架构师

ARCHITECT

特刊

大数据平台架构

SPECIAL ISSUE

Aug,2016

架构师特刊

Geekbang. 极客邦科技 | InfoQ

序言

当笔者一开始就将大数据技术和金钱庸俗地扯上关系时，相信不少开发者和读者是不屑的，然而这里仅仅谈笔者在全球架构师峰会深圳站记录的两笔足以惊讶的账：

Twitter 机器学习平台组负责人郭晓江：我们上了大规模的在线学习的东西之后，把整个 Twitter 的营收提高了大概 30% 左右，这对于 Twitter 是几十亿美金的 business，30% 是一个非常非常可观的数字，而且超越了所有算法工程师这么多年加起来的努力。

阿里巴巴速卖通技术部总监郭东白：（基于大数据的全球电商系统性能优化）我们把订单增加了 10.5%，这一年白花花的银子相当于白来的（例如一年的 GMV 是 10 亿美金，现在变成了 11 亿），直接回报是非常大的，这个项目给整个 AliExpress 每年带来了数亿美金的回报。

对于体量已经非常庞大的企业，大数据技术依旧能给它们带来可观的利润提升，这也许是我们可见的最直观的大数据价值。

相信在这个大数据引领变革的时代，不少开发者和企业都眼红大数据技术能给自身带来怎样的利益。大数据听起来似乎深奥遥远，然而以机器学习和广告为例，广告优化所需要的大数据机器学习算法等知识只需要上过 Andrew N 的机器学习课就已经足够，但无论是 Twitter 还是 Google、 Facebook 都投入了无数的人才和资源，就是要把机器学习做到系统规模化和服务规模化，而支撑起系统规模化和服务规模化关键就是各互联网企业所努力搭建的大数据平台。

回到大数据本身，虽然企业推动自身大数据技术发展的动机不同，但实践起来无非以下三步：

- 依托企业资源获取海量的用户行为，归纳提炼为数据；
- 凭借现有的大数据技术对数据全量挖掘分析；
- 根据企业的不同需求开发对应的应用。

作为用户的我们也许只关心企业在第 3 步中为我们提供了怎样的服务，例如个性化推荐音乐、与人工智能对弈等等；但作为大数据开发者应当了解大数据平台的建设是贯穿每一个环节的，唯有认真学习与实践才能在开发的每一步中提高效率和降低成本。

因此在本期技术特刊中我们总结了酷狗、美团、Airbnb 的大数据平台架构实践范例，以及携程、IFTTT、卷皮等公司业务结合大数据平台的架构分析，希望读者能通过不同的角度从中收获到搭建大数据平台知识。

最后，互联网天生为大数据提供了极易获取数据的平台，我们正处于大数据技术爆炸疯狂的最好时代，也处于处处被记录牺牲隐私的最差时代，我们唯有顺势而为沉淀技术，才不至于被这个充满机遇和挑战的新时代提前淘汰。

ArchSummit 组委会

目录

05 酷狗音乐的大数据平台重构

19 Airbnb 的大数据平台架构

25 美团大数据平台架构实践

41 电商卷皮 BI 的实践演进和架构体系

51 解密 IFTTT 的数据架构

56 面对百亿用户数据，日均亿次请求，携程应用架构如何涅槃

经典大数据架构案例 1：
酷狗音乐的大数据平台重构

作者 **王劲**

【编者按】本文是酷狗音乐的架构师王劲对酷狗大数据架构重构的总结。酷狗音乐的大数据架构本身很经典，而这篇讲解了对原来的架构上进行重构的工作内容，总共分为重构的原因、新一代的大数据技术架构、踩过的坑、后续持续改进四个部分来给大家谈酷狗音乐大数据平台重构的过程。

眨眼就新的一年了，时间过的真快，趁这段时间一直在写总结的机会，也总结下上一年的工作经验，避免重复踩坑。酷狗音乐大数据平台重构整整经历了一年时间，大头的行为流水数据迁移到新平台稳定运行，在这过程中填过坑，挖过坑，为后续业务的实时计算需求打下了很好的基础。在此感谢酷狗团队成员的不懈努力，大部分从开始只知道大数据这个概念，到现在成为团队的技术支柱，感到很欣慰。

从重构原因，技术架构，踩过的坑，后续持续改进四个方面来描述酷狗音乐大数据平台重构的过程，在此抛砖引玉，这次的内容与 6 月份在高可用架构群分享的大数据技术实践的有点不同，技术架构做了些调整。

其实大数据平台是一个庞大的系统工程，整个建设周期很长，涉及的生态链很长（包括：数据采集、接入，清洗、存储计算、数据挖掘，可视化等环节，每

个环节都可以当做一个复杂的系统来建设），风险也很大。

一、重构原因

在讲重构原因前，先介绍下原有的大数据平台架构，如图 1。

从图 1 可知，主要基于 Hadoop1.x+hive 做离线计算 (T+1)，基于大数据平台的数据采集、数据接入、数据清洗、作业调度、平台监控几个环节存在的一些问题来列举下。

数据采集：

- 数据收集接口众多，且数据格式混乱，基本每个业务都有自己的上报接口；
- 存在较大的重复开发成本；
- 不能汇总上报，消耗客户端资源，以及网络流量；
- 每个接口收集数据项和格式不统一，加大后期数据统计分析难度；
- 各个接口实现质量并不高，存在被刷，泄密等风险。

图 1

数据接入：

- 通过rsync同步文件，很难满足实时流计算的需求；
- 接入数据出现异常后，很难排查及定位问题，需要很高的人力成本排查；
- 业务系统数据通过Kettle每天全量同步到数据中心，同步时间长，导致依赖的作业经常会有延时现象。

数据清洗：

- ETL集中在作业计算前进行处理；
- 存在重复清洗。

作业调度：

- 大部分作业通过crontab调度，作业多了后不利于管理；
- 经常出现作业调度冲突。

平台监控：

- 只有硬件与操作系统级监控；
- 数据平台方面的监控等于空白。

基于以上问题，结合在大数据中，数据的时效性越高，数据越有价值（如：实时个性化推荐系统，RTB 系统，实时预警系统等）的理念，因此，开始大重构数据平台架构。

二、新一代大数据技术架构

在讲新一代大数据技术架构前，先讲下大数据特征与大数据技术要解决的问题。

1. 大数据特征：“大量化 (Volume)、多样化 (Variety)、快速化 (Velocity)、价值密度低（Value）”就是“大数据”显著的 4V 特征，或者说，只有具备这些特点的数据，才是大数据（见图 2）。

2. 大数据技术要解决的问题：大数据技术被设计用于在成本可承受的条件

图 2

下，通过非常快速（velocity）地采集、发现和分析，从大量（volumes）、多类别（variety）的数据中提取价值（value），将是 IT 领域新一代的技术与架构（见图 3）。

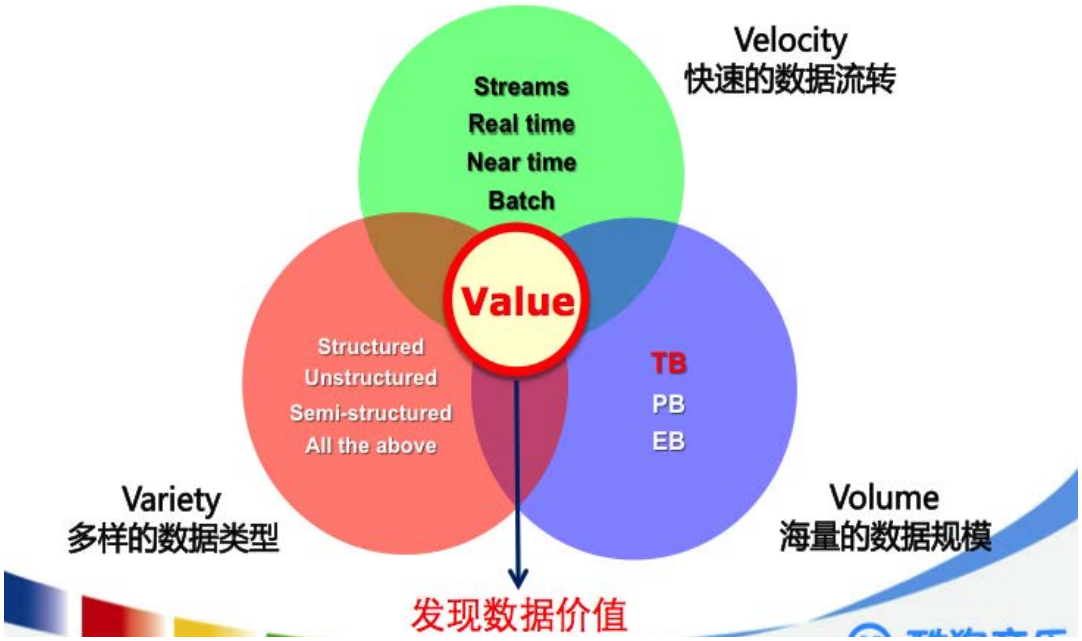
介绍了大数据的特性及大数据技术要解决的问题，图 4 是新一代大数据技术架构的数据流架构图。

从这张图中，可以了解到大数据处理过程可以分为数据源、数据接入、数据清洗、数据缓存、存储计算、数据服务、数据消费等环节，每个环节都有具有高可用性、可扩展性等特性，都为下一个节点更好的服务打下基础。整个数据流过程都被数据质量监控系统监控，数据异常自动预警、告警。

新一代大数据整体技术架构如图 5 所示。

将大数据计算分为实时计算与离线计算，在整个集群中，奔着能实时计算的，一定走实时计算流处理，通过实时计算流来提高数据的时效性及数据价值，同时减轻集群的资源使用率集中现象。

整体架构从下往上解释下每层的作用。



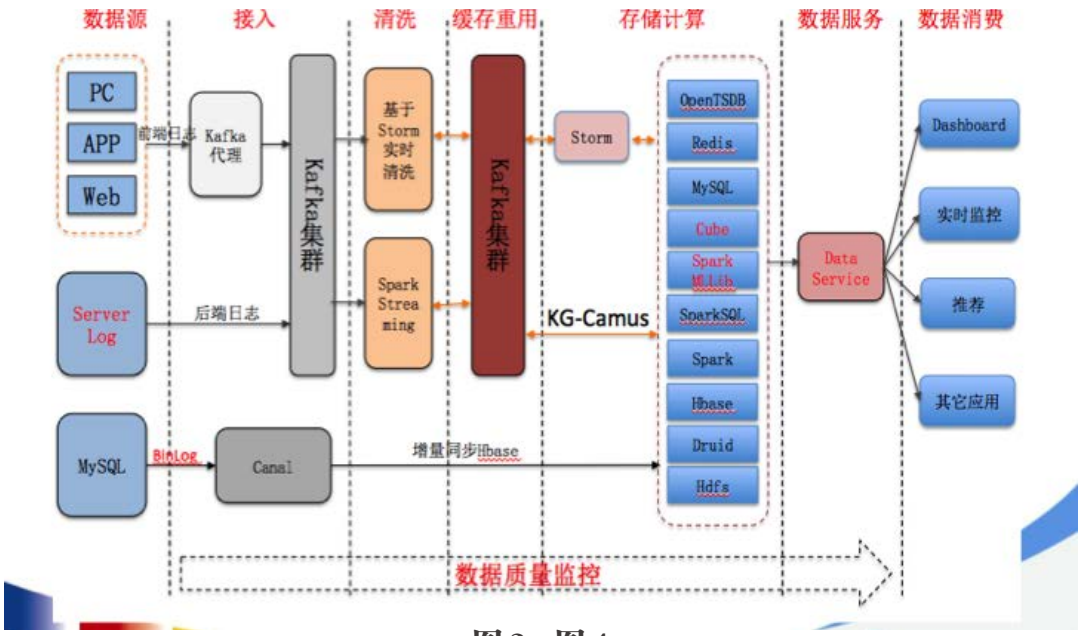


图 3、图 4

数据实时采集：

主要用于数据源采集服务，从数据流架构图中，可以知道，数据源分为前端日志，服务端日志，业务系统数据。下面讲解数据是怎么采集接入的。

a. 前端日志采集接入：

前端日志采集要求实时，可靠性，高可用性等特性。技术选型时，对开源的

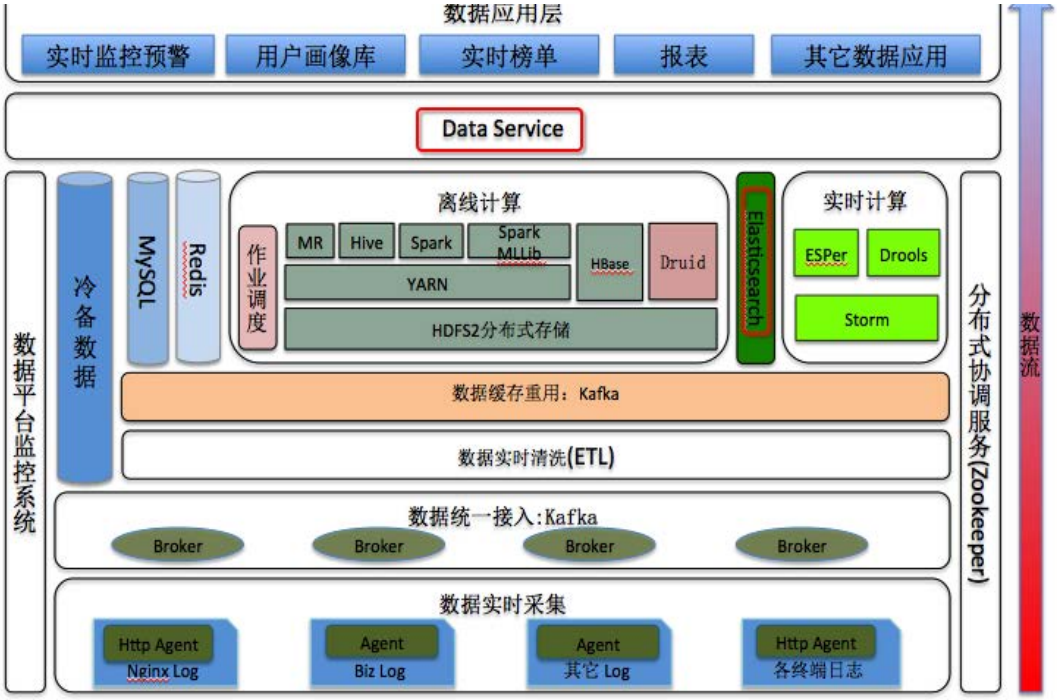


图 5

数据采集工具 flume,scribe,chukwa 测试对比，发现基本满足不了我们的业务场景需求。所以，选择基于 kafka 开发一套数据采集网关，来完成数据采集需求。数据采集网关的开发过程中走了一些弯路，最后采用 nginx+lua 开发，基于 lua 实现了 kafka 生产者协议。有兴趣同学可以去 Github 上看看，另一同事实现的，现在在 github 上比较活跃，被一些互联网公司应用于线上环境了。

b. 后端日志采集接入：

FileCollect，考虑到很多线上环境的环境变量不能改动，为减少侵入式，目前是采用 Go 语言实现文件采集，年后也准备重构这块。

前端，服务端的数据采集整体架构如图 6 所示。

c. 业务数据接入

利用 Canal 通过 MySQL 的 binlog 机制实时同步业务增量数据。

数据统一接入：为了后面数据流环节的处理规范，所有的数据接入数据中心，必须通过数据采集网关转换统一上报给 Kafka 集群，避免后端多种接入方式的处理问题。

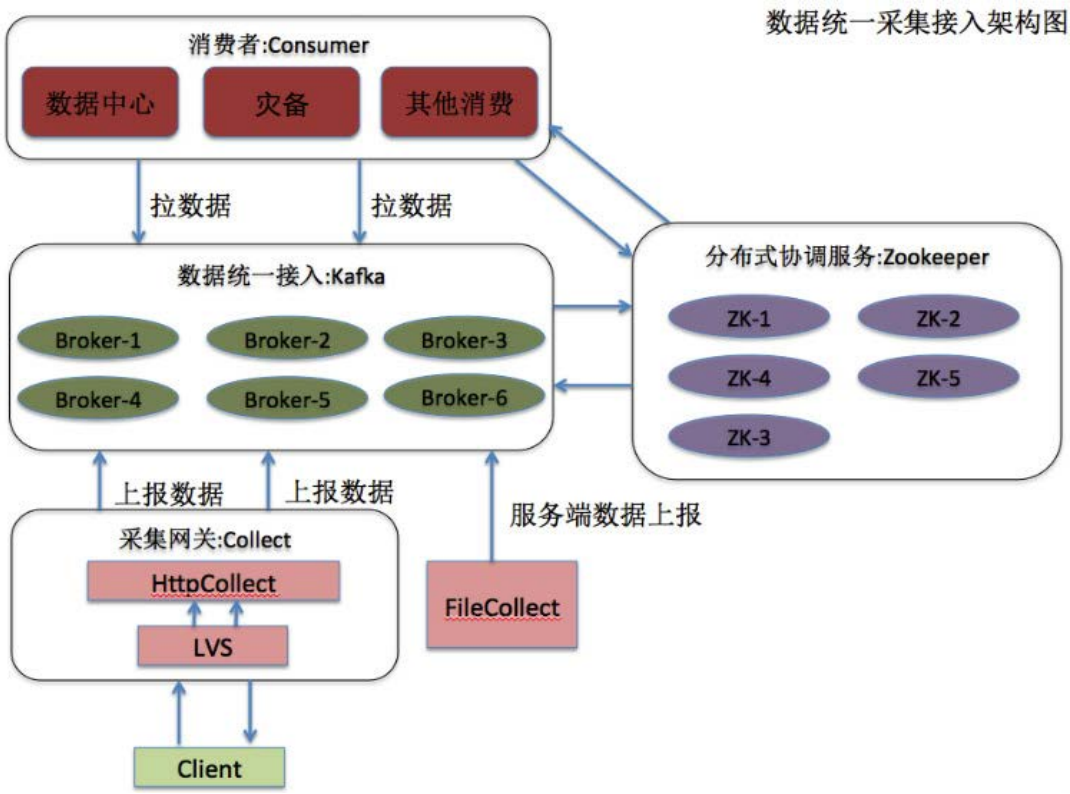


图 6

数据实时清洗（ETL）：为了减轻存储计算集群的资源压力及数据可重用性角度考虑，把数据解压、解密、转义，部分简单的补全，异常数据处理等工作前移到数据流中处理，为后面环节的数据重用打下扎实的基础（实时计算与离线计算）。

数据缓存重用：为了避免大量数据流（400+ 亿条 / 天）写入 HDFS，导致 HDFS 客户端不稳定现象及数据实时性考虑，把经过数据实时清洗后的数据重新写入 Kafka 并保留一定周期，离线计算（批处理）通过 KG-Camus 拉到 HDFS（通过作业调度系统配置相应的作业计划），实时计算基于 Storm/JStorm 直接从 Kafka 消费，有很完美的解决方案 storm-kafka 组件。

离线计算（批处理）：通过 spark，spark SQL 实现，整体性能比 hive 提高 5—10 倍，hive 脚本都在转换为 Spark/Spark SQL；部分复杂的作业还是通过 Hive/Spark 的方式实现。在离线计算中大部分公司都会涉及到数据仓库的问题，酷狗

音乐也不例外，也有数据仓库的概念，只是我们在做存储分层设计时弱化了数据仓库概念。数据存储分层模型如图 7 所示。

大数据平台数据存储模型分为：数据缓冲层 Data Cache Layer（DCL）、数据明细层 Data Detail Layer（DDL）、公共数据层（Common）、数据汇总层 Data Summary Layer（DSL）、数据应用层 Data Application Layer（DAL）、数据分析层（Analysis）、临时提数层（Temp）。

1）数据缓冲层 (DCL)：存储业务系统或者客户端上报的，经过解码、清洗、转换后的原始数据，为数据过滤做准备。

2）数据明细层（DDL）：存储接口缓冲层数据经过过滤后的明细数据。

3）公共数据层（Common）：主要存储维表数据与外部业务系统数据。

4）数据汇总层（DSL）：存储对明细数据，按业务主题，与公共数据层数据

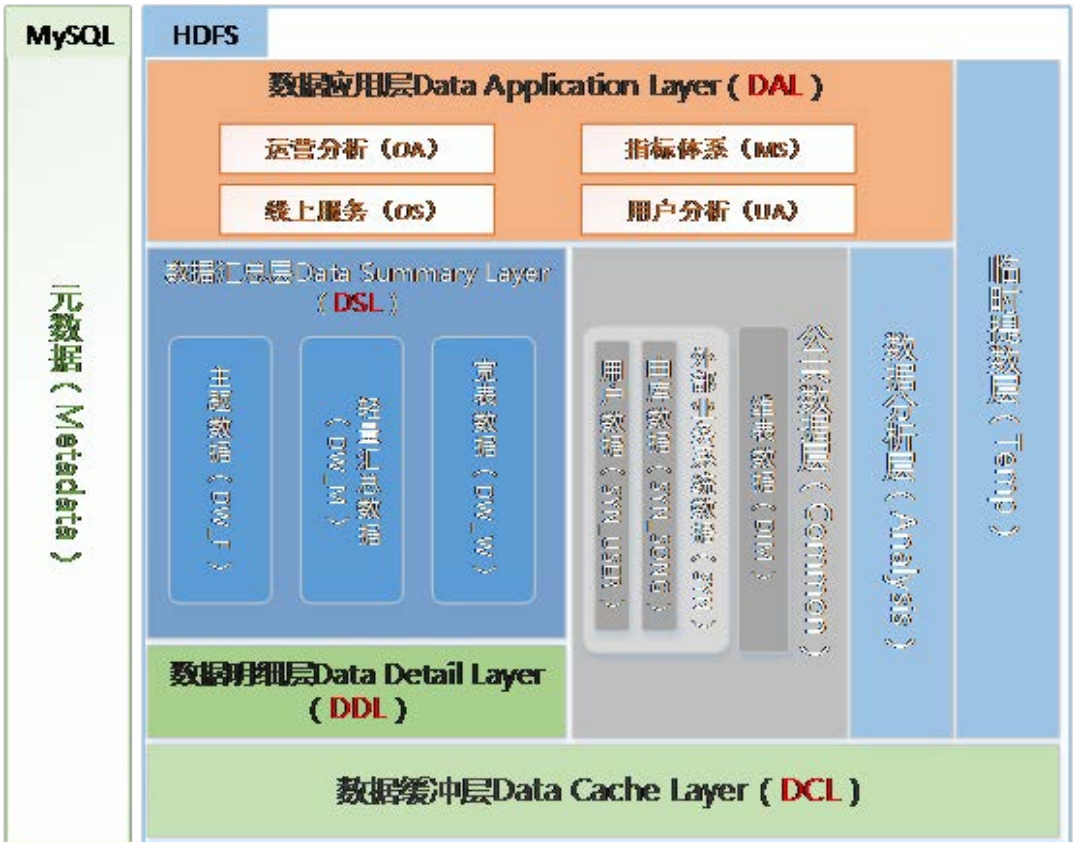


图 7

进行管理后的用户行为主题数据、用户行为宽表数据、轻量汇总数据等。为数据应用层统计计算提供基础数据。数据汇总层的数据永久保存在集群中。

5）数据应用层（DAL）：存储运营分析（Operations Analysis ）、指标体系（Metrics System）、线上服务（Online Service）与用户分析（User Analysis）等。需要对外输出的数据都存储在这一层。主要基于热数据部分对外提供服务，通过一定周期的数据还需要到 DSL 层装载查询。

6）数据分析层（Analysis）：存储对数据明细层、公共数据层、数据汇总层关联后经过算法计算的、为推荐、广告、榜单等数据挖掘需求提供中间结果的数据。

7）临时提数层（Temp）：存储临时提数、数据质量校验等生产的临时数据。

实时计算：基于 Storm/JStorm，Drools,Esper。主要应用于实时监控系统、APM、数据实时清洗平台、实时 DAU 统计等。

HBase/MySQL：用于实时计算，离线计算结果存储服务。

Redis：用于中间计算结果存储或字典数据等。

Elasticsearch：用于明细数据实时查询及 HBase 的二级索引存储（这块目前在数据中心还没有大规模使用，有兴趣的同学可以加入我们一起玩 ES)。

Druid：目前用于支持大数据集的快速即席查询 (ad-hoc)。

数据平台监控系统：数据平台监控系统包括基础平台监控系统与数据质量监控系统，数据平台监控系统分为 2 大方向，宏观层面和微观层面。宏观角度的理解就是进程级别，拓扑结构级别，拿 Hadoop 举例，如：DataNode，NameNode，JournalNode，ResourceManager，NodeManager，主要就是这 5 大组件，通过分析这些节点上的监控数据，一般你能够定位到慢节点，可能某台机器的网络出问题了，或者说某台机器执行的时间总是大于正常机器等等这样类似的问题。刚刚说的另一个监控方向，就是微观层面，就是细粒度化的监控，基于 user 用户级别，基于单个 job，单个 task 级别的监控，像这类监控指标就是另一大方向，这类的监控指标在实际的使用场景中特别重要，一旦你的集群资源是开放给外面的用

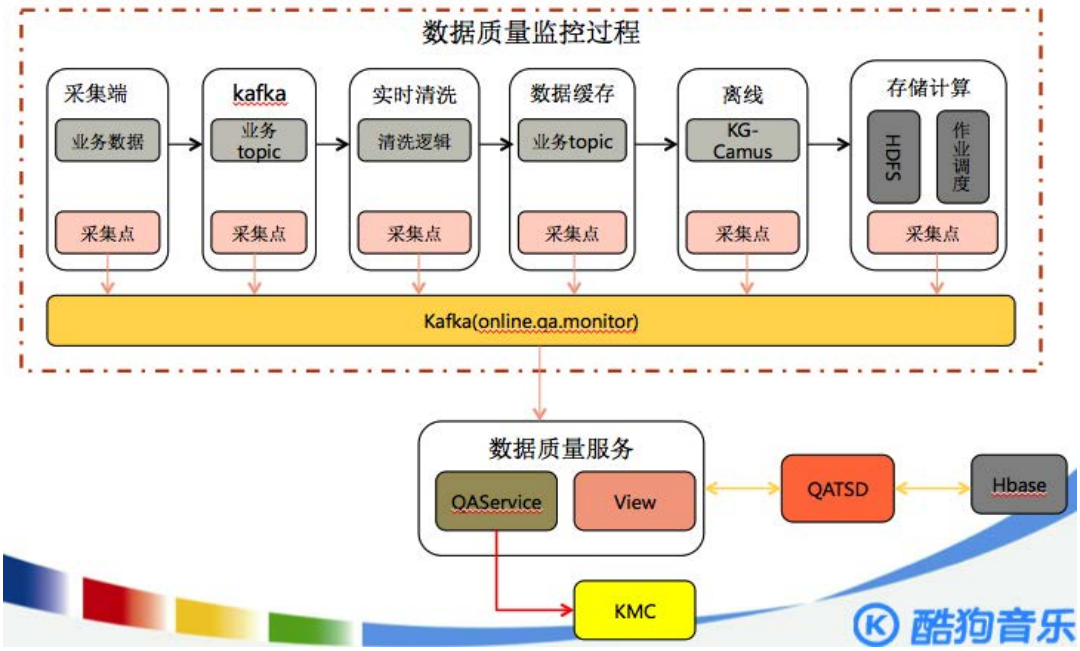


图 8

户使用，用户本身不了解你的这套机制原理，很容易会乱申请资源，造成严重拖垮集群整体运作效率的事情,所以这类监控的指标就是为了防止这样的事情发生。目前我们主要实现了宏观层面的监控。如：数据质量监控系统实现方案如图 8 所示。

三、大数据平台重构过程中踩过的坑

我们在大数据平台重构过程中踩过的坑，大致可以分为操作系统、架构设计、开源组件三类，下面主要列举些比较典型的，花时间比较长的问题。

1、操作系统级的坑

Hadoop 的 I/O 性能很大程度上依赖于 Linux 本地文件系统的读写性能。Linux 中有多种文件系统可供选择，比如 ext3 和 ext4，不同的文件系统性能有一定的差别。我们主要想利用 ext4 文件系统的特性，由于之前的操作系统都是 CentOS5.9 不支持 ext4 文件格式，所以考虑操作系统升级为 CentOS6.3 版本，部署 Hadoop 集群后，作业一启动，就出现 CPU 内核过高的问题。如图 9 所示。

经过很长时间的测试验证，发现 CentOS6 优化了内存申请的效率，引入了

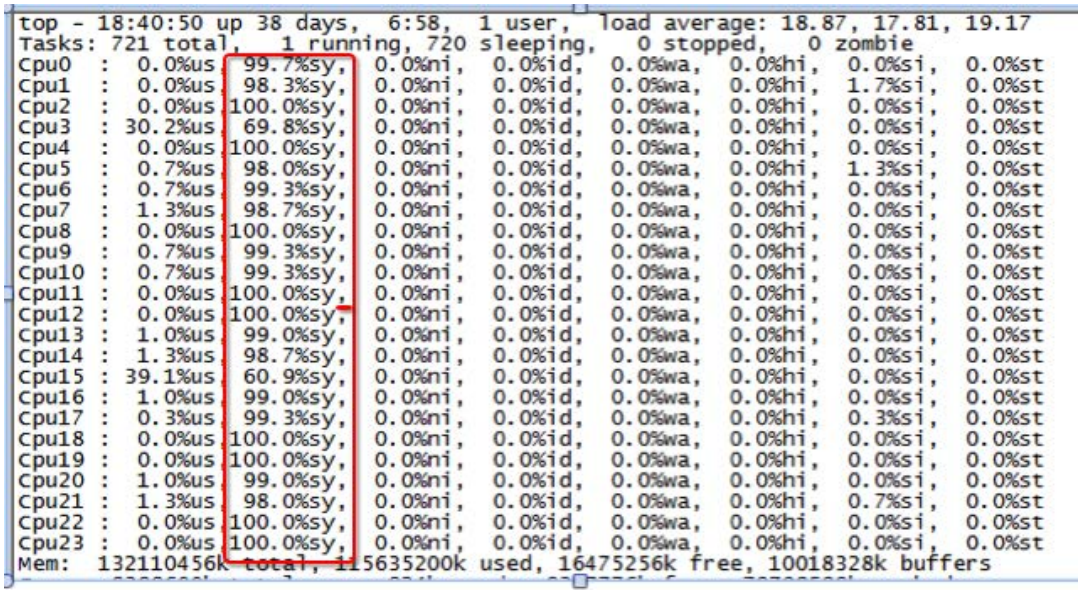


图 9、图 10

THP 的特性，而 Hadoop 是高密集型内存运算系统，这个改动给 hadoop 带来了副作用。通过以下内核参数优化关闭系统 THP 特性，CPU 内核使用率马上下降，如图 10 所示。

```
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

2、架构设计的坑

最初的数据流架构是数据采集网关把数据上报给 Kafka，再由数据实时清洗

平台（ETL）做预处理后直接实时写入 HDFS，如图 11 所示。

图 11

此架构，需要维持 HDFS Client 的长连接，由于网络等各种原因导致 Storm 实时写入 HDFS 经常不稳定，隔三差五的出现数据异常，使后面的计算结果异常不断，当时尝试过很多种手段去优化，如：保证长连接、连接断后重试机制、调整 HDFS 服务端参数等，都解决的不是彻底。

每天异常不断，旧异常没解决，新异常又来了，在压力山大的情况下，考虑从架构角度调整，不能只从具体的技术点去优化了，在做架构调整时，考虑到我们架构重构的初衷，提高数据的实时性，尽量让计算任务实时化，但重构过程中要考虑现有业务的过渡，所以架构必须支持实时与离线的需求，结合这些需求，在数据实时清洗平台（ETL）后加了一层数据缓存重用层（kafka），也就是经过数据实时清洗平台后的数据还是写入 kafka 集群，由于 kafka 支持重复消费，所以同一份数据可以既满足实时计算也满足离线计算，从上面的整体技术架构也可以看出，如图 12 所示。

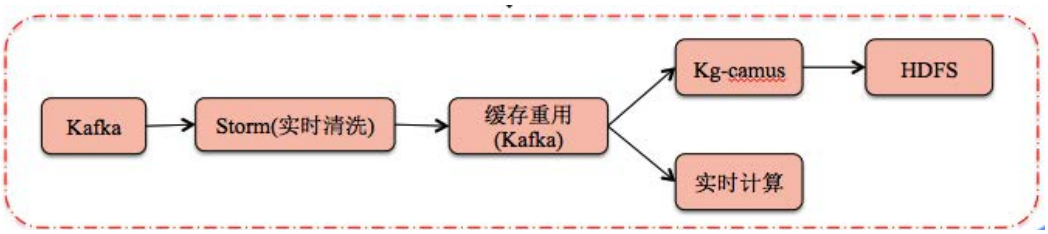


图 12

KG-Camus 组件也是基于架构调整后，重新实现了一套离线消费 Kafka 集群数据的组件，此组件是参考 LinkedIn 的 Camus 实现的。此方式，使数据消费模式由原来的推方式改为拉模式了，不用维持 HDFS Client 的长连接等功能了，直

接由作业调度系统每隔时间去拉一次数据，不同的业务可以设置不同的时间间隔，从此架构调整上线后，基本没有类似的异常出现了。

这个坑，是我自己给自己挖的，导致我们的重构计划延期 2 个月，主要原因是由最初技术预研究测试不充分所导致。

3、开源组件的坑

由于整个数据平台涉及到的开源组件很多，踩过的坑也是十个手指数不过来。

1）当我们的行为数据全量接入到 Kafka 集群（几百亿 / 天），数据采集网卡出现大量连接超时现象，但万兆网卡进出流量使用率并不是很高，只有几百 Mbit/s，经过大量的测试排查后，调整以下参数，就是顺利解决了此问题。调整参数后网卡流量如图 13 所示。

a)num.network.threads（网络处理线程数）值应该比 cpu 数略大。

b)num.io.threads（接收网络线程请求并处理线程数）值提高为 cpu 数两倍。

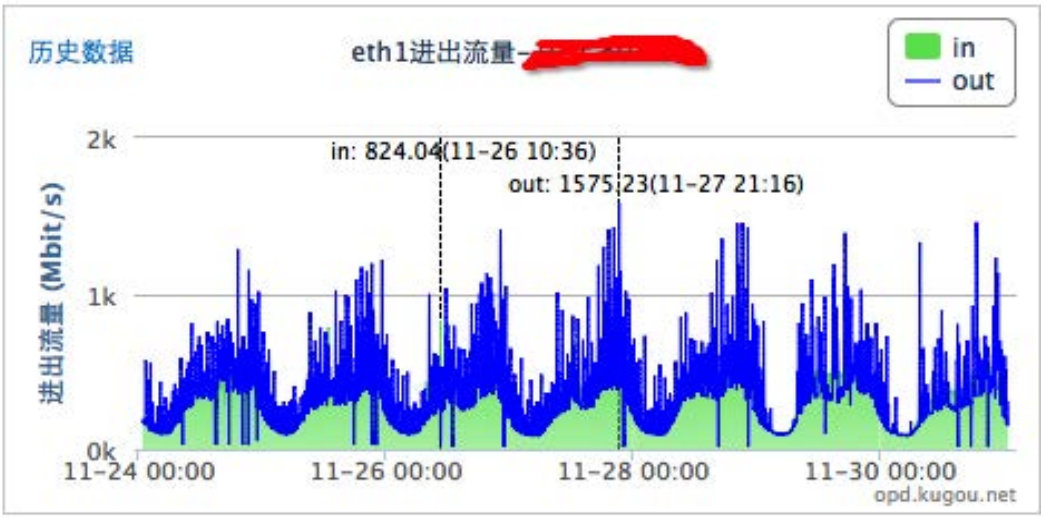


图 13

2）在 hive0.14 版本中，利用函数 ROW_NUMBER() OVER 对数据进行数据处理后，导致大量的作业出现延时很大的现象，经异常排查后，发现在数据记录数没变的情况，数据的存储容量扩大到原来的 5 倍左右，导致 MapReduce 执行很慢造成的。改为自己实现类似的函数后，解决了容量扩大为原来几倍的现象。说到

这里，也在此请教读到此处的读者一个问题，在海量数据去重中采用什么算法或组件进行比较合适，既能高性能又能高准确性，有好的建议或解决方案可以加happyjim2010 微信私我。

3）在业务实时监控系统中，用 OpenTSDB 与实时计算系统（storm）结合，用于聚合并存储实时 metric 数据。在这种实现中，通常需要在实时计算部分使用一个时间窗口（window），用于聚合实时数据，然后将聚合结果写入 tsdb。但是，由于在实际情况中，实时数据在采集、上报阶段可能会存在延时，而导致 tsdb 写入的数据不准确。针对这个问题，我们做了一个改进，在原有 tsdb 写入 api 的基础上，增加了一个原子加 api。这样，延迟到来的数据会被叠加到之前写入的数据之上，实时的准确性由于不可避免的原因（采集、上报阶段）产生了延迟，到最终的准确性也可以得到保证。另外，添加了这个改进之后，实时计算端的时间窗口就不需要因为考虑延迟问题设置得比较大，这样既节省了内存的消耗，也提高了实时性。

四、后续持续改进

数据存储（分布式内存文件系统（Tachyon）、数据多介质分层存储、数据列式存储）、即席查询（OLAP）、资源隔离、数据安全、平台微观层面监控、数据对外服务等。

王劲，目前就职酷狗音乐，大数据架构师，负责酷狗大数据技术规划、建设、应用。 11 年的 IT 从业经验，2 年分布式应用开发，3 年大数据技术实践经验，主要研究方向流式计算、大数据存储计算、分布式存储系统、NoSQL、搜索引擎等。

经典大数据架构案例 2：
Airbnb 的大数据平台架构

作者 **侠天**

Airbnb 成立于 2008 年 8 月，拥有世界一流的客户服务和日益增长的用户社区。随着 Airbnb 的业务日益复杂，其大数据平台数据量也迎来了爆炸式增长。

本文为Airbnb 公司工程师 James Mayfield 分析的 Airbnb 大数据平台构架，提供了详尽的思想和实施。

一、大数据架构背后的哲理

Airbnb 公司提倡数据信息化，凡事以数据说话。收集指标，通过实验验证假设、构建机器学习模型和挖掘商业机会使得 Airbnb 公司高速、灵活的成长。

经过多版本迭代之后，大数据架构栈基本稳定、可靠和可扩展的。本文分享了 Airbnb 公司大数据架构经验给社区。后续会给出一系列的文章来讲述分布式架构和使用的相应的组件。James Mayfield 说，“我们每天使用着开源社区提供的优秀的项目，这些项目让大家更好的工作。我们在使用这些有用的项目得到好处之后也得反馈社区。”

下面基于在 Airbnb 公司大数据平台架构构建过程的经验，给出一些有效的观点。

多关注开源社区：在开源社区有很多大数据架构方面优秀的资源，需要去采用这些系统。同样，当我们自己开发了有用的项目也最好回馈给社区，这样会良

性循环。

多采用标准组件和方法：有时候自己造轮子并不如使用已有的更好资源。当凭直觉去开发出一种“与众不同”的方法时，你得考虑维护和修复这些程序的隐性成本。

确保大数据平台的可扩展性：当前业务数据已不仅仅是随着业务线性增长了，而是爆发性增长。我们得确保产品能满足这种业务的增长。

多倾听同事的反馈来解决问题：倾听公司数据的使用者反馈意见是架构路线图中非常重要的一步。

预留多余资源：集群资源的超负荷使用让我们培养了一种探索无限可能的文化。对于架构团队来说，经常沉浸在早期资源充足的兴奋中，但 Airbnb 大数据团队总是假设数据仓库的新业务规模比现有机器资源大。

二、大数据架构预览

图 1 是大数据平台架构一览图。

Airbnb 数据源主要来自两方面：数据埋点发送事件日志到 Kafka；MySQL 数据库 dumps 存储在 AWS 的 RDS，通过数据传输组件 Sqoop 传输到 Hive“金”集群（其实就是 Hive 集群，只是 Airbnb 内部有两个 Hive 集群，分别为“金”集群和“银”

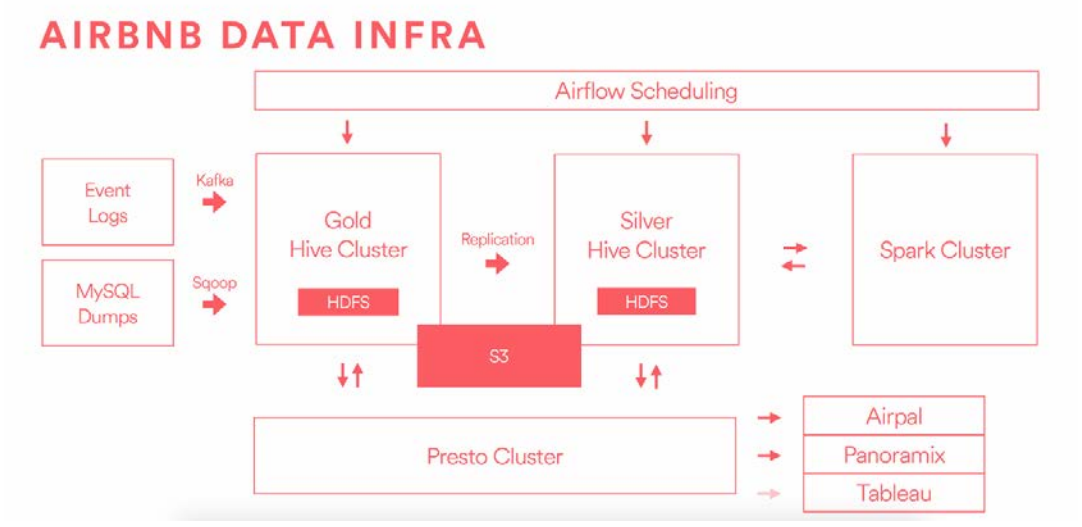


图 1

集群，具体分开两个集群的原因会在文章末尾给出。）。

包含用户行为以及纬度快照的数据发送到 Hive“金”集群存储，并进行数据清洗。这步会做些业务逻辑计算，聚合数据表，并进行数据校验。

在以上架构图中，Hive 集群单独区分“金”集群和“银”集群大面上的原因是为了把数据存储和计算进行分离。这样可以保证灾难性恢复。这个架构中，“金”集群运行着更重要的作业和服务，对资源占用和即席查询可以达到无感知。“银”集群只是作为一个产品环境。

“金”集群存储的是原始数据，然后复制“金”集群上的所有数据到“银”集群。但是在“银”集群上生成的数据不会再复制到“金”集群。你可以认为 “银”集群是所有数据的一个超集。由于 Airbnb 大部分数据分析和报表都出自“银”集群，所以得保证“银”集群能够无延迟的复制数据。更严格的讲，对于“金”集群上已存在的数据进行更新也得迅速的同步到“银”集群。集群间的数据同步优化在开源社区并没有很好的解决方案，Airbnb 自己实现了一个工具，后续文章会详细的讲。

在 HDFS 存储和 Hive 表的管理方面做了不少优化。数据仓库的质量依赖于数据的不变性（Hive 表的分区）。更进一步，Airbnb 不提倡建立不同的数据系统，也不想单独为数据源和终端用户报表维护单独的架构。以以往的经验看，中间数据系统会造成数据的不一致性，增加 ETL 的负担，让回溯数据源到数据指标的演化链变得异常艰难。Airbnb 采用 Presto 来查询 Hive 表，代替 Oracle、Teradata、 Vertica、 Redshift 等。在未来，希望可以直接用 Presto 连接 Tableau。

另外一个值得注意的几个事情，在架构图中的 Airpal，一个基于 Presto，web 查询系统，已经开源。Airpal 是 Airbnb 公司用户基于数据仓库的即席 SQL 查询借口，有超过 1/3 的 Airbnb 同事在使用此工具查询。任务调度系统 Airflow ，可以跨平台运行 Hive，Presto，Spark，MySQL 等 Job，并提供调度和监控功能。Spark 集群时工程师和数据分析师偏爱的工具，可以提供机器学习

和流处理。S3 作为一个独立的存储，大数据团队从 HDFS 上收回部分数据，这样可以减少存储的成本。并更新 Hive 的表指向 S3 文件，容易访问数据和元数据管理。

三、Hadoop集群演化

Airbnb 公司在今年迁移集群到“金和银”集群。为了后续的可扩展，两年前迁移 Amazon EMR 到 EC2 实例上运行 HDFS，存储有 300 TB 数据。现在，Airbnb 公司有两个独立的 HDFS 集群，存储的数据量达 11PB。S3 上也存储了几 PB 数据。

下面是遇到的主要问题和解决方案。

A) 基于 Mesos 运行 Hadoop

早期 Airbnb 工程师发现 Mesos 计算框架可以跨服务发布。在 AWS c3.8xlarge 机器上搭建集群，在 EBS 上存储 3TB 的数据。在 Mesos 上运行所有 Hadoop、 Hive、Presto、 Chronos 和 Marathon。

基于 Mesos 的 Hadoop 集群遇到的问题：

- Job运行和产生的日志不可见；
- Hadoop集群健康状态不可见；
- Mesos只支持MR1；
- task tracker连接导致性能问题；
- 系统的高负载，并很难定位；
- 不兼容Hadoop安全认证Kerberos。

解决方法：不自己造轮子，直接采用其它大公司的解决方案。

B) 远程读数据和写数据

所有的 HDFS 数据都存储在持久性数据块级存储卷（EBS），当查询时都是通过网络访问 Amazon EC2。Hadoop 设计在节点本地读写速度会更快，而现在的部署跟这相悖。

Hadoop 集群数据分成三部分存储在 AWS 一个分区三个节点上，每个节点都在不同的机架上。所以三个不同的副本就存储在不同的机架上，导致一直在远程的读数据和写入数据。这个问题导致在数据移动或者远程复制的过程出现丢失或者崩溃。

解决方法：使用本地存储的实例，并运行在单个节点上。

C) 在同构机器上混布任务

纵观所有的任务，发现整体的架构中有两种完全不同的需求配置。Hive/Hadoop/HDFS 是存储密集型，基本不耗内存和 CPU。而 Presto 和 Spark 是耗内存和 CPU 型，并不怎么需要存储。在 AWS c3.8xlarge 机器上持久性数据块级存储卷（EBS）里存储 3 TB 是非常昂贵的。

解决方法：迁移到 Mesos 计算框架后，可以选择不同类型的机器运行不同的集群。比如，选择 AWS c3.8xlarge 实例运行 Spark。AWS 后来发布了“D 系列”实例。从 AWS c3.8xlarge 实例每节点远程的 3 TB 存储迁移数据到 AWS d2.8xlarge 4 TB 本地存储，这给 Airbnb 公司未来三年节约了上亿美元。

D) HDFS Federation

早期 Airbnb 公司使用 Pinky 和 Brain 两个集群联合，数据存储共享，但 mappers 和 reducers 是在每个集群上逻辑独立的。这导致用户访问数据需要在 Pinky 和 Brain 两个集群都查询一遍。并且这种集群联合不能广泛被支持，运行也不稳定。

解决方法：迁移数据到各 HDFS 节点，达到机器水平的隔离性，这样更容易容灾。

E) 繁重的系统监控

个性化系统架构的严重问题之一是需要自己开发独立的监控和报警系统。Hadoop、Hive 和 HDFS 都是复杂的系统，经常出现各种 bug。试图跟踪所有失败的状态，并能设置合适的阈值是一项非常具有挑战性的工作。

解决方法：通过和大数据公司 Cloudera 签订协议获得专家在架构和运维这

些大系统的支持。减少公司维护的负担。Cloudera 提供的 Manager 工具减少了监控和报警的工作。

总结

在评估老系统的问题和低效率后进行了系统的修复。无感知的迁移 PB 级数据和成百上千的 Jobs 是一个长期的过程。作者提出后面会单独写相关的文章，并开源对于的工具给开源社区。

大数据平台的演化给公司减少大量成本，并且优化集群的性能，下面是一些统计数据：

- 磁盘读写数据的速度从70 - 150 MB / sec到400 + MB / sec。
- Hive任务提高了两倍的CPU时间
- 读吞吐量提高了三倍
- 写吞吐量提高了两倍
- 成本减少百分之七十

经典大数据架构案例 3：
美团大数据平台架构实践

作者 **谢语宸**

本文介绍了美团大数据平台的架构，然后讲解了整个平台演进的时间演进线，以及开发过程中遇到的一些挑战和应对策略，最后总结了对平台化的看法。

1.美团大数据平台的架构

1.1总体架构

图 1 是美团网数据体系组织架构图，上面每一个竖线都是数据开发业务线，下面是我所在的基础数据库团队， 最下面我们依赖美团云提供的一些虚拟机、

图 1

图 2

物理机、机房等基础设施，同时我们也协助美团云做了大数据云服务的产品探索。

1.2数据流架构

图 2 以数据流的架构角度介绍一下整个美团数据平台的架构，这是最恢复的架构图，最左边首先从业务流到平台，分别到实时计算，离线数据。

最下面支撑这一系列的有一个数据开发的平台，图 3 比较细，这是我们详细的整体数据流架构图。包括最左边是数据接入，上面是流式计算，然后是

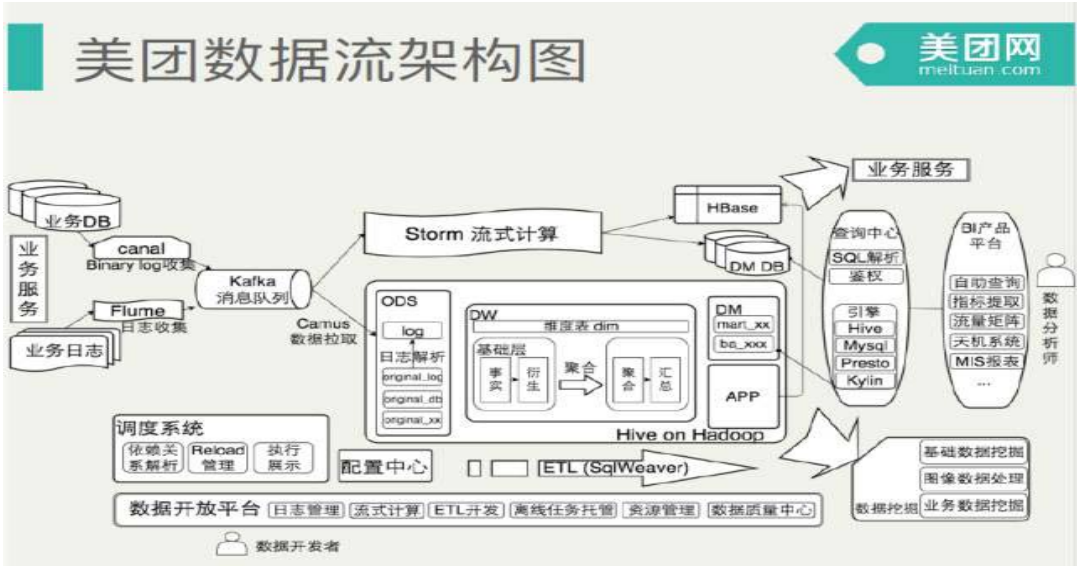


图 3

Hadoop 离线计算。

将图 3 左上角扩大来看，首先是数据接入与流式计算，电商系统产生数据分两个场景，一个是追加型的日志型数据，另外是关系型数据的维度数据。我们对于前一种是使用 Flume 比较标准化的，大家都在用的日志收集系统。最近使用了阿里开源的 Canal，之后有三个下游。所有的流式数据都是走 Kafka 这套流走的。

数据收集特性：

对于数据收集平台，日志数据是多接口的，可以打到文件里观察文件，也可以更新数据库表。关系型数据库是基于 Binlog 获取增量的，如果做数据仓库的话有大量的关系型数据库，有一些变更没法发现等等的情况，通过 Binlog 手段可以解决。通过一个 Kafka 消息队列集中化分发支持下游，目前支持了 850 以上的日志类型，峰值每秒有百万介入。

流式计算平台特性：

构建流式计算平台的时候充分考虑了开发的复杂度，基于 Storm。有一个在线的开发平台，测试开发过程都在在线平台上做，提供一个相当于对 Storm 应用场景的封装，有一个拓扑开发框架，因为是流式计算，我们也做了延迟统计和报警，现在支持了 1100 以上的实时拓扑，秒级实时数据流延迟。

图 4

图 5

这上面可以配置公司内部定的某个参数，某个代码，可以在平台上编译有调试。实时计算和数据接入部分就介绍到这儿，下面介绍一下离线计算（见图 4）。

离线计算：我们是基于 Hadoop 的数据仓库数据应用，主要是展示了对数据仓库分成的规划，包括原始数据接入，到核心数据仓库的基础层，包括事实和衍生事实，维度表横跨了聚合的结果，最右边提供了数据应用：一些挖掘和使用场景，上面是各个业务线自建的需求报表和分析库。

图 5 是离线数据平台的部署架构图，最下面是三个基础服务，包括 Yarn、HDFS、HiveMeta。不同的计算场景提供不同的计算引擎支持。如果是新建的公司，其实这里是有一些架构选型的。Cloud Table 是自己做的 HBase 分装封口。我们使用 Hive 构建数据仓库，用 Spark 在数据挖掘和机器学习，Presto 支持 Adhoc 上查询，也可能写一些复杂的 SQL。对应关系这里 Presto 没有部署到 Yarn，跟 Yarn 是同步的，Spark 是 on Yarn 跑。目前 Hive 还是依赖 Mapreduce 的，目前尝试着 Hive on tez 的测试和部署上线。

离线计算平台特性：

目前 42P+ 总存储量，每天有 15 万个 Mapreduce 和 Spark 任务，有 2500 万节点，支持 3 机房部署，后面跨机房一会儿会介绍，数据库总共 16K 个数据表，复杂度

图 6

还是蛮高的。

1.3数据管理体系

数据管理体系特性：

下面简单聊一下数据管理体系，这相当于主要面向数据开发者的操作经验，主要包括自研的调配系统，然后数据质量的监控，资源管理和任务审核一条开发配置中心等等，都是在数据管理体系的，下面会整合到整个的数据开放平台（见图 6）。

数据管理体系我们这边主要实现了几点，

第一点我们是基于 SQL 解析去做了 ETL 任务之间的自动解析。

基于资源预留的模式做了各业务线成本的核算，整体的资源大体是跑到 Yarn 上的，每个业务线会有一些承诺资源、保证资源，还可以弹性伸缩，里面会有一些预算。

我们工作的重点，对于关键性任务会注册 SLA 保障，并且包括数据内容质量，数据时效性内容都有一定的监控（见图 7）。

图 8 是解析出来的依赖关系，红色的是展示的一条任务，有一系列的上游。这是我们的资源管理系统，可以分析细到每个任务每时每刻的资源使用，可以聚

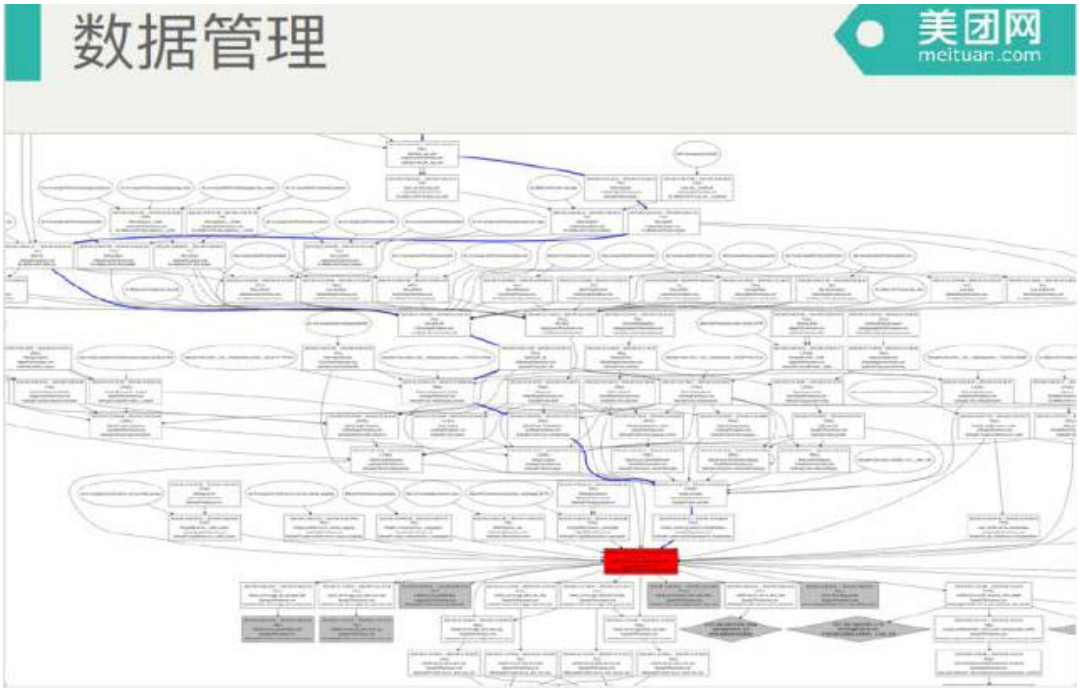


图 7

合，给每个业务线做成本核算。

这是对于数据质量管理中心，图比较小，上面可以写一些简单的 SQL，监控某一个表的数据结果是否符合我们业务的预期。下面是数据管理，就是我们刚刚提到的，对每个关键的数据表都有一些 SLA 的跟踪保障，会定期发日报，观察他们完成时间的一些变动（见图 8）。

图 8

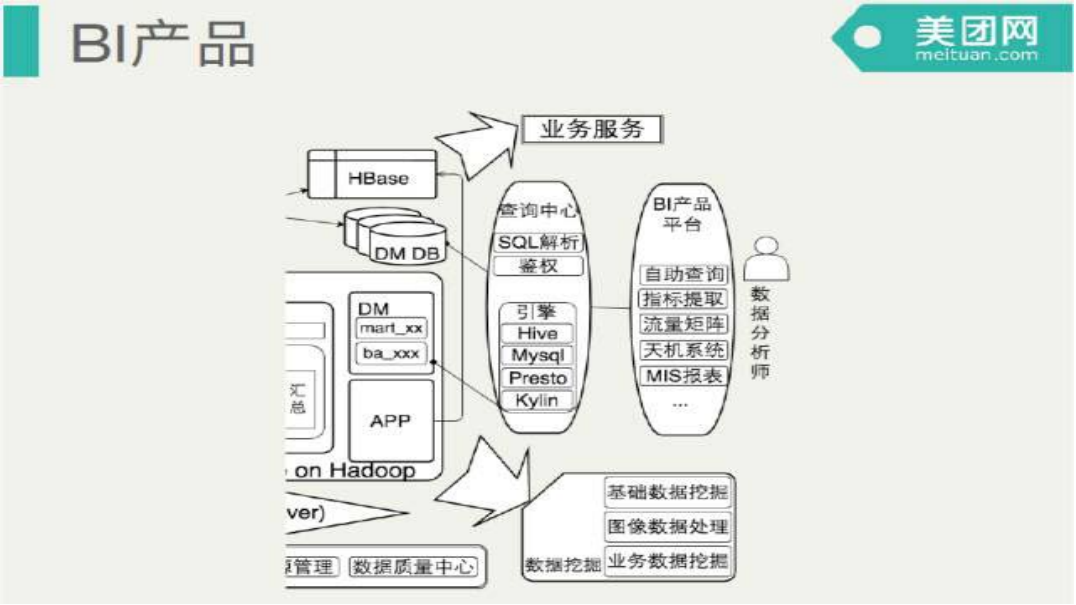


图 9

1.4 BI产品

图 9 是 BI 产品，数据应用平台化的场景。我们的查询主要是有一个查询中心来支持，包括 Hive，MySQL，Presto，Kylin 等等的引擎，在查询中心里面我们做 SQL 解析。前面是一系列的 BI 产品，大部分是自研的，面向用户可以直接写 SQL 的自主查询，并且看某一个指标，某一个时间段类似于 online 的分析数据产品，以及给老大们看的天机系统。

还有指标提取工具，其实跟商用 oneline 前端分析引擎设计是比较类似的，选取维度范围，还有适时的计算口径，会有一系列对维度适时的管理。数据内容数据表不够，还会配一些 dashboard。

我们开发了星空展示中心，可以基于前面指标提取结果，配置一系列的饼图、线图、柱状图，去拖拽，最后 build 出来一个 dashboard。

2.平台演进时间线

2.1 平台发展

下面聊一下整个数据平台发展的时间线。因为我是 2011 年加入美团的，美团刚刚建立一年左右。最开始 2011 年的时候，我们主要的数据统计都是基于手

写的报表，就是来一个需求我们基于线上数据建立一个报表页面，写一些表格。这里带来的严重的问题，首先是内部信息系统的工作状态，并不是一个垂直的，专门用做数据分析的平台。这个系统当时还是跟业务去共享的，跟业务的隔离非常弱，跟业务是强耦合的，而且每次来数据需求的时候我们都要有一些特殊的开发，开发周期非常长。

我们面对这个场景怎么办呢？我们做了一个目前来看还算比较好的决策，就是重度依赖 SQL。我们对 SQL 分装了一些报表工具，对 SQL 做了 etl 工具。主要是在 SQL 层面做一些模板化的工具，支持时间等变量。这个变量会有一些外部的参数传递进来，然后替换到 SQL 的行为。

我们在 2011 下半年引入了整个数据仓库的概念，梳理了所有数据流，设计整个数据体系。做完了数据仓库整体的构建，我们发现有整体的 ETL 被开发出来了。首先 ETL 都是有一定的依赖关系的，但是管理起来成本非常高。所以我们自研了一个系统，另外我们发现数据量越来越大，原来基于单机 MySQL 的数据解析是搞不定的，所以 2012 年我们上了四台 Hadoop 机器，后面十几台，到最后的几千台，目前可以支撑各个业务去使用。

2.2 最新进展

我们也做了一个非常重要的事就是 ETL 开发平台，原来都是基于 Git 仓库管理，管理成本非常高，当时跟个业务线已经开始建立自己数据开发的团队了。我们把他们开发的整个流程平台化，各个业务线就可以自建。之后我们遇到的业务场景需求越来越多，特别是实时应用。2014 年启动了实时计算平台，把原来原有关系型数据表全量同步模式，改为 Binlog 同步模式。我们也是在国内比较早的上了 Hadoop2.0 on Yarn 的改进版，好处是更好的激起了 Spark 的发展。另外还有 Hadoop 集群跨多机房，多集群部署的情况，还有 OLAP 保障，同步开发工具。

3.近期挑战和应对

3.1Hadoop多机房

Hadoop 多机房背景：

下面重点讲三个挑战还有应对策略，首先是 Hadoop 多机房。Hadoop 为什么要多机房部署呢？之前只有淘宝这样做。2015 年初我们被告知总机房架位只有 500 个节点，我们迁到的机房，主要还是机房合同发生了一些违约。我们沟通到新的离线机房需要在 9 月份交付，2015 年 6 月份我们需要 1000 个计算节点，12 月份的时候需要 1500 个计算节点，这肯定是不够的。那就要进行梳理，业务紧耦合，快速拆分没法支撑快速增长，而且数据仓库拆分会带来数据拷贝，数据传输成本的，这时候只能让 Hadoop 多机房进行部署。

我们思考了一下，为什么 Hadoop 不能多机房部署呢？

其实就两个问题。

一个是跨机房带宽非常小，而且跨机房带宽比较高，几十 G，可能给力的能上百 G，但是机房核心交换节点是超过这些的。而且 Hadoop 是天生的分布式系统，他一旦跨节点就一定会有跨机房的问题。

我们梳理了 Hadoop 运行过程中，跨节点的数据流程，基本上是三种。

首先是 APP 内部，就是任务内部的一些 Container 通信的网络交换，比较明确的场景就是 Map 和 educe 之间。

第二个是非 DataNode 本地读取，如果跨机房部署读数据就是跨机房的，带宽量非常大。

第三个写入数据的时候要构建一个三节点的 pipeline，可能是跨机房的，就要带来很多数据流量。

Hadoop 多机房架构决策：

我们当时考虑到压力，先做多机房的方案再做 NameSpace，这跟淘宝方案有所差别。我们每个节点都有一个所属的机房属性，把这个东西维护起来，基本上也是基于网络段判断的。对于刚刚提到的第一个问题，我们的方案在 Yarn 队列上打一个机房的 tag，每个队列里面的任务只会在某一个机房里跑起来，这里要修改一下 Yarn fairscheduler 的代码的。

第二个是基于 HDFS 修改了 addBlock 策略，只返回 client 所在机房的 DataNode 列表，这样写入的时候 pipeline 就不会有跨机房，读取也会优先选取 clinet 所在的机房。还有其他的场景会跨机房，比如说 Balancer 也是节点之间做数据迁移的。最终我们还做了一件事，就是 Balancer 是直接 DataNode 沟通，有通道的，我们是直接构造了 Block 文件分布工具。

Hadoop 多机房结构效果见图 10。

效果上看，左边是 2015 年 3 月份节点数，300 多，2016 年 3 月份是 2400 多，中间不同的段是每个机房当时承载的节点数。这时候我们只有一个机房了，因为我们整个跨机房，多机房的方案是为了配合一个临时的状态，所以它方案前面通过 Balancer 模块的接口，把所有数据最终都搬迁到了大的离线计算机房。

Hadoop 多机房架构特点：

做这个架构的时候，我们设计的时候主要考虑第一代码改动要小，因为当时我们团队没有那么深的对 Hadoop 代码的掌控，我们要保证设计出来的结果，对于 Hadoop 原生逻辑的影响范围是可控的；第二个是能快速开发，优先顶住节点资源分布不够的问题；第三个整个迁移过程是业务全透明的，只要在他数据读取

图 10

之前把块分布到我希望任务所调动的机房就可以了。

3.2 任务托管和交互式开发

任务托管和交互式开发背景：

我们原来的方式是给业务线去布一些开源原生Hadoop和Spark的Client的。

在本机要编写代码和编译，拷到线上的执行节点，因为要有线上的认证。

并且要部署一个新的执行节点的时候，要给我们提申请，分配虚拟机，key和client，这个管理成本非常高。

而且同一个团队共享一个虚拟机开发总会遇到一个问题，某个虚拟机会被内存任务占满，要解决这个问题。

而且由于在Spark发展的过程中，我们会持续地给业务提供Spark技术支持这样一个服务。如果大家写代码运行失败了，他们没有那么强的debug能力，当我们上手帮他们debug的时候，首先编译环境、执行环境，编译代码内容我们都没法第一时间获取，这个沟通成本是非常高的。同时在推Spark的时候，我们发现它的开发效率非常高，学习尝试的成本也是非常高的。那怎么办呢？

任务托管和交互式开发架构决策：

为了解决学习成本高的问题，我们做了两个事。

一个是任务托管平台，将任务的代码编译打包、执行、测试还有最终上线跑，都统一在一个平台进行管理。

另一个是我们推动了交互式开发工具，当时调研了ipthon notebook + spark和zeppelin，最后选择了zeppelin，觉得比较成熟。基于后者开发，修复了一系列bug，补充登陆认证。效果是任务托管平台，本机编写代码，提交代码到公司公有的地址上。在这个平台界面，平台界面进来都不是必须的了，还进行了本机的任务行，提交一个任务，开始在平台上统一测试，统一执行，最后还可以基于这个配置到我们刚刚说到的自研调度系统。

交互式开发目前可能都需要二次开发才能做起来，但是值得尝试。业务线用它的话主要是两个场景，第一个场景是要分析、调研一些数据。原来我们提

供 adhoc 的 Sql 的查询接口其实并不一定能满足他的需求，他要查查接口有一些 sql 查询复杂数据，如果想用 spark 每次用 spark 都要编译或者用 Spark 管理起来非常不直观。

另外有一些先行 Spark 尝试者写了一些 Spark 的应用，这些应用如何让其他同学也能看到，也能对他进行学习和理解，并且能支持他自己构建自己的应用场景呢？也可以通过这么一个平台化的代码、结果，对应展示的平台来解决他们交互的问题。

3.3 OLAP引擎

OLAP 引擎的需求特点：

最后聊一下在 OLAP 引擎部分的探索，大概 2015 年末的时候，我们开始关注到业务的数据集市，数据量已经非常大了，而且包括维度，表的大小、复杂度都增长的非常快。这些业务也比较崩溃，MySQL 和 HBase 都会做一些特殊的方法来支持。我们调研了一下需求，普遍说是要支持亿级别的事实，指标的话每个 cube 数据 立方体要有 50 个以内，要支持取值范围在千万级别维度 20 个以内类别。

查询请求，因为数据集市一般都是提供给销售管理团队去看业绩，对延迟要求比较高，对我们当时 TP99，前 99% 查询要小于 3 秒钟。

有多种维度组合聚合查询，因为要上转下转对业务进行分析。

还有一个特点，就是对去重的指标要求比较精确，因为有些涉及到业绩的指标比如团购单，去重访问用户数如果有偏差会影响到业绩的预算。

OLAP 引擎可能的方案：

当时考虑到了业界可能的方案，

一个是原来推荐的使用方法，就是 Presto、hive、Spark on ORCFile，这是最早的方案。

另外有先行的业务方案，基于 hive grouping set 的功能，把 grouping set 按不同维度组合去做聚合，然后形成一个大表，导到 HBase 里，HBase 按需

做二级索引的方案，这其实还是有一些瓶颈的。

还有社区里兴起的 Druid、Elasticsearch 还有 Kylin 这些项目，我们面临这样的场景思路是这样的。首先直观的看，考虑稳定性、成熟度，以及团队对这个产品可能的掌控程度，还有社区的活跃度，我们优先尝试 Kylin。我们团队有两个 Kylin contributors。

OLAP 引擎探索思路：

由于前面有这样多的解决方案，我们怎么保证我们选的解决方案是靠谱的呢？我们基于 dpch 构建了一个 Star Schema Benchmark 构造了 OLAP 场景和测试数据；我们用这一套数据结构和数据内容对不同的引擎进行测试，看它的表现和功能性，满足的情况。并且推动的过程中持续的分享我们调研和压缩的进展，优先收集他们实际业务场景需求之后，再回过头来改进数据集市的需求，更适合业务线需求，图 11 就是 Kylin 的界面。

具体它提供一个界面声明你的维度、事实，有哪些指标，这些指标会被怎样聚合，会生成 Mapreduce 任务，出来的结果会按照设计进行压缩，导到 HBase 里面。他还提供一个 SQL 引擎，会转成 HBase 上查询，把结果捞出来，总体来讲还

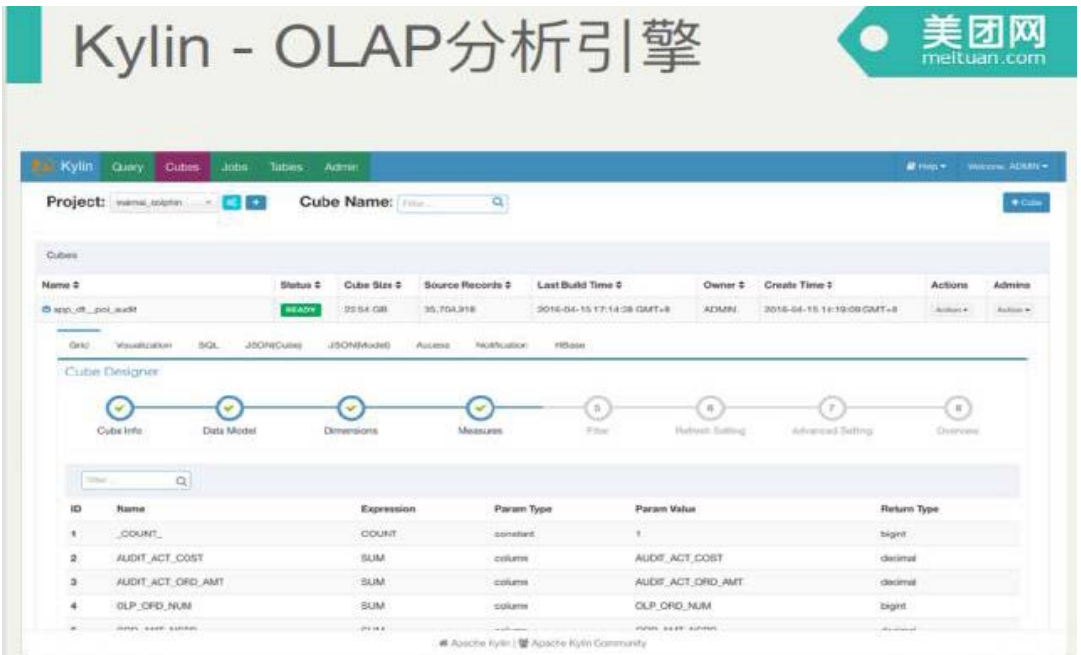


图 11

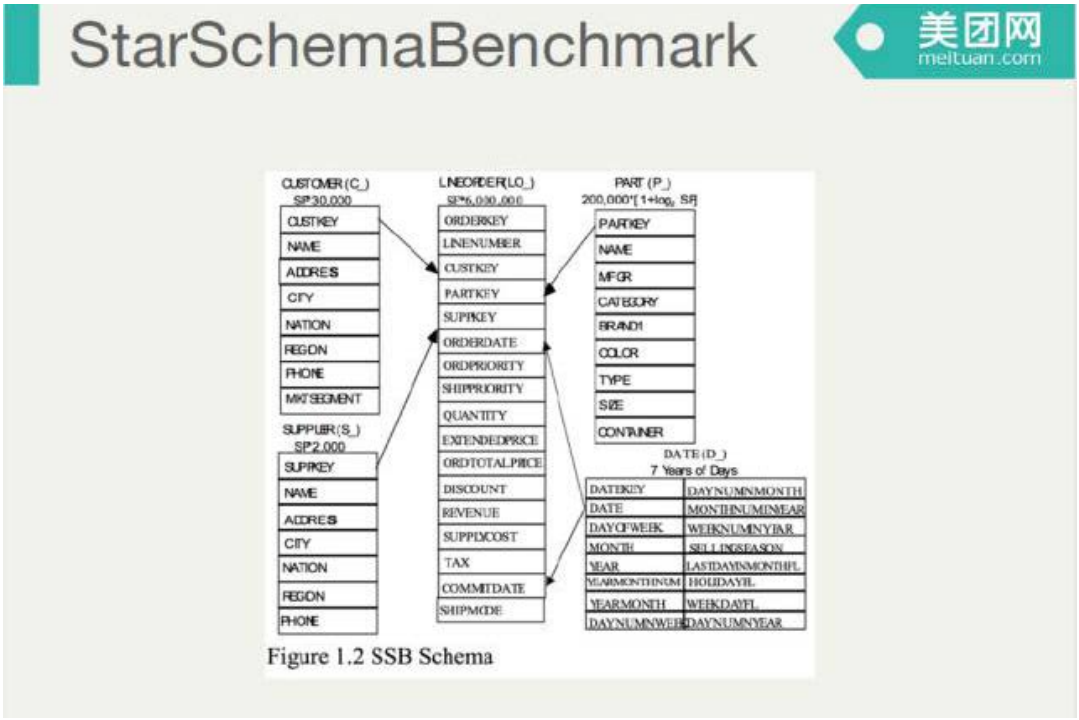


Figure 1.2 SSB Schema

图 12

是蛮成熟的。

图 12 是 StarSchemaBenchmark，一张大的事实表，有很多维度挂在上面，我们做了很多不同数据量级的参照，也参照了现实的数据。

OLAP 引擎目前进展：

目前进展的话，我们完成了 Presto、Kylin1.3、Kylin1.5，Druid 测试。这个确实比 Kylin 好一些，但是有特殊场景，天生不支持 SQL 接口，所以不会重度使用。

我们拿 Kylin 支持了某个 BI 项目 7 个数据立方体，数据立方体基本上是一个事实，带一系列维度，是某一个场景下的分析。

业务开发周期做一系列的聚合表，梳理聚合成绩，维护这些聚合成绩 7 天缩短到一天。

线上实际跑的数据有 3 亿行数据，TP95% 查询响应时间在 1S 内，TP99 是 3 秒内；支撑外卖团队日查询量 2 万。由于这是外卖的销售团队去看，他们量非常大。

4.平台化思路总结

4.1 平台的价值

最后聊一下做了这么多年数据平台，对于数据平台的思考。我觉得平台不管是不是数据平台，作为一个平台的团队，核心价值其实就是这三个。

第一个是对重复的事情，这一个平台团队做精做专，而且重复的事情只做一次，减少投入。

另外统一化，可以推一些标准，推一些数据管理的模式，减少业务之间的对接成本，这是平台的一大价值。

最重要的是为业务整体效率负责，包括开发效率、迭代效率、维护运维数据流程的效率，还有整个资源利用的效率，这都是要让业务团队对业务团队负责的。无论我们推什么事情，第一时间其实站在业务的角度要考虑他们的业务成本。

4.2 平台的发展

如果才能发展成一个好的平台呢？

我理解是这三点：

首先支持业务是第一位的，如果没有业务我们平台其实是没法继续发展的。

第二是与先进业务同行，辅助并沉淀技术。在一个所谓平台化的公司，有多个业务线，甚至各个业务线已经是独立的情况下，必定有一些业务线是先行者，他们有很强的开发能力、调研能力，我们的目标是跟这些先行业务线同行。我们跟他们一起走的过程中，一方面是辅助他们，能解决一系列的问题。比如说他们有突发的业务需求，遇到问题我们来帮助解决。

第三是设立规范，用积累的技术支撑后发业务。就是跟他们一起前进的过程中，把一些经验、技术、方案、规范慢慢沉淀下来。对于刚刚新建的业务线，或者发展比较慢的业务线，我们基本策略是设置一系列的规范，跟优先先行业务线积累去支撑后续的业务线，以及功能开发的时候也可以借助。保持平台团队对业务的理解。

4.3 关于开源

最后聊一下开源，刚刚也提到了我们同时对开源有一些自己需求的改进和重构，但是同时又一些产品是我们直接开源的来用的，比如说 zeppelin，Kylin。

我们的策略是持续关注，其实也是帮业务线做前瞻性调研，他们团队每天都在看数据，看新闻，他们会讲新出的一个项目你们怎么推，你们不推我们推了，我们可能需要持续关注，设计一系列的调研方案，帮助这些业务去调研，这样调研这个事情我们也是重复的事情只干一次。

如果有一些共性 patch 的事情，特别一些 bug、问题内部也会有一个表共享，内部有大几十个 patch。选择性的重构，最后才会大改，特别在选择的时候我们起来强调从业务需求出发，理智的进行选型权衡，最终拿出来的方案是靠谱能落地实施的方案，我的分享就到这里，谢谢大家。

观看嘉宾演讲视频

经典大数据架构案例 4：
电商卷皮 BI 的实践演进和架构体系

作者 **柴楹**

BI&大数据是什么

首先我们来聊一下 BI 和大数据。BI 和大数据到底有什么关系和不同。

BI（Business Intelligence）即商业智能技术，主要包括有三方面的技术：

- 数据仓库（Data Warehousing）
- 联机分析处理（OLAP:On-Line Analytical Processing）
- 数据挖掘（Data Mining）

以三种技术的整合为基础，建立企业数据中心和业务分析模型，以提高企业获取经营分析信息的能力，从而提高企业经营和决策的质量和效率（见图 1）。

数据仓库（Data Warehouse）是一个面向主题的（Subject Oriented）、集成的（Integrate）、相对稳定的（Non-Volatile）、反映历史变化（Time Variant）的数据集合，用于支持管理和决策。OLAP: On-Line Analytical Processing 使分析人员、管理人员能够从多种角度对从原始数据中转化出来的、能够真正为用户所理解的、并真实反映数据维特性的信息，进行快速、一致、交互地访问，从而获得对数据的更深入了解的一类软件技术。（OLAP 委员会的定义）。Data Mining 是通过数学模型发现隐藏的、潜在的规律，以辅助决策。

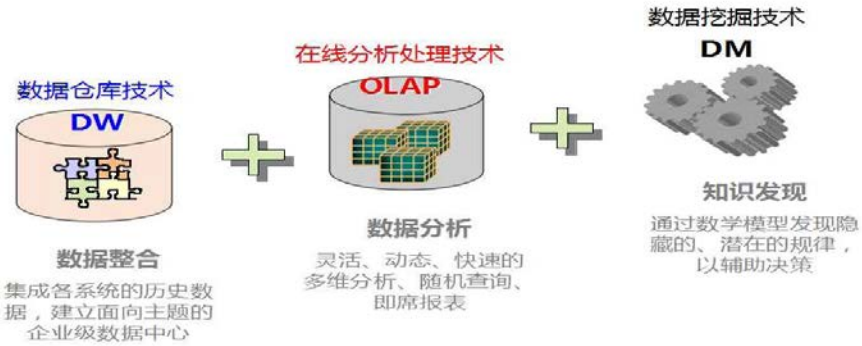


图 1

传统 BI 和数据仓库大约是 98-99 年从国外进入中国，经过十几年的发展，更多的是做企业级的数据中心，主要应用在电信业和银行业，需求更多的是做报表和进行一些分析等等。传统的 BI 主要想实现从宏观到微观、从广度到深度、从定量到定性各种层次的决策分析。

大数据是什么？通俗的讲，就是体量特别大的数据集，这个数据集大到无法用传统的数据库工具或者分析工具进行处理。大数据主要有三个特点：

第一，数据体量巨大。从 TB 级别，跃升到 PB 级别。

第二，数据类型繁多，例如网络日志、视频、图片、地理位置信息等等各种结构化非结构化的数据。

第三，处理速度快。1 秒定律。最后这一点也是和传统的数据挖掘技术有着本质的不同（见图 2）。

一般的大数据平台都有几个过程：数据采集、数据存储、数据处理和数据展现，当然处理的数据也提供做分析和挖掘。

大数据在 08 年的时候还没有很多人提及，但是随着互联网的快速发展，技术的变革，大数据越来越流行，现在也是逢技术论坛，必谈大数据。

大数据同传统 BI 比较，多了一个专门的数据采集阶段，主要是因为数据种类多，数量大，从结构化的数据到非结构化的数据。但是其存储、处理及可视化

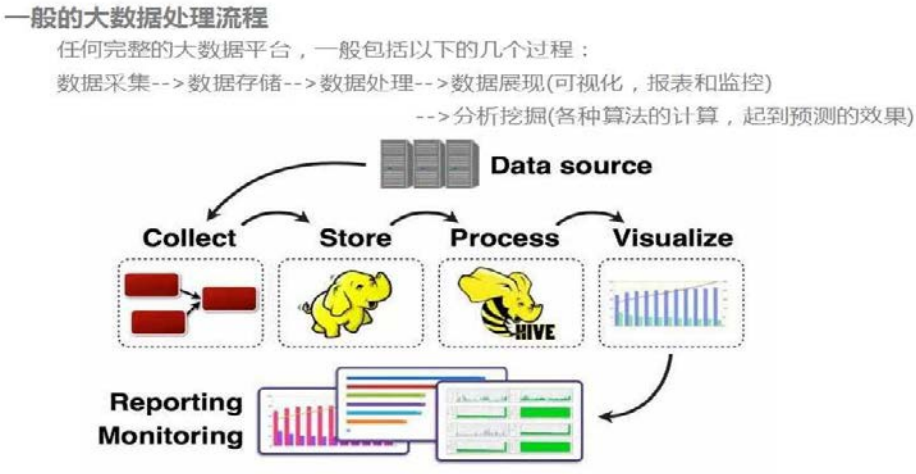


图 2

的思想等都和传统 BI 如出一辙。

总结一下，大数据是从 BI 中发展来的，但现在 BI 也借助着互联网和大数据的快速发展，有了第二春，因为无论数据方面，还是技术方面，大数据都给 BI 提供了翔实的基础。

以上是抛砖引玉的给大家介绍一下 BI 和大数据，具体的我就不展开了，有兴趣的同学可以自己去多了解一下。下面我来介绍一下我们卷皮的 BI 体系。

卷皮BI长什么样

首先介绍一下我们卷皮 BI 的数据体系，分为四层（见图 3）。

第一层是基础平台层，包括 BI 所有的数据的接入，加工，等等；

第二层是数据服务层，主要给业务部门提供报表和 OLAP 分析系统、给分析师提供自助取数平台等等；

第三层是智慧运营层，主要是把数据以数据产品的方式渗透到业务部门的日常工作中，例如精细化的运营，针对不同的区域或者人群进行不同的运营策略；

第四层是决策支持。当然决策支持可以说是在数据服务层和智慧运营层都在做，因为也是以数据支撑每一个具体的业务决策。但是这里讲的第四层的决策更多是以重大决策为主。举个例子：公司选择区域扩张策略，或者仓库选址，还有

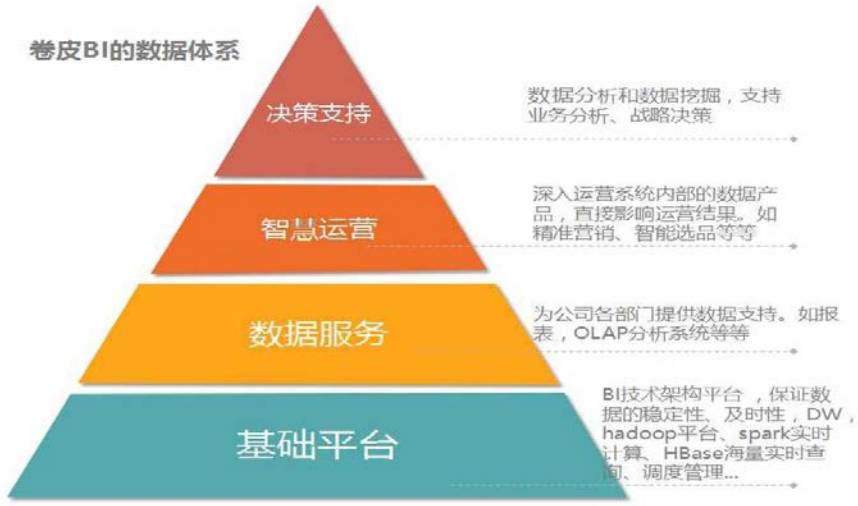


图 3

新业务模式探索等等方向性的决策。

目前我们BI团队处于第三层阶段，正在推进各项智慧运营数据产品的建设。

接下来介绍一下我们卷皮BI的架构体系。我们主要有五大基础平台（见图4）。

一、数据采集同步平台：负责接入所有的数据源，用户行为的数据是通过埋点直接生产到kafka，数据库之间的抽取用的阿里开源的datax，实时库的同步用也是阿里开源的otter，然后竞品数据是用爬虫平台采集来的。

二、实时计算平台：我们直接上的Spark Streaming，它直接去消费kafka中的数据。虽然Spark Streaming不是真正的流计算，而是高频率的批处理，没有storm的实时性好，但是目前秒级的延迟我们还是接受的，因为Scala语言开发起来更加简洁，而且Spark后续可以支撑更多，例如我们的挖掘就直接用的SparkR。其中还涉及一些内存计算我们用的是memcached和redis，实时数据计算的数据一般直接存储到hbase或者es里面，便于更快的检索。

三、离线计算平台：主要用的hadoop平台，Mysql里面有极少量的存储过程，当前DW全部都在HDFS上，Mysql更多存储的是为报表展示的数据集市类的表。

四、数据服务平台：主要是对外的平台，报表系统，即席查询，OLAP分析系统，数据分析和挖掘等，然后BI也会给公司其他业务研发团队提供各种数据支撑，统一都是走BI自己搭建的数据服务层。

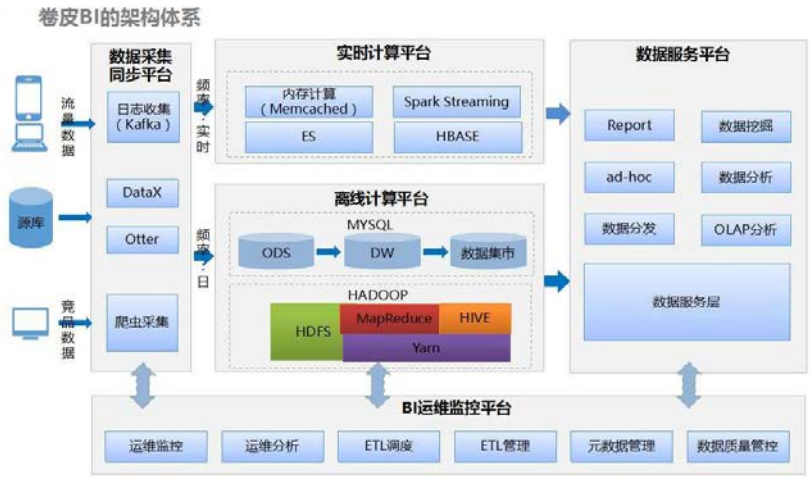


图 4

五、运维监控平台：调度系统用的阿里开源的 Zeus，然后针对我们自己的需求进行很多二次开发；日志收集分析用的 ELK；监控平台负责 BI 这边所有的硬件软件还有数据质量等等监控；当然这里还要做 BI 的元数据管理。

这五大技术平台是 BI 的物质基础，基于这些物质基础，才能继续产出我们的上层建筑：数据产品。

我们 BI 的产品体系主要有两条线，也就是两只脚走路。

先说一下数据服务线的数据产品，这部分产品主要是支撑公司内所有的数据需求，满足不同层次的人看数据的需要。因为这个也是 BI 的基础，基本的数据服务你满足，后面业务部门才能配合一起做其他智慧运营的数据产品。智慧运营线主要想将数据渗透到公司业务部门人员工作的每一个环节中，辅助业务部门人员能够更加好的做好运营工作。具体的应用有精准化营销系统、个性化的推荐系统、鹰眼的反欺诈系统和智能选品系统等。

以上就是我们卷皮 BI 的数据、架构和产品的体系。

我们做了哪些东西

下面介绍一下我们已经做了的三个数据产品（见图 5）。

第一是用户画像。卷皮是电商平台，我们必须要充分的了解我们的用户，所

图 5

以卷皮 BI 也基于自有的用户消费数据、行为数据，进行相应的算法模型去挖掘用户的特征，给用户打上各种标签。当然也接入一些外部的数据来验证我们的标签。目前的用户标签，主要分为四个方面：自然属性，兴趣偏好，消费特征，生命周期。

然后基于用户画像（见图 6），我们团队的精准化小组，就在做以下三个方面的事情。

1. 精准的营销：通过精准的 push 提升用户到达率；针对不同群体用户做专题活动；对于濒危用户进行挽留，等等。

图 6

图 7

2. 个性化的推荐：业内所说的千人千面，每个人专属的商品的排序；其他的推荐场景，例如猜你喜欢和热门推荐。但是对于第一次来的用户，没有任何行为信息，更多以热门推荐为主。目前我们也在做基于用户实时的浏览行为，进行实时的商品推荐。

3. 精准的服务：对于不同会员的等级进行差异化的服务，例如信用好的用户如果选择退货，那么我们可以先退钱后收货，但是对于信用等级不够高的用户，那么我们会收到货以后再退钱等；优化客服的服务，对于接入的客户，更加了解客户的信息，便于提高服务质量。

第二个，就是我们的鹰眼系统（见图 7），也就是反欺诈系统。目前定位是主要是实时的甄别异常订单。鹰眼系统主要做两方面的事情，识别坏人和识别坏

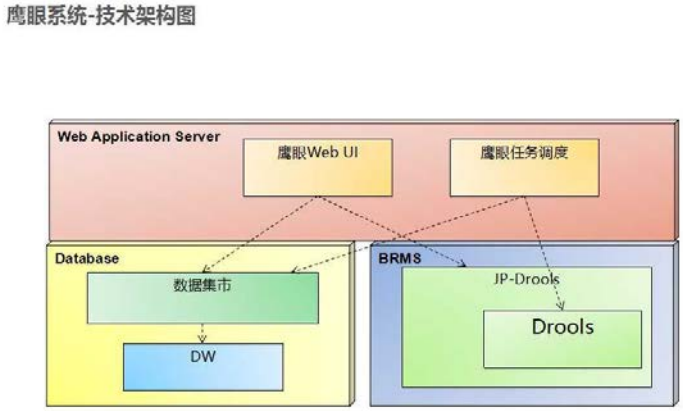


图 8

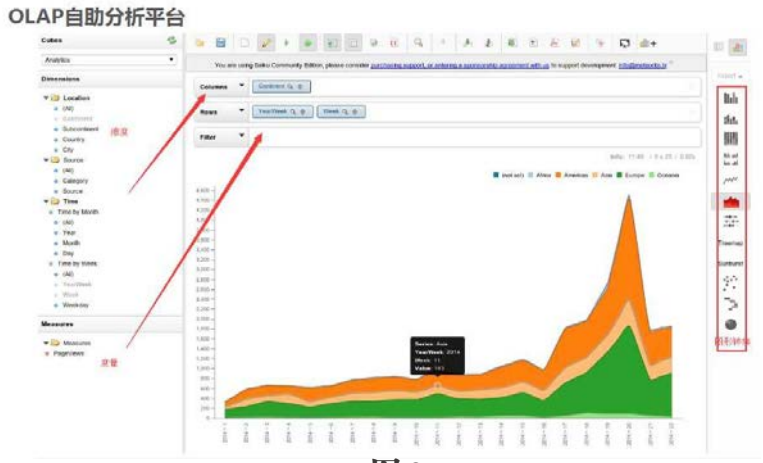


图 9

事。目前我们的鹰眼系统一共有 4 个子系统：鹰眼马甲系统、鹰眼售后系统、鹰眼订单甄别、鹰眼诚信系统（见图 8）。

鹰眼系统的核心模块是 BRMS（业务规则管理系统），基于规则引擎 (Drools)。工作人员可通过 Web UI 制定规则，形成规则库，每个规则都有个阈值。实时的数据结合数据集市的历史数据，在规则引擎里面进行判断，如果超出的规则的阈值，则进行相应的操作，如告警，转人工审核等。

鹰眼的 WebUI 是我们自己开发的界面，便于我们的业务运营人员，基于一些现有的指标来配置规则，调整阈值。JP-drools 是在 drools 我们在外面封装了一层，主要是为了做到分布式部署、历史库共享和规则的热部署（见图 9）。

最后这个产品是 OLAP 分析系统，图片是一个截图，左边这边有维度和度量，通过拖拽到中间的行或者列进行生成相应的表格，右边可以把表格的数据变成各种图形。业内这种类型的分析工具其实比较多，例如 Microstrategy，Tableau 等。但这些都是商业的，我们更多还是基于开源来做（见图 10）。

我们主要用了如下几个开源的项目：

Saiku 提供了一个多维分析的用户操作界面，可以通过简单拖拉拽的方式迅速生成报表，它的主要工作是根据事先配置好的 schema，将用户的操作转化成 MDX 语句提供给 Mondrian 引擎执行。

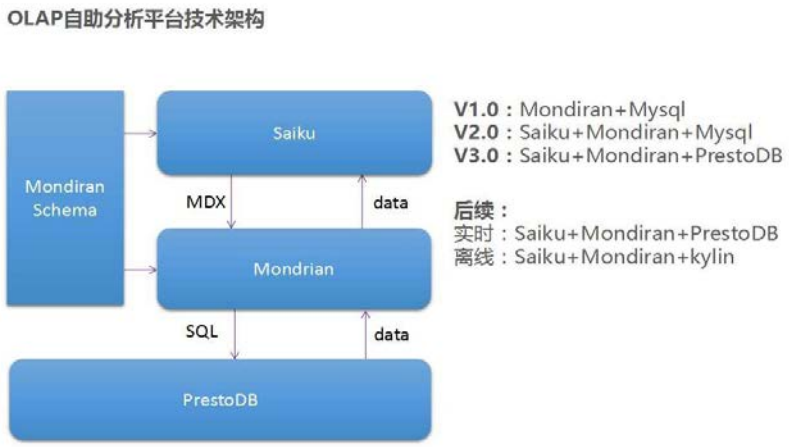


图 10

Mondrian 是一个 OLAP 分析的引擎，主要工作是根据事先配置好的 schema，将输入的多维分析语句 MDX（Multidimensional Expressions）翻译成目标数据库 / 数据引擎的执行语言（比如 SQL）。

Presto 是一个分布式 SQL 查询引擎， 它被设计为用来专门进行高速、实时的数据分析。它支持标准的 ANSI SQL，包括复杂查询、聚合（aggregation）、连接（join）和窗口函数（window functions)。

当前这个架构是我们第三个版本的架构。第一个版本我们是直接用的 Mondrian+Mysql，但是我们发现 Mondrian 的界面太丑了，所以在第二版加入了 Saiku。但是随着业务数据量的增加，Mysql 的查询性能很快就到瓶颈了，所以在第三个版本用 Presto 替代了 Mysql。

在这套架构里面 Saiku 提供了界面的支持，Mondrain 提供了 schema 到 MDX 的转换，并构建 SQL 语句，向 Prestodb 查询数据，Prestodb 执行查询任务，返回其结果，Saiku 显示结果，输出报表。整个 OLAP 系统我们需要关注 Saiku 的二次开发，Mondrain schema.xml 生成及其读取数据和维表方面的优化。

但是当前这个架构目前也逐渐遇到瓶颈，对于像具体到每一个用户成单路径的数据的分析时候查询还是需要比较久的时间，所以我们现在依然在调整，希望把 kylin 加入进来。kylin 是 apache 软件基金会的顶级项目，一个开源的分布

式多维分析工具。Kylin 通过预计算所有合理的维度组合下各个指标的值并把计算结果存储到 HBASE 中的方式，大大提高分布式多维分析的查询效率。Kylin 接收 sql 查询语句作为输入，以查询结果作为输出。对于可以离线分析的业务数据，可以用 kylin 的框架，而对于实时分析的业务数据还是可以用来 Presto 支持。

以上就是我们卷皮 BI 的一些经验的分享。最后送给大家一句话：数据本身不是最终价值，带有分析的数据，渗透到业务中，影响到决策才产生价值。

Q&A

Q1：查询 HBase 中的数据有没有用什么 SQL 引擎呢？有的话用的是什么 SQL 查询引擎？

A1：我们没有用什么 SQL 引擎，我们主要是靠 row-key 的设计。

Q2：hadoop 平台的部署是通过？ ambari 这些吗？

A2：我们使用 cloudera 的版本的。

Q3：老师好，能否大概讲解一下怎么根据用户画像做推荐，这里面用到什么技术点。

A3：主要还是数据挖掘的算法，有聚类，协同过滤，商品相似度之类的算法，不过针对不同的业务场景使用的算法不一样。技术上，我们是用的 sparkR。

Q4：我们现在 olap 目前正在使用 apache kylin,saiku 和 kykin 结合怎么样，有过调研没？

A4：Saiku 直接 +kylin 我们还没有用过，不过应该是 OK 的，因为 saiku 主要是界面展现。

Q5：BI 挖掘的用户画像和鹰眼系统，有什么离线指标来评价相关的数据质量？

A5：类似用户画像的性别，主要看两个方面，覆盖率和准确率，覆盖率提升了，也许准确率就会下降，后续我们可以持续跟踪用户的行为，或者进行一些实际的回访，来验证并优化我们的数据模型。鹰眼更多的是基于规则引擎做的。

Q6：数据 meta 管理是怎么做的？

A6：业务上我们对所有的数据指标口径进行统一，所有展示数据的地方都是一致的，然后对于变更等等流程都有一定的管理。

经典大数据架构案例 5：
解密 IFTTT 的数据架构

作者 **张天雷**

随着信息技术的发展，人们在日常生活和工作中都不可避免的要用到邮箱、聊天工具、云存储等网络服务。然而，这些服务很多时候都是单独运行的，不能很好的实现资源共享。针对该问题，IFTTT 提出了“让互联网为你服务”的概念，利用各网站和应用的开放 API，实现了不同服务间的信息关联。例如，IFTTT 可以把指定号码发送的短信自动转发邮箱等。为了实现这些功能，IFTTT 搭建了高性能的数据架构。近期，IFTTT 的工程师 Anuj Goyal 对数据架构的概况进行了介绍，并分享了在操作数据时的一些经验和教训。

在 IFTTT，数据非常重要——业务研发和营销团队依赖数据进行关键性业务决策；产品团队依赖数据运行测试 / 了解产品的使用情况，从而进行产品决策；数据团队本身也依赖数据来构建类似 Recipe 推荐系统和探测垃圾邮件的工具等；甚至合作伙伴也需要依赖数据来实时了解 Channel 的性能。鉴于数据如此重要，而 IFTTT 的服务每天又会产生超过数十亿个事件，IFTTT 的数据框架具备了高度可扩展性、稳定性和灵活性等特点。接下来，本文就对数据架构进行详细分析（见图 1）。

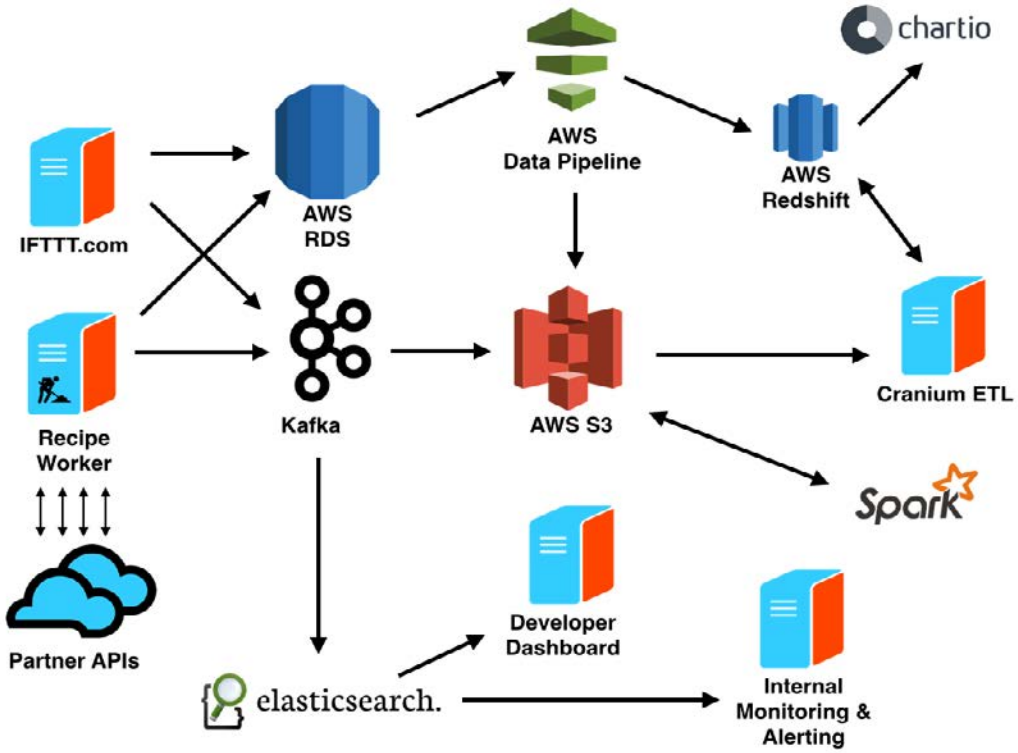


图 1

数据源

在 IFTTT，共有三种数据源对于理解用户行为和 Channel 性能非常重要。首先，AWS RDS 中的 MySQL 集群负责维护用户、Channel、Recipe 及其相互之间的关系等核心应用。运行在其 Rails 应用中的 IFTTT.com 和移动应用所产生的数据就通过 AWS Data Pipeline，导出到 S3 和 Redshift 中。其次，用户和 IFTTT 产品交互时，通过 Rails 应用所产生的时间数据流入到 Kafka 集群中。最后，为了帮助监控上百个合作 API 的行为，IFTTT 收集在运行 Recipe 时所产生的 API 请求的信息。这些包括反应时间和 HTTP 状态代码的信息同样流入到了 Kafka 集群中。

IFTTT的Kafka

IFTTT 利用 Kafka 作为数据传输层来取得数据产生者和消费者之间的松耦合。数据产生者首先把数据发送给 Kafka。然后，数据消费者再从 Kafka 读取数据。因此，数据架构可以很方便的添加新的数据消费者。

由于 Kafka 扮演着基于日志的事件流的角色，数据消费者在事件流中保留着

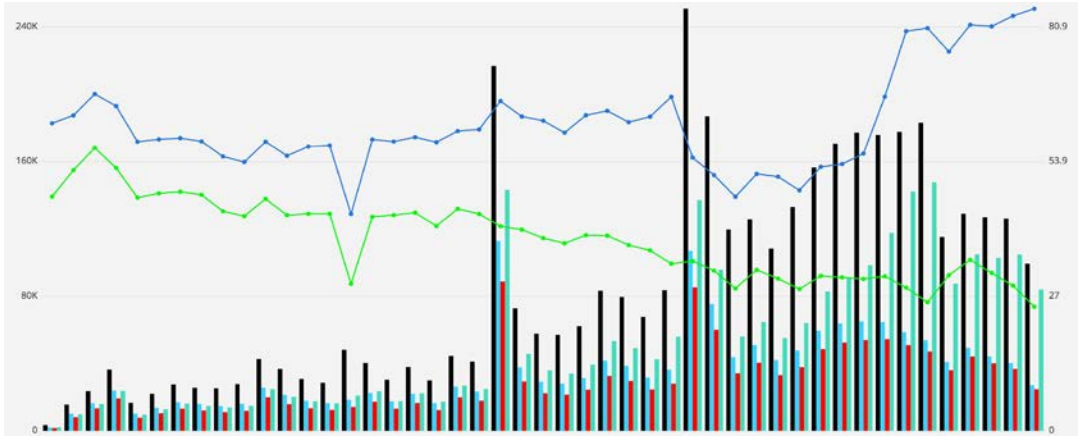


图 2

自己位置的轨迹。这使得消费者可以以实时和批处理的方式来操作数据。例如，批处理的消费者可以利用 Secor 将每个小时的数据拷贝发送到 S3 中；而实时消费者则利用即将开源的库将数据发送到 Elasticsearch 集群中。而且，在出现错误时，消费者还可以对数据进行重新处理。

商务智能

S3 中的数据经过 ETL 平台 Cranium 的转换和归一化后，输出到 AWS Redshift 中。Cranium 允许利用 SQL 和 Ruby 编写 ETL 任务、定义这些任务之间的依赖性以及调度这些任务的执行。Cranium 支持利用 Ruby 和 D3 进行的即席报告。但是，绝大部分的可视化工作还是发生在 Chartio 中（见图 2）。

而且，Chartio 对于只了解很少 SQL 的用户也非常友好。在这些工具的帮助下，从工程人员到业务研发人员和社区人员都可以对数据进行挖掘。

机器学习

IFTTT 的研发团队利用了很多机器学习技术来保证用户体验。对于 Recipe 推荐和问题探测，IFTTT 使用了运行在 EC2 上的 Apache Spark，并将 S3 当作其数据存储。

实时监控和提醒

API 事件存储在 Elasticsearch 中，用于监控和提醒。IFTTT 使用 Kibana 来

实时显示工作进程和合作 API 的性能。在 API 出现问题时，IFTTT 的合作者可以访问专门的 Developer Channel，创建 Recipe，从而提醒实际行动（SMS、Email 和 Slack 等）的进行（见图 3）。

在开发者视图内，合作者可以在 Elasticsearch 的帮助的帮助下访问 Channel 健康相关的实时日志和可视化图表。开发者也可以通过这些有力的分析来了解 Channel 的使用情况（见图 4）。

经验与教训

最后，Anuj 表示，IFTTT 从数据架构中得到的教训主要包括以下几点。

- 通过Kafka这样的数据传输层实现的生产者和消费者的隔离非常有用，且使得Data Pipeline的适应性更强。例如，一些比较慢的消费者也不会影响其他消费者或者生产者的性能。

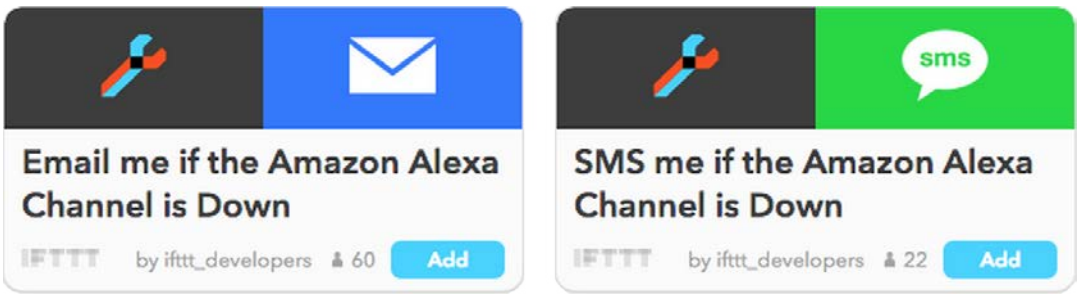


图 3

- 从一开始就要使用集群，方便以后的扩展！但是，在因为性能问题投入更多节点之前，一定要先认定系统的性能瓶颈。例如，在Elasticsearch中，如果碎片太多，添加更多的节点或许并不会加速查询。最好先减少碎片大小来观察性能是否改善。
- 在类似的复杂架构中，设置合适的警告来保证系统工作正常是非常关键的！IFTTT使用Sematext来监控Kafka集群和消费者，并分别使用Pingdom和Pagerduty进行监控和提醒。
- 为了完全信任数据，在处理流中加入若干自动化的数据验证步骤非常重要！例如，IFTTT开发了一个服务来比较产品表和Redshift表中的行

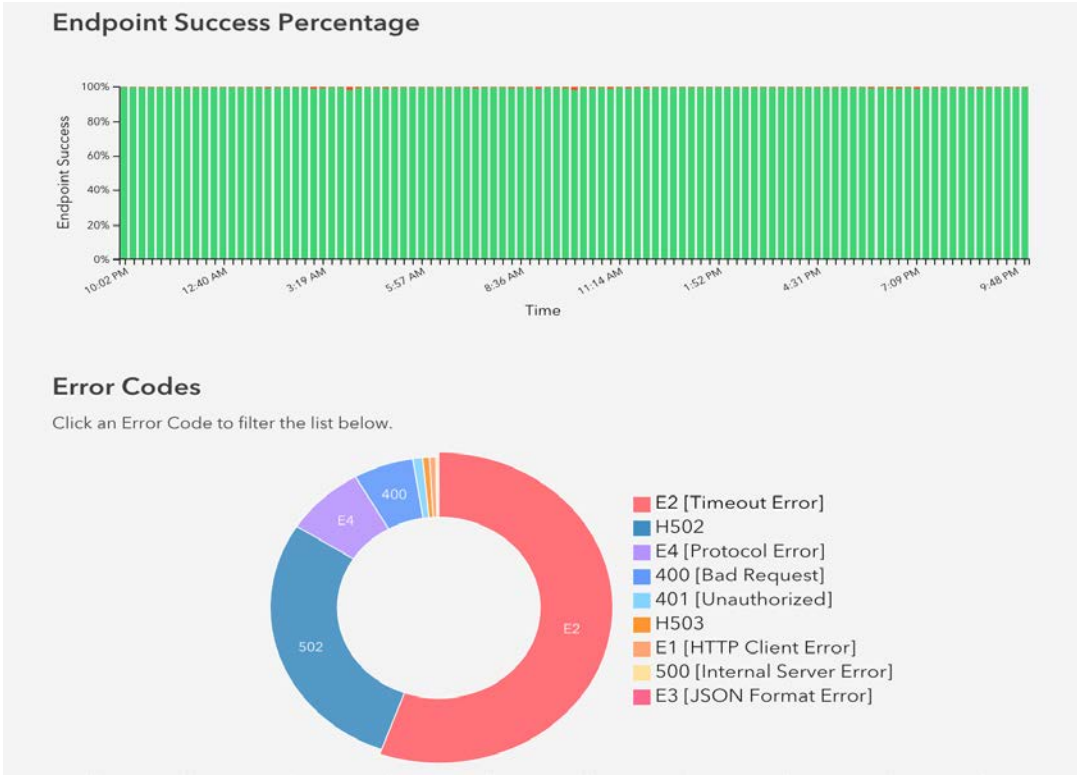


图 4

数，并在出现异常情况时发出提醒。

- 在长期存储中使用基于日期的文件夹结构（YYYY/MM/DD）来存储事件数据。这样存储的事件数据可以很方便的进行处理。例如，如果想读取某一天的数据，只需要从一个文件夹中获取数据即可。
- 在Elasticsearch中创建基于时间（例如，以小时为单位）的索引。这样，如果试图在Elasticsearch中寻找过去一小时中的所有API错误，只需要根据单个索引进行查询即可。
- 不要把单个数据马上发送到Elasticsearch中，最好成批进行处理。这样可以提高IO的效率。
- 根据数据和查询的类型，优化节点数、碎片数以及每个碎片和重复因子的最大尺寸都非常重要。

经典大数据架构案例 6：
面对百亿用户数据，日均亿次请求，携程应用架构如何涅槃

作者 **董锐**

互联网二次革命的移动互联网时代，如何吸引用户、留住用户并深入挖掘用户价值，在激烈的竞争中脱颖而出，是各大电商的重要课题。通过各类大数据对用户进行研究，以数据驱动产品是解决这个课题的主要手段，携程的大数据团队也由此应运而生；经过几年的努力，大数据的相关技术为业务带来了惊人的提升与帮助。

以基础大数据的用户意图服务为例，通过将广告和栏位的“千人一面”变为“千人千面”，在提升用户便捷性，可用性，降低费力度的同时，其转化率也得到了数倍的提升，体现了大数据服务的真正价值。

在新形势下，传统应用架构不得不变为大数据及新的高并发架构，来应对业务需求激增及高速迭代的需要。

一、业务高速发展带来的应用架构挑战

公司业务高速发展带来哪些主要的变化，以及给我们的系统带来了哪些挑战？

1）业务需求的急速增长，访问请求的并发量激增，2016 年 1 月份以来，业

务部门的服务日均请求量激增了 5.5 倍。

2）业务逻辑日益复杂化，基础业务研发部需要支撑起 OTA 数十个业务线，业务逻辑日趋复杂和繁多。

3）业务数据源多样化，异构化，接入的业务线、合作公司的数据源越来越多；接入的数据结构由以前的数据库结构化数据整合转为 Hive 表、评论文本数据、日志数据、天气数据、网页数据等多元化异构数据整合。

4）业务的高速发展和迭代，部门一直以追求以最少的开发人力，以架构和系统的技术优化，支撑起携程各业务线高速发展和迭代的需要。

在这种新形势下，传统应用架构不得不变，做为工程师也必然要自我涅槃，改为大数据及新的高并发架构，来应对业务需求激增及高速迭代的需要。计算分层分解、去 SQL、去数据库化、模块化拆解的相关技改工作已经刻不容缓。

以用户意图（AI 点金杖）的个性化服务为例， 面对 BU 业务线的全面支持的迫切需要，其应用架构必须解决如下技术难点：

1）高访问并发：每天近亿次的访问请求；

2）数据量大：每天 TB 级的增量数据，近百亿条的用户数据，上百万的产品数据；

3）业务逻辑复杂：复杂个性化算法和 LBS 算法；例如：满足一个复杂用户请求需要大量计算和 30 次左右的 SQL 数据查询，服务延时越来越长；

4）高速迭代上线：面对 OTA 多业务线的个性化、Cross-saling、Up-saling、需满足提升转化率的迫切需求，迭代栏位或场景要快速，同时减少研发成本。

二、应对挑战的架构涅磐

面对这些挑战，我们的应用系统架构应该如何涅磐？主要分如下三大方面系统详解：

存储的涅磐，这一点对于整个系统的吞吐量和并发量的提升起到最关键的作

用，需要结合数据存储模型和具体应用的场景。

计算的涅磐，可以从横向和纵向考虑：横向主要是增加并发度，首先想到的是分布式。纵向拆分就是要求我们找到计算的结合点从而进行分层，针对不同的层次选择不同的计算地点。然后再将各层次计算完后的结果相结合，尽可能最大化系统整体的处理能力。

业务层架构的涅磐，要求系统的良好的模块化设计，清楚的定义模块的边界，模块自升级和可配置化。

三、应用系统的整体架构

认识到需要应对的挑战，我们应该如何设计我们的系统呢，下面将全面的介绍下我们的应用系统整体架构。

图 1 就是我们应用系统整体架构以及系统层次的模块构成。

数据源部分， Hermes 是携程框架部门提供的消息队列，基于 Kafka 和 Mysql 做为底层实现的封装，应用于系统间实时数据传输交互通道。Hive 和 HDFS 是携程海量数据的主要存储，两者来自 Hadoop 生态体系。Hadoop 这块大

图 1

家已经很熟悉， 如果不熟悉的同学只要知道 Hadoop 主要用于大数据量存储和并行计算批处理工作。

Hive 是基于 Hadoop 平台的数据仓库，沿用了关系型数据库的很多概念。比如说数据库和表，还有一套近似于 SQL 的查询接口的支持，在 Hive 里 叫做 HQL，但是其底层的实现细节和关系型数据库完全不一样，Hive 底层所有的计算都是基于 MR 来完成，我们的数据工程师 90% 都数据处理工作都基于它来完成。

离线部分，包含的模块有 MR, Hive , Mahout, SparkQL/MLLib。Hive 上面已经介绍过，Mahout 简单理解提供基于 Hadoop 平台进行数据挖掘的一些机器学习的算法包。Spark 类似 hadoop 也是提供大数据并行批量处理平台，但是它是基于内存的。SparkQL 和 Spark MLLib 是基于 Spark 平台的 SQL 查询引擎和数据挖掘相关算法框架。我们主要用 Mahout 和 Spark MLLib 进行数据挖掘工作。

调度系统 zeus，是淘宝开源大数据平台调度系统，于 2015 年引进到携程，之后我们进行了重构和功能升级，做为携程大数据平台的作业调度平台。

近线部分，是基于 Muise 来实现我们的近实时的计算场景，Muise 是也是携程 OPS 提供的实时计算流处理平台，内部是基于 Storm 实现与 HERMES 消息队列搭配起来使用。例如，我们使用 MUSIE 通过消费来自消息队列里的用户实时行为，订单记录，结合画像等一起基础数据，经一系列复杂的规则和算法，实时的识别出用户的行程意图。

后台 / 线上应用部分，Mysql 用于支撑后台系统的数据库。ElasticSearch 是基于 Lucene 实现的分布式搜索引擎，用于索引用户画像的数据，支持离线精准营销的用户筛选，同时支持线上应用推荐系统的选品功能 。Hbase 基于 Hadoop 的 Hdfs 上的列存储 Nosql 数据库，用于后台报表可视化系统和线上服务的数据存储。

这里说明一下， 在线和后台应用使用的 ElasticSearch 和 Hbase 集群是分开的，互不影响。 Redis 支持在线服务的高速缓存，用于缓存统计分析出来的热点数据。

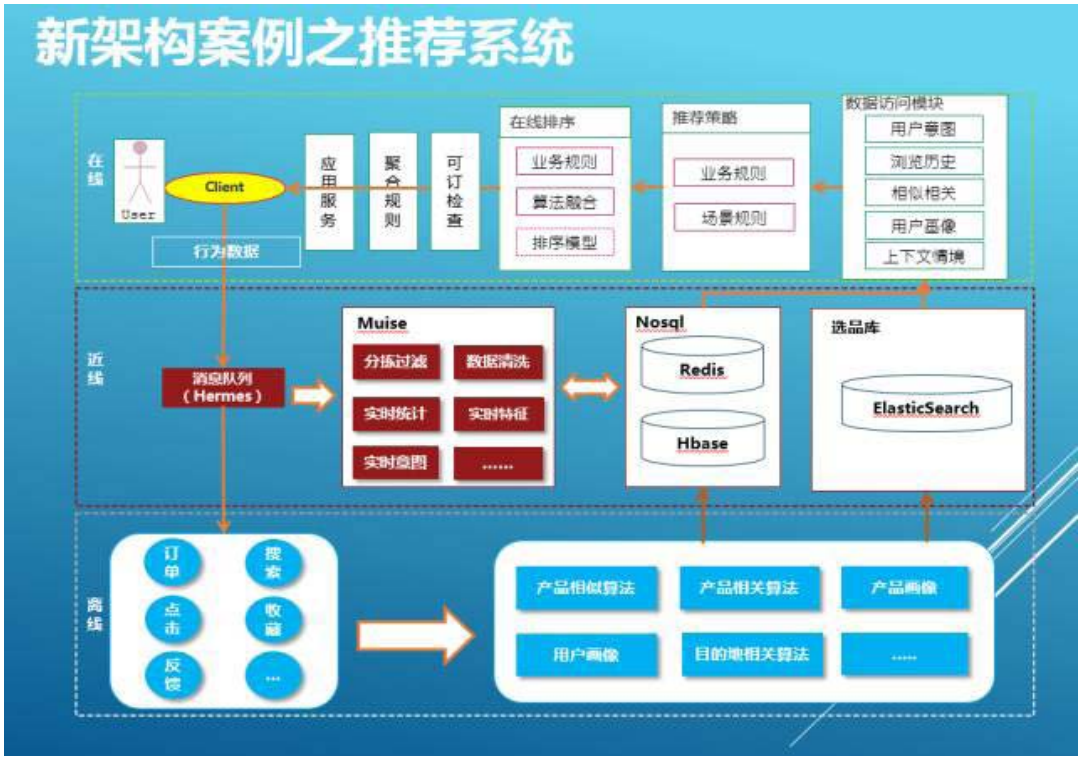


图 2

四、推荐系统案例

介绍完我们应用系统的整体构成， 接下来分享基于这套系统架构实现的一个实例——携程个性化推荐系统（见图 2）。

1 存储的涅磐

1）NoSQL (Hbase+Redis)

我们之前存储使用的是 MySQL， 一般关系型数据库会做为应用系统存储的首选。大家知道 MySQL 非商业版对分布式支持不够，在存储数据量不高，查询量和计算复杂度不是很大的情况下，可以满足应用系统绝大部分的功能需求（见图 3）。

我们现状是需要安全存储海量的数据，高吞吐，并发能力强，同时随着数据量和请求量的快速增加，能够通过加节点来扩容。另外还需要支持故障转移，自动恢复，无需额外的运维成本。综上几个主要因素，我们进行了大量的调研和测试，最终我们选用 Hbase 和 Redis 两个 NoSQL 数据库来取代以往使用的 MySQL，。我

图 3

们把用户意图以及推荐产品数据以 KV 的形式存储在 Hbase 中，我对操作 Hbase 进行一些优化，其中包括 rowkey 的设计，预分配，数据压缩等， 同时针对我们的使用场景对 Hbase 本身配置方面的也进行了调优。目前存储的数据量已经达到 TB 级别，支持每天千万次请求，同时保证 99% 在 50 毫秒内返回 。

Redis 这块和多数应用系统使用方式一样，主要用于缓存热点数据，这里就不多说了。

2）搜索引擎 (ElasticSearch)

ES 索引各业务线产品特征数据，提供基于用户的意图特征和产品特征复杂的多维检索和排序功能，当前集群由 4 台大内存物理机器构成，采用全内存索引。对比某一个复杂的查询场景， 之前用 Mysql 将近需要 30 次查询，使用 ES 只需要一次组合查询且在 100 毫秒内返回 。目前每天千万次搜索，99% 以上在 300 毫秒以内返回（见图 4、图 5）。


存储—ElasticSearch
根据用户的feature和产品的feature提供统一的选品查询接口。
提供复杂排序功能，如按照地理信息排序，按照命中率排序。
索引实例
索引管理
dim_act
dim_cruise
dim_diy
dim_pkg
dim_test
dim_ttd
产品信息
用户产品rating
全量/增量
格式化
格式化
统计评分
数据管理工具
ElasticSearch

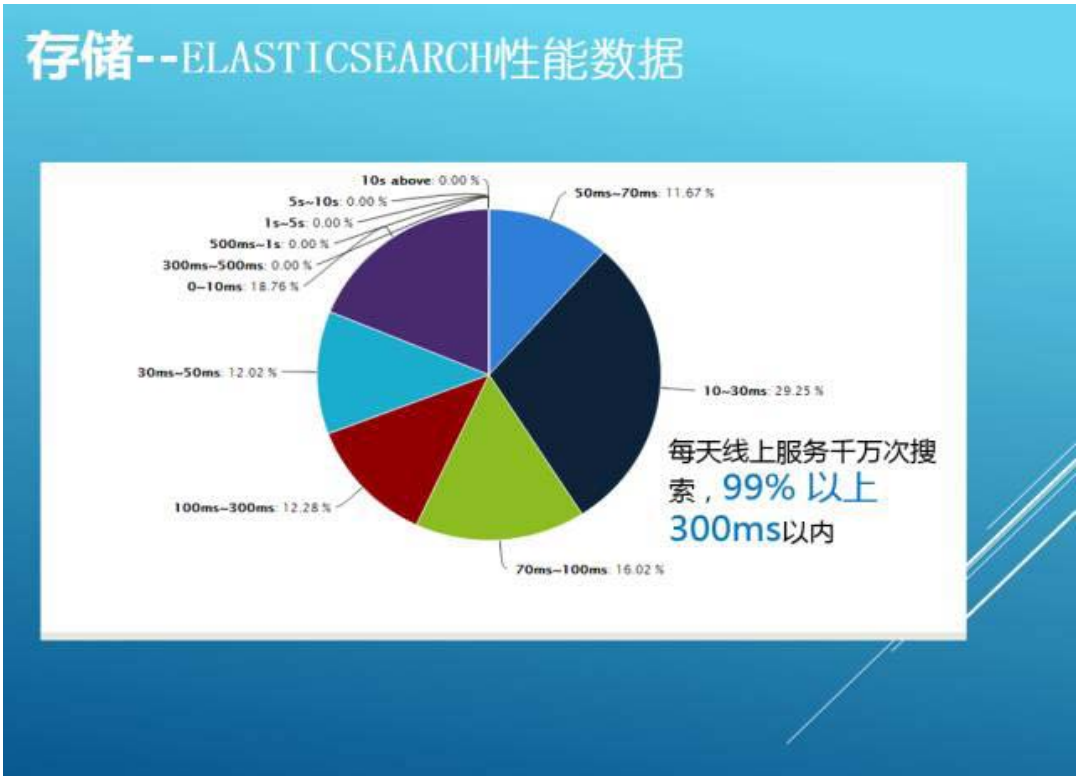

存储--ELASTICSEARCH性能数据
10s above: 0.00 %
5s~10s: 0.00 %
1s~5s: 0.00 %
500ms~1s: 0.00 %
300ms~500ms: 0.00 %
0~10ms: 18.76 %
50ms~70ms: 11.67 %
10~30ms: 29.25 %
30ms~50ms: 12.02 %
100ms~300ms: 12.28 %
70ms~100ms: 16.02 %
每天线上服务千万次搜索，99% 以上300ms以内

图4、图5

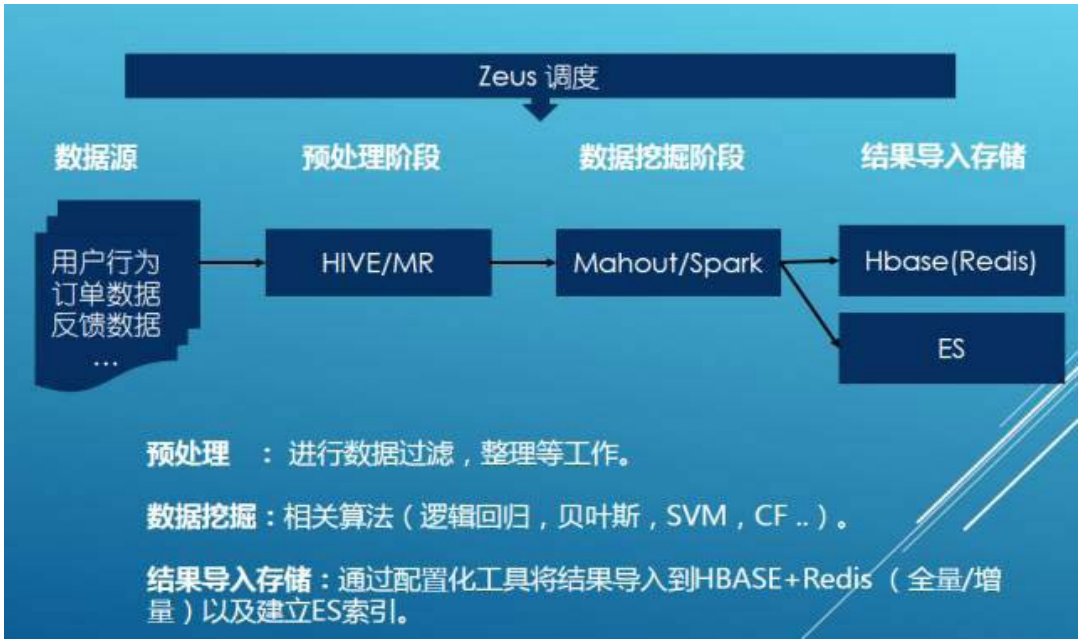


图 6

2 计算的涅磐

1）数据源，我们的数据源分结构化和半结构化数据以及非结构化数据

结构化数据主要是指携程各产线的产品维表和订单数据，有酒店，景酒，团队游，门票，景点等；还有一些基础数据，比如城市表，车站等，这类数据基本上都是 T+1。每天会有流程去各 BU 的生产表拉取数据。

半结构化数据是指，携程用户的访问行为数据，例如浏览，搜索，预订，反馈等，这边顺便提一下，这些数据这些是由前端采集框架实时采集，然后下发到后端的收集服务，由收集服务在写入到 Hermes 消息队列，一路会落地到 Hadoop 上面做长期存储，另一路近线层可以通过订阅 Hermes 此类数据 Topic 进行近实时的计算工作。

我们还用到外部合作渠道的数据，还有一些评论数据，评论属于非结构化的，也是 T+1 更新。

2）离线计算，主要分三个处理阶段

预处理阶段，这块主要为后续数据挖掘做一些数据的准备工作，数据去重，过滤，对缺失信息的补足（见图 6）。举例来说采集下来的用户行为数据，所含

有的产品信息很少，我们会使用产品表的数据进行一些补足，确保给后续的数据挖掘使用时候尽量完整的。

数据挖掘阶段，主要运用一些常用的数据挖掘算法进行模型训练和·推荐数据的输出（分类，聚类，回归 ，CF 等）。

结果导入阶段，我们通过可配置的数据导入工具将推荐数据，进行一系列转换后，导入到 HBASE，Redis 以及建立 ES 索引， Redis 存储的是经统计计算出的热点数据 。

3）近线计算（用户意图，产品缓存）

当用户没有明确的目的性情况下，很难找到满足兴趣的产品，我们不仅需要了解用户的历史兴趣，用户实时行为特征的抽取和理解更加重要，以便快速的推荐出符合用户当前兴趣的产品，这就是用户意图服务需要实现的功能（见图 7）。

一般来说用户特征分成两大类：一种是稳定的特征（用户画像），如用户性别，常住地，主题偏好等特征；另一类是根据用户行为计算获取的特征，如用户对酒店星级的偏好，目的地偏好，跟团游 / 自由行偏好等。基于前面所述的计算的特点，

图 7

我们使用近在线计算来获取第二类用户特征，整体框图如下。从图中可以看出它的输入数据源包括两大类：第一类是实时的用户行为，第二类是用户画像，历史交易以及情景等离线模块提供的数据。结合这两类数据，经一些列复杂的近线学习算法和规则引擎，计算得出用户当前实时意图列表存储到 Hbase 和 Redis 中。

近线另一个工作是产品数据缓存，携程的业务线很多，而我们的推荐系统会推各个业务线的产品，因此我们需要调用所有业务线的产品服务接口，但随着我们上线的场景的增加，这样无形的增加了对业务方接口的调用压力。而且业务线产品接口服务主要应用于业务的主流程或关键型应用，比较重，且 SLA 服务等级层次不齐，可能会影响到整个推荐系统的响应时间。

为了解决这两个问题，我们设计了近在线计算来进行业务的产品信息异步缓存策略，具体的流程如下。

我们会将待推荐的产品 Id 全部通过 Kafka 异步下发，在 Storm 中我们会对各业务方的产品首先进行聚合，达到批处理个数或者时间 gap 时，再调用各业务方的接口，这样减少对业务方接口的压力（见图 8）。通过调用业务方接口更新的产品状态临时缓存起来（根据各业务产品信息更新周期分别设置缓存失效时间），在线计算的时候直接先读取临时缓存数据，缓存不存在的情况下，再击穿

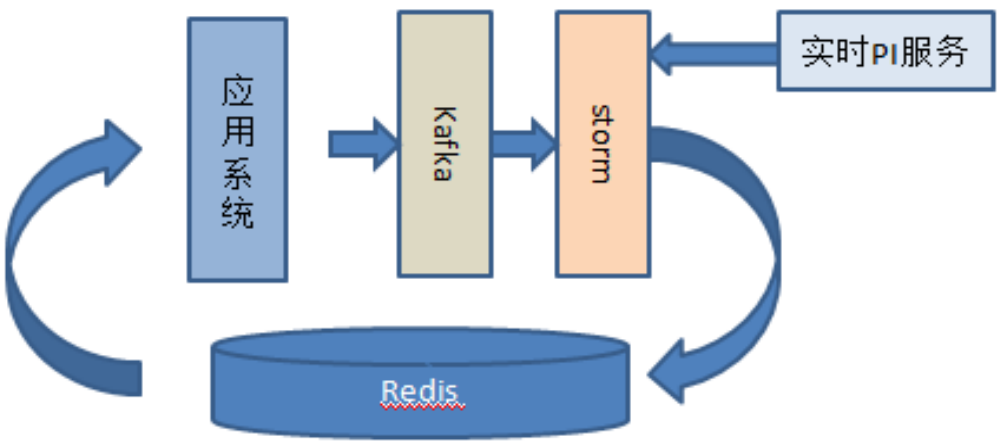


图 8

图 9

到业务的接口服务。

4）在线计算（2 个关键业务层架构模块介绍）

a）业务层架构 - 数据治理和访问模块，支持的存储介质，目前支持的存储介质有 Localcache，Redis，Hbase，Mysql 可以支持横向扩展（见图 9） 。统一配置，对同一份数据，采用统一配置，可以随意存储在任意介质，根据 id 查询返回统一格式的数据，对查询接口完全透明。

穿透策略和容灾策略，Redis 只存储了热数据，当需要查询冷数据则可以自动到下一级存储如 Hbase 查询，避免缓存资源浪费。当 Redis 出现故障时或请求数异常上涨，超过整体承受能力，此时服务降级自动生效，并可配置化。

b）业务层架构 - 推荐策略模块，整个流程是先将用户意图、用户浏览，相关推荐策略生成的产品集合等做为数据输入，接着按照场景规则，业务逻辑重新过滤，聚合、排序。最后验证和拼装业务线产品信息后输出推荐结果；

我们对此流程每一步进行了一些模块化的抽象，将重排序逻辑按步骤抽象解

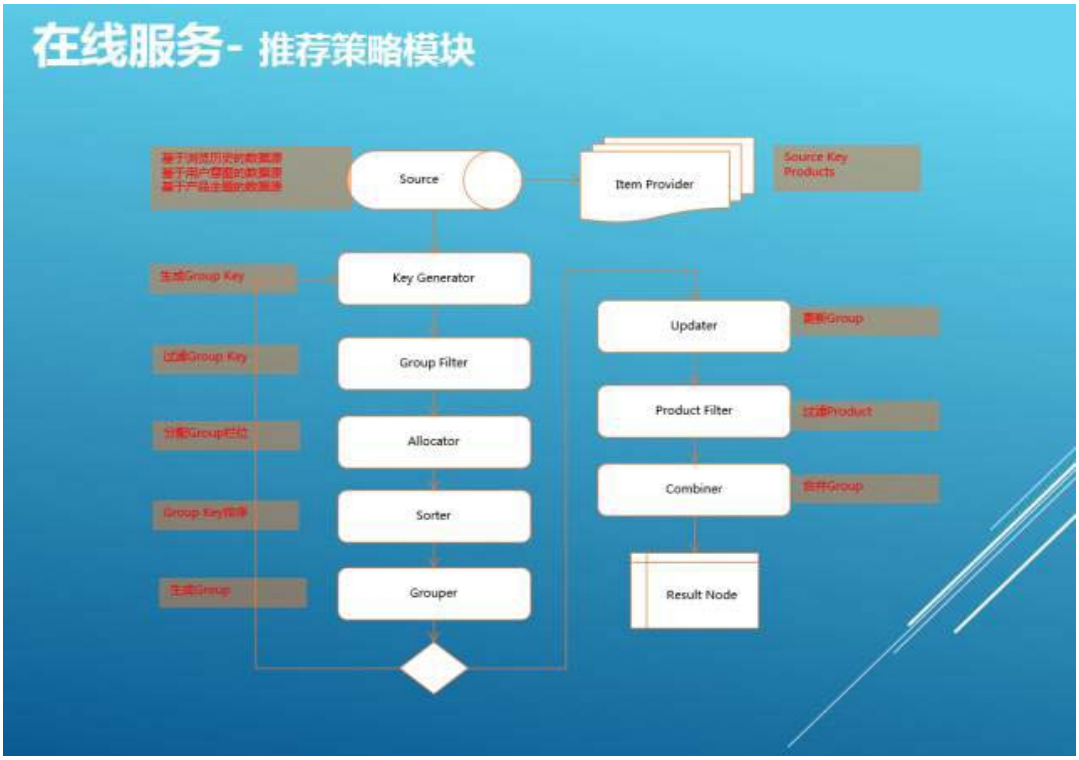


图 10

耦，抽象如图 10 所示的多个组件，开发新接口时仅需要将内部 DSL 拼装便可以得到满足业务需求的推荐服务；提高了代码的复用率和可读性，减少了超过 50% 的开发时间；对于充分验证的模块的复用，有效保证了服务的质量。

董锐，近 9 年的互联网从业经验。2013 年 1 月加入携程，曾在商业智能部设计和开发基于 HADOOP 生态体系的大数据数据仓库。现任基础业务研发部 - 数据智能应用组研发 Leader，专注于携程个性化推荐系统，ABTEST 等系统研发工作。

版权声明

InfoQ 中文站出品

架构师特刊：大数据平台架构

出版：极客邦控股（北京）有限公司

北京市朝阳区洛娃大厦 C 座 1607

欢迎共同参与 InfoQ 中文站的内容建设工作，包括原创投稿和翻译，请联系 editors@cn.infoq.com。

网 址：www.infoq.com.cn