我吧

阿里云实时计算

MaxCompute 开发者交流钉钉群

DataWorks 开发者交流钉钉群

扫码关注阿里技术