架构师

ARCHITECT

特刊

Apache Kylin实践

SPECIAL ISSUE

October,2016

架构师特刊

Geekbang 极客邦科技 | InfoQ

Geekbang.
极客邦科技

极客邦科技官网

Mission 使命

整合全球优质学习资源
帮助技术人和企业成长

Vision 愿景

全球领先的技术人
学习和交流平台

极客邦科技是一家 IT 技术学习服务综合提供商，旗下运营 InfoQ 技术媒体、EGO 社交网络、StuQ 职业教育三大业务品牌，致力于通过整合全球优质学习资源，帮助技术人和企业成长。

了解更多，请访问官网：http://www.geekbang.org

序言

麒麟者，神兽也，古人以为，其为四灵之一，仁兽，凡其出没，必有祥瑞。

两年前，我们在 eBay 的时候开源了一个大数据 OLAP 框架，在基本都是使用动物来命名项目名字的大数据生态中，我们选取了麒麟，Kylin，作为项目的名字，最终成为了今天的 Apache Kylin 项目，这在世界众多大数据及开源项目中一眼即知这是来自中国的开源项目。

在 Hadoop 被证明是一个优秀的大数据存储及批处理的平台之后，自然而然用户的需求是如何在 Hadoop 平台之上实现传统数据仓库，商业智能的能力，提供交互式的，多维分析能力，或者，换句话说，如何在 Hadoop 之上，构建数据集市及数据仓库，并提供在传统数据仓库技术所不能做到的超大规模数据集的快速查询，并使用普通的 PC 硬件，而无需采购专用的，私有的一体机或者高端存储等。

因此，Kylin 项目一经开源，即获得了业界众多的称赞，并被邀请加入

Apache 软件基金会的孵化项目，在 2014 年 11 月，正式经投票加入了 Apache 大家庭，项目名字也改成了“Apache Kylin”，在项目开源伊始即幸运的加入全球最大的开源软件基金会，从而开启了麒麟大数据之旅。

在 2015 年 11 月，Apache Kylin 经项目管理委员会及孵化器管理委员会共同投票，正式毕业成为 Apache 顶级项目，和 Apache Hadoop, Apache Spark, Apache Kafka 等众多软件一起成为顶级项目，这是中国工程师的骄傲，目前为止，Apache Kylin 是唯一来自中国的 Apache 顶级开源项目。我本人也成为 Apache 软件基金会唯一一位来自中国的项目管理委员会主席，副总裁。并被提名成为 Apache Member (ASF Member)。目前，项目发展了众多的 PMC member 及 committer, 包括来自 Kyligence、美团、百度、京东、eBay 等众多的贡献者及用户活跃在 Kylin 的社区。

开源项目，开源容易使用难，被广泛使用则更难。而今天，经过 Kylin 社区两年不断的努力和发展，已 经有超过 100 多家国内国外的公司正式使用 Kylin 作为其大数据分析平台的核心。包括 eBay、Glispa、微软、Expedia、百度、美团、网易、京东、唯品会、中国移动、中国电信、国泰君安、华泰证券、联想、OPPO、魅族、去哪儿，等等。Apache Kylin 被用到了诸多如数据仓库，用户行为分析，流量（日志）分析，自助分析平台，电商分析，广告效果分析，实时分析，数据服务平台等各种场景。并且众 多使用者在贡献了非常多的代码，特性等同时，也分享了很多的案例参考，促进了 Kylin 社区的进一步发展。

开源，也促进的商业模式的变革，从最早的 Linux 商业化公司红帽，到今天，Hadoop 诞生了 Cloudera, Hortonworks, MaR, Kafka 诞生了 Confluent, Spark 核心开发者则组建了 Databricks。同样，Kylin 的核心开发者，也在中国组建了 Kyligence, 作为中国唯一一家完全由 Apache 顶级项目核心开发者组建的公司，在中国及国际上进行开源 - 商业模式的探索和尝试，从提供企业级的 Kyligence Analytics Platform,, 到在线的诊断与支持服务 KyBot 等，正在一步步实践开源 - 创业 - 商业的转换路径。

在过去的两年，在 Kylin 的开源道路上，InfoQ一直是我们的好朋友，作为国内最专业的技术媒体，一直关注、帮助并支持 Kylin 的开源和发展，发表了众多的技术文章，为广大开发者，使用者等各位朋友带去了最新的 Kylin 技术架构、算法及众多的实际用户案例。值此开源两年之际，InfoQ 将之前发表的 Kylin 相关的文章集结成册，向社会发行电子书，在此我谨代表 Kylin 社区对此表示由衷的感谢。并且也希望本电子书能为各位读者带来更多更有价值的 Kylin，大数据分析技术及相关的案例，同时，也期待更多的朋友不仅仅使用 Kylin, 而更多的贡献到 Kylin 社区，推动 Kylin 技术的演进，并一起来发展和壮大 Kylin 社区在国际上的影响力。也期待更多的朋友能够撰写相关的技术文章，案例分享并推荐给 InfoQ。

最后，感谢 eBay 最初贡献并开源了 Apache Kylin 项目，感谢 Kylin 的导师、核心开发者、贡献者、用户及社区的每一位朋友，欢迎大家使用并参与 Kylin 社区，如果有任何问题或者建议，欢迎将相关内容发 送给我们邮件列表。

——韩卿 | Luke Han

Kyligence 联合创始人 &CEO

Apache Kylin 联合创始人 &PMC Chair

ASF Member, 微软 MVP

EGO 会员，QCon 明星讲师，QCon 明星出品人

目录

Apache Kylin
在电信运营商的实践和案例分享

作者 **赵磊**

我最近看了一篇文章，名为《开源项目的正确打开方式》，文章中把开源项目的研究分成了三个阶段：选、用、修改。

一是怎么选开源项目，包括满足业务需求，具备运维能力，项目基本成熟，团队靠谱，社区活跃等；

二是怎么用开源项目，包括深入研究仔细测试，做好应急以防万一，小心应用灰度发布，结合业务场景做好参数调整等；

三是怎么修改开源项目，就是保持纯洁加以包装，发明适合自己的轮子。

目前我们团队处于第二阶段，我希望未来我们能够有更多的成熟经验，也能够为 Kylin 社区做一些贡献。

我们为什么选择Kylin

首先，我们的数据规模决定要选择高效的处理技术。（见图 1）

北京移动的用户规模超过两千万，每天入库的原始数据超过三百亿条。经过处理后入库的数据是 3TB，而集群规模是 400TB 存储；每天执行的任务超过 800 个，其中大概有 600-700 个是属于临时产生的任务，所以我们的集群很繁忙。如果不选择高效的数据处理技术，将无法满足分析需求。Kylin 可以在夜间非忙时进行

图 1

一些预计算，这样可以满足一些临时任务的数据需求，从而提升集群的工作效率。

我们选择 Kylin 的第二点原因是数据需求的困境。

这两年，分析人员、优化人员对数据的临时性查询越来越多，探索性数据需求越来越旺盛，我们需要找到一个方法来满足这类需求。首先，我们寻求固定化报表的解决，相信大家都会去做。我们也是原来做了很多报表放在 MySQL 里供查询。但这样做非常不灵活，开发周期缓慢，而且经常出现需求变更和需求不明确的情况，所以报表只适用于固定化场景的情况。

使用 Hive 和 Spark Sql 可以满足探索性数据分析的需求，但 Hive 速度较慢，Spark Sql 对内存资源要求很高，不适合多并发。如果应用的场景是数据来源固定，但是查询不固定且要求速度时，Kylin 就是一个选择了。

我们选择 Kylin 的第三点原因是它部署速度快，查询速度快。

如果了解 Kylin 的话就知道，Kylin 建立一个项目只需要简单几步：选定事实表，选定维度，选定测量值，选定好过滤条件，再把一些优化条件设定好之后，这个项目基本就建立完成了，学习成本相对比较低。

从查询速度上而言，我做过一个测试，原始数据大小 103GB，条目数 11 亿，任务是统计用户数。我用 Hive 测试，任务花费 1522 秒，Spark Sql 测试是 125 秒，用 Kylin 测试用时 3.43 秒。

诚然，Kylin 预计算也要花费时间和资源，但它是在晚上闲时进行的，所以当应用这个预计算结果足够多时，之前预计算的花费也是值得的。

	执行资源	执行时长	备注
Hive	86vcores+380GBMEM	1522秒	orc+zlib
Spark Sql	131vcores+912GBMEM	125秒	orc+zlib
Kylin		3.43秒	

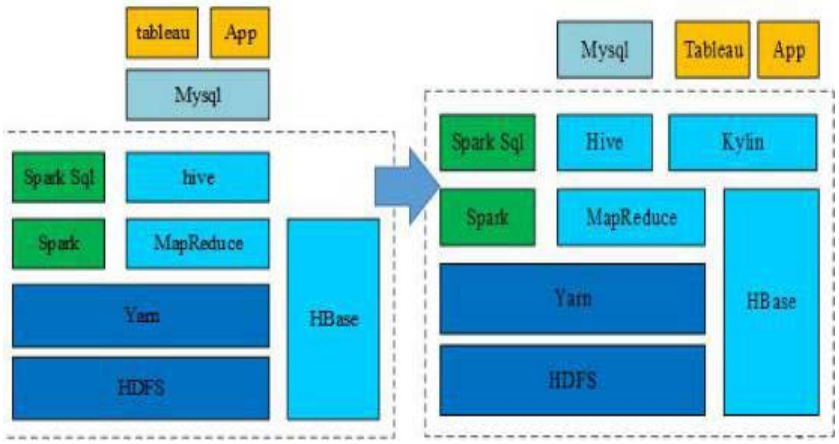


图 2

使用Kylin的应用场景

首先介绍下我们的离线平台的**架构变化**。

图 2 左半部分是应用 Kylin 之前的架构，前台查询都基于 MySQL，底层数据是抽取后放入到 MySQL。个人认为这是一个割裂的架构，即大数据平台和前台查询模块不是一体的。我们也曾尝试把前台查询对接到 Hive 上，但效率比较低。下图右半部分是我们现在的架构，Kylin 可以有效的衔接前后台。

接下来介绍一下我们的第一个应用场景。

图 3 是用户上网统计表的字段说明，蓝色字段是这个表的维度，绿色字段是测量值。首先是统计报表，即基于日期、时间等八个维度统计用户上网的流量和、次数和，等等。

在实践中，我们发现用户 ID 这个维度最好不要放到项目中，原因是 Kylin 不推荐基数高的字段作为维度，而我们 ID 的基数是 2000 万以上。那如果应用场景就是基于用户 ID 的该如何处理呢？我们的做法就是把用户上网的统计表全部

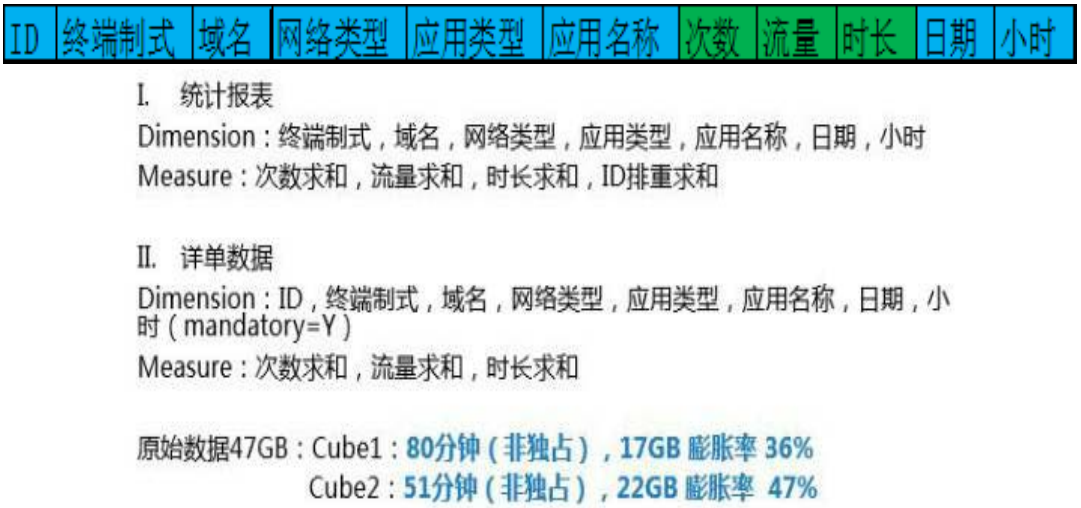


图 3、图 4

八个维度一起作为一个维度来看，从而避免了单一维度基数过高的问题。

当我把 ID 作为输入条件时，查询结果就是原始表中符合条件的记录，这样基于这个表的各种灵活查询场景都可以满足了。

图 4 是两个子场景的一些统计数据，原始的数据是 47GB。统计报表任务执行时间是 80 分钟，详单任务执行时间是 51 分钟，都是在白天忙时执行的。第一个任务膨胀率是 36%，第二个任务膨胀率是 47%，即两个任务相加产生的预计算结果数据小于原始数据。

第二个场景比较有意思，业务人员的诉求是查询到不同方向的流量信息，就像一个交通路口，我们想分别统计到东、南、西、北各个方向上的流量是多少。

下面这个表的主要字段说明，它是一张宽表，有 40 多列。这个表需要特别说明的是 Hostname 的基数超过五百万，Rate 取值范围是 0.00-100%，而各个方向上流量的数值就更加离散了，从 0 到几亿。这么多维度，数值离散，再加上某些字段的基数很高，为我们设计查询造成了困难。

通过和业务人员了解需求，其核心诉求是明确在各个方向上是否产生过流量，所以我们对原始数据进行了重新设计，为各个方向设计了 Flag 字段：就是这个方向只要有流量，我就把 Flag 变成 1，如果没有流量，Flag 就变成零。

通过这么设计后，我们大部分查询可以将时间控制在 20 秒以内，基本满足

APPTYPE	APPNAME	HOSTNAME	方向1	方向2	...省略一些方向字段	成功率	失败率	延时标志	日期

APPTYPE	APPNAME	HOSTNAME	方向1	方向1流量标志	...省略一些方向和标志字段	成功率	失败率	延时标志	日期

需求；但范围查询，如查询成功率大于 90.00% 的情况，执行时长超过 200 秒，无法满足业务查询需求，这种类型的查询目前用 Spark Sql 满足。可以说 Kylin 可以支持我们 70%-80% 的 OLAP 分析。

使用Kylin时应注意哪些问题

接下来我想分享一下使用 Kylin 时的注意事项，实际上就是我们曾经踩过的“坑”。

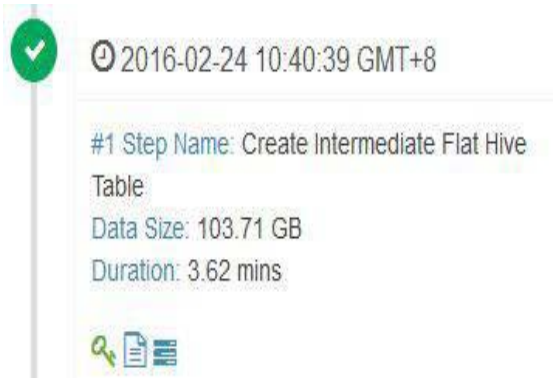
第一、设计好你的原始数据。首先要处理好脏数据，不要把过多的数据处理工作让 Kylin 来完成；再者是原始数据表字段类型的选择，测量值要选择 Double 而不是 Float 型；还有就是要控制测量值长度的控制范围。

第二、设计好你的 Cube。在设计查询任务时，要首先思考一下——真的所有维度都需要吗？这个问题不仅是问你的，也是问你能接触到的业务人员的。

这里分享一个我们的惨痛经历：我们第一次尝试建立 cube 时，有两个非常高基数的维度，分别是两千万以上和三百万以上，建立 cube 的任务执行了 8 个小时，数据膨胀了 28 倍不止（见图 5）。

从我了解到的情况，Kylin 的 cube 膨胀率一般不会超过 50%，可见这是设计 cube 的问题。最终我们这个需求分拆为两个 cube，膨胀率和不到 100%，查询速度也满足需求（见图 6）。

第三，选择合适的维度、测量值类型。举个例子，我们的一个应用场景里有



Name	Status	Cube Size	Source Records
phone_kylin1_clone	READY	2.84 TB	2,984,616,168

图5、图6

三个维度分别是应用大类（比如门户网站）、应用小类（比如新浪）和Host（www.sina.com），这三个维度是有层级关系的，所以在选择维度时就不要用normal而应该选择hierarchy。还有就是理解每个参数的信息，好好理解Cube建立和设计的思想。比如某个维度基数超过两千万时，它就不适合用字典，只需要规定下其长度就好。

基于Kylin的前景规划

最后讲一下我们的规划：首先，将Kylin升级到1.5版本，支持TopN功能。我们当前的版本是1.2，当基于某些基数较高的维度复杂查询时，就会出现图7的报错。其实，对于基数较高的查询场景，可以通过不同的TopN加排序来满足。

第二个规划是重新思考Cube引擎的选择。我们有一些应用场景是要快速回溯很近一段时间的数据，比如投诉和故障的信息。这种需求场景的查询是很不确定的。而我们每天数据量级是几百亿条，天粒度来执行Kylin是无法满足需求的。现在Kylin已经解耦MapReduce，我们正在考虑采用Spark Streaming作为运算引擎，采用micro cube的方式实现准实时的OLAP分析。

第三个规划是要设计符合需求的拖拽前台界面。原因很简单：一是支持探索性数据查询。作为一个开发人员，你一定希望自己设计的查询系统用户黏性大，

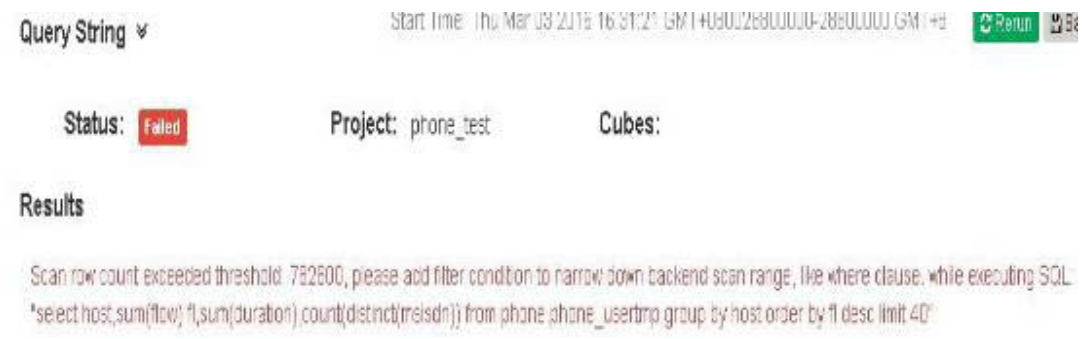


图 7

采用拖拽式方便用户使用；二是规定化前台界面，屏蔽后台技术细节，避免低效的查询出现。

最后，我们会持续关注 Kylin 的发展变化。目前，Kylin 一般只支持 15 个维度，而我们的一些应用场景是远远大于这个限制的，我们怎么基于宽表的 OLAP 查询，怎么更快更好的查询到数据？我们想把 Kylin 应用到我们的标签业务上，这就要求系统支持灵活多变且具有一定的时效性，我们该如何做到？这些问题都需要在进一步的学习和交流中找到答案。

提问环节

提问：您刚才提到的基数较高的维度复杂查询时报错的问题，能再介绍一下吗？

回答：我们的应用场景是基于域名的统计，想要把域名按照流量、次数、人数进行汇总排名，由于域名量超过 200 万，Kylin 在扫描全部数据时会导致内存不足的问题，所以引起了这个报错。

提问：在我们的使用场景中也遇到了类似的问题，所以一直没有办法应用 Kylin。

回答：这个问题其实可以利用 TopN 来基本满足需求，因为很多时候我们只关心 80% 的情况，或者通过逆序来查询剩余的数据，我们将尽快升级到 1.5 版来检验 TopN 功能。

Apache Kylin
在美团数十亿数据 OLAP 场景下的实践

作者 **孙业锐**

作为公司的平台部门，需要给各个业务线提供平台的服务，那么如何建设一个满足各种需求的公司平台级 OLAP 分析服务呢。首先，一个开源项目在公司真正落地会遇到很多障碍，这主要是由各个业务线不同的数据特点和业务特点决定的，所以本文会介绍一下美团的数据场景有什么特点；其次，针对这些数据特点，尤其是和 Kylin 设计初衷不太相符的部分，有什么样的解决方案；第三，目前 OLAP 领域还没有所谓事实上的标准，很多引擎都可以做类似事情，比如普通的 MPP，Kylin，或者 ES 等。这些系统之间的对比情况如何，应该如何选择，我们也有部分测试数据可以分享；最后，简单讨论一下未来准备在 Kylin 上做的工作。

一、美团的数据场景特点

第一个特点是数据规模和模型特点。一方面从数据规模上来讲，事实表一般在 1 亿到 10 亿量级，同时还有千万量级的维表，也就是超高基数的维表。另一方面，数据模型是一开始遇到的最大困难。因为 Kylin 最初的设计是基于一个星形模型的，但很不幸由于各种原因，很多数据都是雪花的模型，还有其它的模型，比如所谓“星座”模型，也就是中间是两张或者三张事实表，周围关联了其它很多维

表。业务逻辑决定了这些数据的关联方式非常复杂，根本无法用经典标准的理论来解释。

第二个是维度。维度最理想的情况是固定的，每天变化的只是事实表。但实际上维度经常会变，这可能和行业特点有关，比如组织架构，相关的维度数据可能每天都会变化。除此之外还可能要用今天的维度去关联所有的历史数据，因此要重刷历史数据，相应的开销也比较大。

第三个是数据回溯的问题。比如发现数据生成有问题，或者上游出错了，此时就需要重跑数据。这也是和经典理论模型有区别的。

从维度的角度来看，一般维度的个数在 5-20 个之间，相对来说还是比较适合用 Kylin 的。另一个特点是一般都会有一个日期维度，有可能是当天，也有可能是一个星期，一个月，或者任意一个时间段。另外也会有较多的层次维度，比如组织架构从最上面的大区一直到下面的蜂窝，就是一个典型的层次维度。

从指标的角度来讲，一般情况下指标个数在 50 个以内，相对来说 Kylin 在指标上的限制并没有那么严格，都能满足需求。其中有比较多的表达式指标，在 Kylin 里面聚合函数的参数只能是单独的一列，像 sum(if…) 这种就不能支持，因此需要一些特别的解决方法。另外一个非常重要的问题是数据的精确性，目前在 OLAP 领域，各个系统都是用 hyperloglog 等近似算法做去重计数，这主要是出于开销上的考虑，但我们的业务场景要求数据必须是精确的。因此这也是要重点解决的问题。

在查询上也有比较高的要求。因为平台的查询服务可能直接向城市 BD 开放，每次会有几十、上百万次的访问，所以稳定性是首先要保证的。第二要求有很高的性能。因为用 Kylin 主要是为了实现交互式的分析，让使用者能够很快拿到结果，所以需要秒级响应。

另外经常会有人问到，Kylin 有没有可视化的前端，在我们内部更多是由业务方来做，因为原来本身就有这样的系统，以前接的是 MySQL 等其它的数据源，现在可以直接使用 Kylin 的 JDBC driver 对接起来。

以上是美团在 OLAP 查询方面的一些特点。在用 Kylin 之前，实际上有一些方案，但效果并不理想。

比如用 Hive 直接去查，这种情况下，第一个是慢，第二会消耗计算集群的资源。尤其每个月第一天，大家都要出月报，跑的 SQL 非常多，全提到集群上去，并发度限制导致跑的比平时更慢。

我们原来也做过预聚合的尝试，这个思路跟 Kylin 很像，只不过是自己做这个事，用 Hive 先把所有的维度算出来，然后导入 MySQL 或者 HBase。但是这个方案并没有像 Kylin 这么好的模型定义抽象，也没有从配置到执行，预计算，查询这样整体的框架。现在通过使用 Kylin 实现了低成本的解决这些问题。

二、 接入Apache Kylin的解决方案

针对上述的问题，经过大量的尝试和验证，目前主要的解决方案有以下几点。最重要的第一点，就是采用宽表。所有非标准星型的数据模型，都可以通过预处理先拉平，做成一个宽表来解决。只要能根据业务逻辑把这些表关联起来，生成一张宽表，然后再基于这张表在 Kylin 里做数据的聚合就可以了。宽表不只能解决数据模型的问题，还能解决维度变化、或者超高基数的维度等问题。

第二点是表达式指标的问题，也可以通过提前处理解决。把表达式单独转成一列，再基于这列做聚合就可以了。实际上宽表和表达式变换的处理可以用 hive 的 view，也可以生成物理表。

第三个是精确去重的问题，目前的方案是基于 Bitmap。由于数据类型的限制，目前只支持 int 类型，其它包括 long、string 等类型还不支持。因为需要把每个值都能映射到 Bitmap 里，如果是 long 的话开销太大。如果用哈希的话就会冲突，造成结果不准确。另外 Bitmap 本身开销也是比较大的，尤其跑预计算的时候，如果算出来的基数很大，对应的数据结构就是几十兆，内存会有 OOM 的风险。这些问题后面我们也会想一些办法解决，也欢迎在社区里一起讨论。（补充说明：目前已在 1.5.3 版本中实现了全类型精确去重计数的支持。）

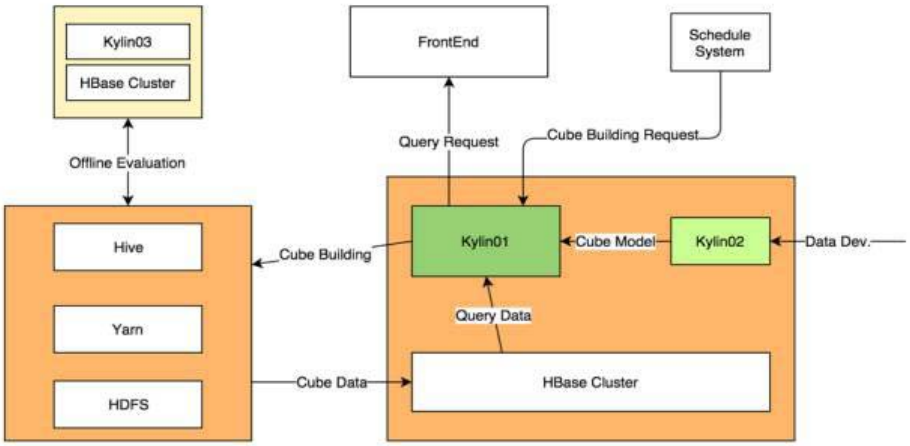


图 1

从整个系统的部署方式上来说，目前 Server 采用了分离部署的方式。Kylin Server 本质上就是一个客户端，并不需要太多资源，一般情况下使用虚拟机就能够满足需求。

实际的部署情况可以看这张图，左下角的是 Hadoop 主集群，用于执行每天所有 Hadoop 作业。中间最重要的是 Kylin01 和 02 这两个 server，是用于线上环境的 serve。其中 kylin01 是生产环境，这个环境一方面要负责从主机群上跑计算，把数据导到 HBase，另外也要响应前端的请求，从 HBase 里读数据。如果想新增一个 Cube 的话，需要在 kylin02 上操作，也就是预上线环境。所有业务方人员的 cube 数据模型定义都是在 kylin02 上做，没有问题后由管理员切到 kylin01 上。

这样做的一个好处是 kylin01 作为一个线上服务能保证稳定性，甚至权限控制能更严格一些；第二，预上线环境下开发完成后，管理员可以在投入生产前进行一次 review，保证 cube 的效率。

图 1 右上角是另外的调度系统。整个数据仓库的数据生产都是通过这个调度系统来调度的，其中的任务类型很多，Kylin 的 cube build 任务也是作为其中的一种类型。在上游的数据就绪以后，根据配置的依赖关系，自动触发 Cube 建立的过程。

图 1 左上角这边还有一个完全独立的线下测试集群，这个集群是完全开放的，主要是给用户做一些最开始的可行性调研或者评估的工作，但同时也不保证稳定性。

一个开源的系统从社区拿回来，到真正的落地，再到上生产，这个过程相对还是比较长的，这里并没有太多的技术问题，更多的是一些流程上的经验。就是如何在各个阶段给业务方提供更好的服务，使得接入 Kylin 的过程更顺畅，沟通成本更低。整个过程主要分为四个阶段。

第一个阶段是方案选型，业务方根据业务需求，选择一些方案进行调研。我们在这个阶段提供了需求的 Checklist，从数据模型，维度各个方面列出来比较详细的点，可以让业务方自己对照，确定需求是不是能够被满足。

在确定 Kylin 能满足需求的基础上，接下来是第二步，线下探查，也就是线下评估或者测试。我们提供了非常详细的接入文档，以及线下测试的环境。

第三步是线上开发，我们也有一些文档支持，基于抽象出来的场景，每个场景怎么配置 Cube，或者做哪些预处理，如何优化等，能够给业务方一个指导性的意见。

最后是开发完成后的切表上线。这个过程目前还是由管理员来操作，一方面是为了避免误操作或者滥操作，另一方面也会对 cube 进行 review，帮助进行优化。

三、主流OLAP系统对比分析

通过和其它同学交流，有一个感觉就是大家都觉得 Kylin 还不错，但并不是特别有信心，或者不知道非要用它的理由是什么，或者它和其它系统的对比是什么样的？这里也有部分测试结果可以和大家分享。

整个测试基于 SSB 的数据集，也是完全开源的，实际上是专门用于星型模型 OLAP 场景下的测试。整个测试数据集是非常标准的五张表，可以配置一些参数决定生成的数据集规模，然后在不同的规模下做不同查询场景的测试。现在已经完成的测试的系统包括：Presto，Kylin1.3，Kylin1.5 和 Druid。数据规模包含

千万、亿、十亿三种规模；维度个数为 30 个；指标个数为 50 个。典型的测试场景包括：上卷、下钻，和常用的聚合函数。

这里挑选了典型的五个查询场景：一个事实表的过滤和聚合；五张表全关联之后的查询；两个 Count Dstinct 指标和两个 Sum 指标；后面两个查询包含 8~10 个的维度过滤。

这张图是千万规模下的一个测试结果，包括了四个系统。我们在用 Kylin 或者其它系统之前没有专门用于 OLAP 分析的引擎，只能用通用的。Presto 是其中表现非常好的引擎，但是在 OLAP 这种特定的场景下，可以看到不管跟 Kylin 还是 Druid 相比差的都比较多，所以前两个测试包含了 Presto 结果，后面就没有包含了。

这里比较有趣的现象是在第三个查询，Kylin1.5 反而比 Kylin1.3 要慢一些。这个地方我们还没有搞清楚是什么原因，后面会详细的看一下。当然这个也可以证明数据没有修改过，是真实的测试数据。

从后面的两个查询上可以看到（图 2），在千万规模的级别，和 Druid 还是有比较大的差距。这主要和它们的实现模式相关，因为 Druid 会把所有的数据预处理完以后都加载到内存里，在做一些小数据量聚合的时候，可以达到非常快的

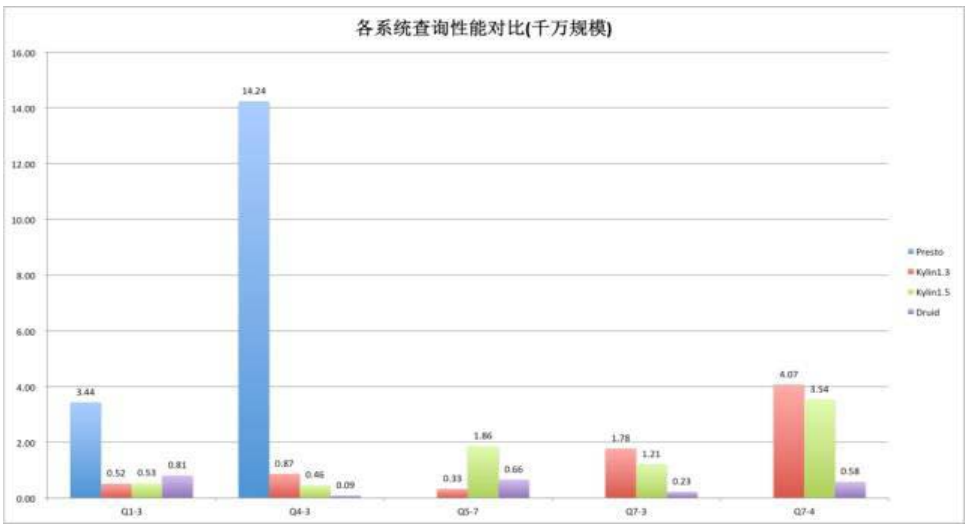


图 2

速度；但是 Kylin 要到 HBase 上读，相对来说它的性能要差一些，但也完全能满足需求。

在亿级的规模上情况又有了变化，还是看后面两个查询，Kylin1.3 基本上是一个线性的增长，这个数据已经变得比较难看了，这是由于 Kylin1.3 在扫描 HBase 的时候是串行方式，但是 Kylin1.5 反而会有更好的表现，这是因为 Kylin1.5 引入了 HBase 并行 Scan，大大降低了扫描的时间。Kylin1.5 的数据会 shard 到不同的 region 上，在千万量级上数据量还比较小，没有明显的体现，但是上亿以后，随着数据量上升，region 也变多了，反而能把并发度提上去。所以在这里可以看到 Kylin1.5 表现会更好。这里也可以看出，在数据量成数量级上升后，Kylin 表现的更加稳定，在不同规模数据集上依然可以保持不错的查询性能。而 Druid 随着数据量的增长性能损失也成倍增长（见图 3）。

刚才是在性能方面做的一些分析，其实对于一个系统来说，性能只是一个方面，除此之外，我们也会去考量其它方面的情况，主要有以下四点。

第一，功能的完备性。刚才提到我们所有的数据必须是精确的，但是现在基本上没有哪个系统能完全覆盖我们这个需求。比如 Druid 性能表现确实更好，但是它去重计数没有办法做到精确。

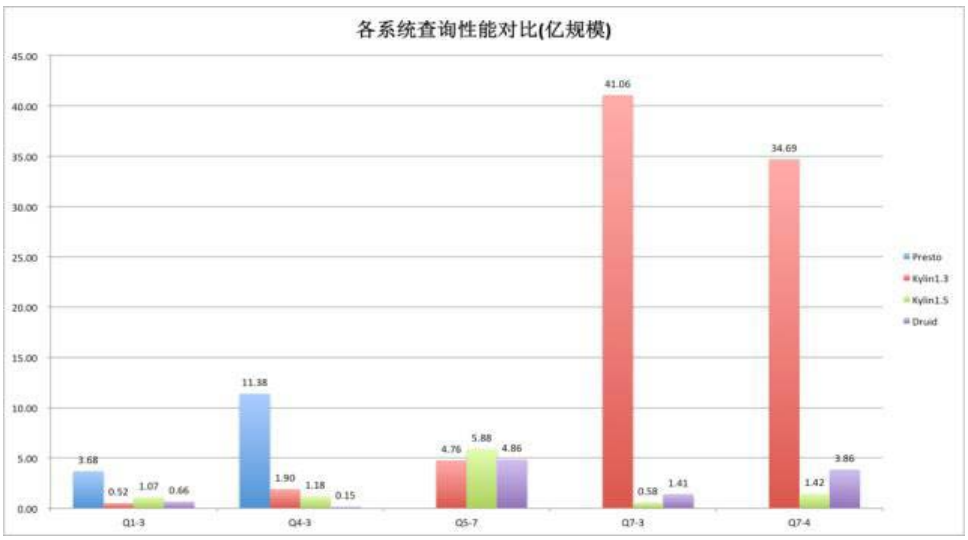


图 3

第二，系统的易用性。作为一个平台服务，不仅要把系统用起来，还要维护它，因此要考虑部署和监控的成本。这方面 Kylin 相对来说也是比较好的。Druid 一个集群的角色是非常多的，如果要把这个系统用起来的话，可能光搭这个环境，起这些服务都要很长的时间。这个对于我们做平台来讲，实际上是一个比较痛的事。不管是在部署，还是加监控的时候，成本都是相对比较高的。另外一个查询接口方面，我们最熟悉或者最标准，最好用的当然是标准 SQL 的接口。ES、Druid 这些系统原来都不支持 SQL，当然现在也有一些插件，但是在功能的完备性和数据的效率上都不如原生的支持。

第三，数据成本。刚才提到了有些数据需要做一些预处理，比如表的拉平或者表达式列的变换，除此之外还有一些格式的转化，比如有的系统只能读 TEXT 格式，这样都会带来数据准备的成本。另一方面是数据导入的效率。从数据进入数据仓库到真正能够被查询，这个时间中间有多长。数据存储和服务的时候需要多少机器资源，这个都可以归为数据成本，就是使用这个数据需要付出的成本。

第四，查询灵活性。经常有业务方问到，如果 Cube 没定义的话怎么办？现在当然查询只能失败。这个说明有的查询模式不是那么固定的，可能突然要查一个数，但以后都不会再查了。实际上在需要预定义的 OLAP 引擎上，这种需求普遍来讲支持都不是太好。

图 4 是各个系统全方位的一个对比。

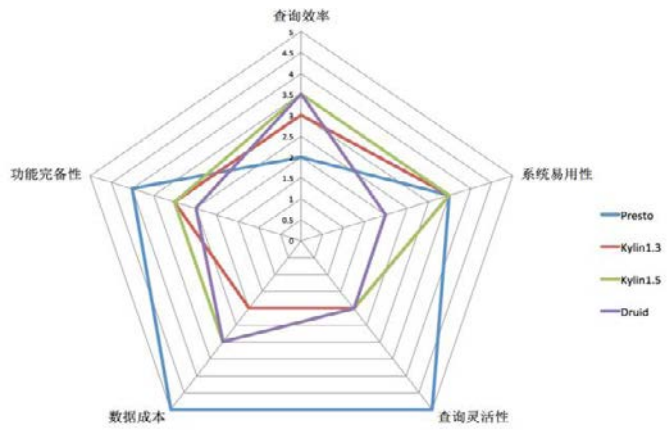


图 4

从查询效率上看，这里表现最不好的就是 Presto，表现最好的应该是 Druid 和 Kylin1.5，两者不相上下。从功能完备性上来讲，确实 Presto 语法和 UDF 等等是很完备的，Kylin 会稍微差一些，但比 Druid 好一点。

系统易用性上区别不是太大，这里主要考虑有没有标准的 SQL 接口或者部署成本高不高，用户上手能不能更快，目前来看 Druid 接口上确实不够友好，需要去翻它的文档才知道怎么去写查询的语法。

在查询成本上，Presto 是最好的，因为几乎不需要做什么特殊的处理，基本上 Hive 能读的数据 Presto 也都能读，所以这个成本非常低。Druid 和 Kylin 的成本相对较高，因为都需要提前的预计算，尤其是 Kylin 如果维度数特别多，而且不做特别优化的话，数据量还是很可观的。

最后从灵活性上来讲， Presto 只要 SQL 写出来怎么查都可以，Druid 和 Kylin 都要做一些预先模型定义的工作。这方面也可以作为大家选型时候的参考。

刚才比较客观的对比了几个系统，接下来再总结一下 Kylin 的优势。

第一、性能非常稳定。因为 Kylin 依赖的所有服务，比如 Hive、HBase 都是非常成熟的，Kylin 本身的逻辑并不复杂，所以稳定性有一个很好的保证。目前在我们的生产环境中，稳定性可以保证在 99.99% 以上。同时查询时延也比较理想。我们现在有一个业务线需求，每天查询量在两万次以上，95% 的时延低于 1 秒，99% 在 3 秒以内。基本上能满足我们交互式分析的需求。

第二、对我们特别重要的一点，就是数据的精确性要求。其实现在能做到的只有 Kylin，所以说我们也没有什么太多其他的选择。

第三、从易用性上来讲，Kylin 也有非常多的特点。首先是外围的服务，不管是 Hive 还是 HBase，只要大家用 Hadoop 系统的话基本都有了，不需要额外工作。在部署运维和使用成本上来讲，都是比较低的。其次，有一个公共的 Web 页面来做模型的配置。相比之下 Druid 现在还是基于配置文件来做。这里就有一个问题，配置文件一般都是平台方或者管理员来管理的，没办法把这个配置系统开放出去，这样在沟通成本和响应效率上都不够理想。Kylin 有一个通用的 Web Server 开

放出来，所有用户都可以去测试和定义，只有上线的时候需要管理员再 review 一下，这样体验就会好很多。

第四，最后一点就是活跃开放的社区和热心的核心开发者团队，社区里讨论非常开放，大家可以提自己的意见及 patch，修复 bug 以及提交新的功能等，包括我们美团团队也贡献了很多特性，比如写入不同的 HBase 集群等。这里特别要指出的是核心团队都是中国人，这是 Apache 所有项目里唯一中国人为主的顶级项目，社区非常活跃和热心，有非常多的中国工程师。特别是当你贡献越来越多的时候，社区会邀请成为 committer 等，包括我自己及团队成员也已经是 Apache Kylin 的 committer。同时也非常高兴看到以韩卿为首的 Apache Kylin 核心团队在今年初成立的创业公司 Kyligence，相信可以为整个项目及社区的发展带来更大的空间和未来。

四、未来工作

在未来工作方面，我们认为 Kylin 还有一些不理想的方面，我们也会着力去做优化和改进。

第一，精确去重计数。刚才提到只支持 Int，接下来有一个方案会支持所有的数据类型，能够扩展大家使用精确去重的场景范围（补充说明：目前该功能已在 1.5.3 版本中实现）。

第二，在查询效率和 Build 效率上也看到了一些可以优化的部分。比如队列资源拆分，我们所有计算集群的资源都是按照业务线核算成本的，但是现在 Kylin 本身还不太支持，这个我们也会抓紧去做，相信在很多公司也有类似的需求。还有大结果集和分页。当结果到了上百万的量级时，查询时延会上升到几十秒。同时在查询的时候有可能需要排序并且分页，就是把结果全读出来之后，根据其中的一个指标再 order by，这个开销也是比较大的。我们也会想办法进行优化。

最后，Kylin1.5 之后有明细数据和 Streaming 特性出来，后面也会做这方面的尝试。

五、Q&A

Q1：之前在 Build 的时候一直提到成本的问题，能给出一个估计值吗，如果一百亿的数据，需要多少时间？

孙业锐：有一个简单数据，大概是两亿行数据，维度的话有十四五个，Build 时间不超过两个小时，Build 出来的数据是五六百 G。

Q2：原始值是多大？

孙业锐：把这个数据抽出来之后，就是只参与 Build 的数据压缩后只有几个 G。

Q3：Kerberos 认证失效的问题你们遇到过没有？

孙业锐：Kerberos 认证完之后，会在本地临时目录下有一个 ticket 问题，每天在外部定时刷新一下就可以了，服务是不用停的。

主持人：我补充一下我们为什么选择 SQL 接口？Kylin 解决的是真正的用户面是谁，其实是业务人员和分析人员，他只会 SQL，几乎那些人很少说再学个 JAVA，所以能给他一个标准的 SQL 这个是让他上船最快的事情。其实这就是易用性很重要。

刚才看到了 Kylin 在千万级规模和亿级规模的表现，如果数据规模上到十亿，百亿，千亿的时候，我相信 Kylin 应该会秒杀所有一切。因为我们现在有另一个案例，生产环境上千亿规模的一张表，可以做到 90% 查询在 1.8 秒以内。另外我觉得非常好的一点，像美团、京东这边贡献了很多 patch，其实就是把需求提出来，大家可以一起来做。

作者介绍

孙业锐，美团高级工程师，Apache Kylin Committer。2012 年毕业于电子科技大学，曾在奇虎 360 工作，负责 Hadoop 平台建设，2015 年加入美团。目前主要负责数据生产和查询引擎的改进和优化，专注于分布式计算，OLAP 分析等领域，对分布式存储系统亦有丰富经验。

Apache Kylin
在百度地图的实践

作者 **王冬**

1. 前言

百度地图开放平台业务部数据智能组主要负责百度地图内部相关业务的大数据计算分析，处理日常百亿级规模数据，为不同业务提供单条 SQL 毫秒级响应的 OLAP 多维分析查询服务。

对于 Apache Kylin 在实际生产环境中的应用，在国内，百度地图数据智能组是最早的一批实践者之一。Apache Kylin 在 2014 年 11 月开源，当时，我们团队正需要搭建一套完整的大数据 OLAP 分析计算平台，用来提供百亿行级数据单条 SQL 毫秒到秒级的多维分析查询服务，在技术选型过程中，我们参考了 Apache Drill、Presto、Impala、Spark SQL、Apache Kylin 等。对于 Apache Drill 和 Presto 因生产环境案例较少，考虑到后期遇到问题难以交互讨论，且 Apache Drill 整体发展不够成熟。对于 Impala 和 Spark SQL，主要基于内存计算，对机器资源要求较高，单条 SQL 能够满足秒级动态查询响应，但交互页面通常含有多条 SQL 查询请求，在超大规模数据规模下，动态计算亦难以满足要求。后来，我们关注到了基于 MapReduce 预计算生成 Cube 并提供低延迟查询的 Apache Kylin 解决方案，并于 2015 年 2 月左右在生产环境完成了 Apache Kylin 的首次

完整部署。

Apache Kylin 是一个开源的分布式分析引擎，提供 Hadoop 之上的 SQL 查询接口及多维分析（OLAP）能力以支持超大规模数据，最初由 eBay Inc. 开发并贡献至开源社区，并于 2015 年 11 月正式毕业成为 Apache 顶级项目。

2. 大数据多维分析的挑战

我们在 Apache Kylin 集群上跑了多个 Cube 测试，结果表明它能够有效解决大数据计算分析的 3 大痛点问题。

- 痛点一：百亿级海量数据多维指标动态计算耗时问题，Apache Kylin通过预计算生成Cube结果数据集并存储到HBase的方式解决。
- 痛点二：复杂条件筛选问题，用户查询时，Apache Kylin利用router查找算法及优化的HBase Coprocessor解决；
- 痛点三：跨月、季度、年等大时间区间查询问题，对于预计算结果的存储，Apache Kylin利用Cube的Data Segment分区存储管理解决。

这 3 个痛点的解决，使我们能够在百亿级大数据规模下，且数据模型确定的具体多维分析产品中，达到单条 SQL 毫秒级响应。因此，我们对 Apache Kylin 产生了较高的兴趣，大数据计算查询分析的应用中，一个页面通常需要多条 SQL 查询，假设单条 SQL 查询需要 2 秒响应，页面共有 5 个 SQL 请求，总共就需要 10 秒左右，这是不可接受的。而此时，Apache Kylin 对于一个页面多条 SQL 查询响应的优势就尤为突出。

在实践过程中，根据公司不同业务的需求，我们数据智能团队的大数据 OLAP 平台后台存储与查询引擎采用了由 Apache Kylin、Impala 及 Spark SQL 组成，在中小数据规模且分析维度指标较为随机的情况下，平台可提供 Impala 或 Spark SQL 服务；在超大规模百亿级行数据的具体产品案例上，因查询性能需求较高，同时具体产品对其需要分析的维度和指标较为明确，我们使用 Apache Kylin 解决方案。下文将主要介绍 Apache Kylin 在百度地图内部的实践使用。

3. 大数据OLAP平台系统架构

主要模块（见图 1）

- 数据接入：主要负责从数据仓库端获取业务所需的最细粒度的事实表数据。
- 任务管理：主要负责Cube的相关任务的执行、管理等。
- 任务监控：主要负责Cube任务在执行过程中的状态及相应的操作管理。
- 集群监控：主要包括Hadoop生态进程的监控及Kylin进程的监控。

集群环境

因业务特殊性，我们并未采用公司内部的Hadoop集群进行计算、存储和查询，而是独立部署一台完整的集群，并独立维护。

- 集群机器：共4台，1台master(100G内存) + 3台slaves(30G内存)。
- 软件环境：CDH + Hive + HBase + Kylin 0.71

4. 基于Apache Kylin的二次开发

4.1 数据接入模块二次开发

对于任何一个数据计算处理平台，数据的接入十分关键，就像熟知的Spark，对数据接入也是十分重视。目前，我们的大数据 OLAP 平台可以支持 2 种

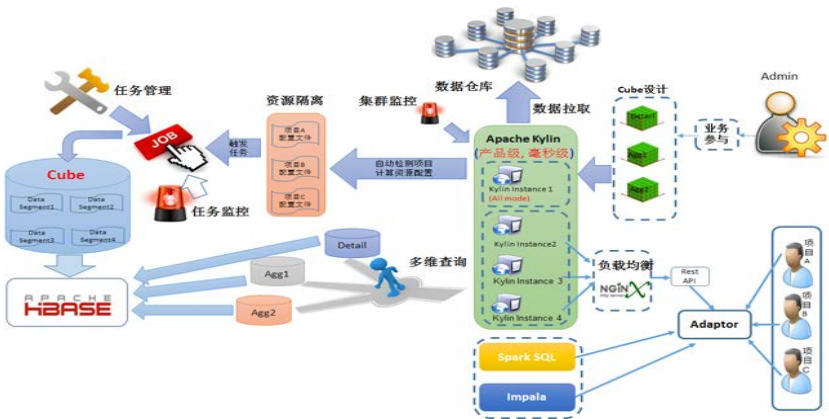


图 1

数据源的引入： MySQL 数据源及 HDFS 数据源。在实践中，我们遇到一个问题，假设 MySQL 及 HDFS 数据源没有标识表示 T-1 天的数据已经计算完成的情况下，如何确定 T-1 天的数据已经准备就绪。对于 Hive 数据源，查询数据所在 Hive Meta 的 partition 是否就绪；对于 MySQL，我们目前想到的办法是间隔一定时间循环探测当天数据行数是否变化，如果没有变化，我们基本能够简单认为第 T-1 天的数据已经由数据仓库计算完毕，接下来就可以触发数据拉取模块逻辑将数据拉取到 Master 节点的本地文件系统中，根据业务判断是否需要对这些数据细加工，然后，导入到 Master 的 Hive 中，触发事实表对应任务涉及到的所有 cube，启动 MapReduce 计算，计算结束后，前端可以刷新访问最新数据。另外，如果到了指定时间，发现数据仓库端的数据仍旧没有准备好，数据接入模块会短信报警给仓库端，并继续循环检测直至指定时刻退出（见图 2）。

4.2 任务管理模块二次开发

任务管理对于计算型平台服务十分重要，也是我们大数据 OLAP 多维分析平台的核心扩展工作之一。对于用户而言，Apache Kylin 对于 Cube 的最小存储单位为 data segment，类似于 Hive 的 partition，data segment 采用左闭右开区间表示，如 [2015-11-01，2015-11-02) 表示含有 2015-11-01 这一天的数据。对于 Cube 数据的管理主要基于 data segment 粒度，大致分为 3 种操作： 计算

图 2 数据引入模块

(build)、更新 (refresh)、合并 (merge)。对于一个具体产品来说，它的数据是需要每天例行计算到 cube 中，正常例行下，每天会生成 1 个 data segment，但可能会因为数据仓库的任务延迟，2 天或多天生成 1 个 segment。随着时间推移，一方面，大量的 data segment 严重影响了性能，另一方面，这也给管理带来了困难和麻烦。因此，对于 1 个 cube，我们按照 1 个自然月为 1 个 data segment，清晰且易管理。

假设我们有 1 个月 30 天的数据，共 23 个 data segment 数据片段，如：[2015-11-01，2015-11-02)， [2015-11-02，2015-11-04)， [2015-11-04，2015-11-11)， [2015-11-11，2015-11-12)， [2015-11-12，2015-11-13)， … [2015-11-30，2015-12-01)

问题 1：假设因为数据有问题，需要回溯 2015-11-01 的数据，因为我们能够在 cube 中找到 [2015-11-01，2015-11-02) 这样一个 data segment，满足这个时间区间，于是，我们可以直接界面操作或者 Rest API 启动这个 data segment 的 refresh 更新操作。

问题 2：假设我们需要回溯 2015-11-02 到 2015-11-03 的数据，同理，可以找到一个符合条件的 data segment [2015-11-02，2015-11-04)，然后 refresh 更新这个 data segment。

问题 3：假设我们需要回溯 2015-11-01 到 2015-11-02 的数据，我们找不到直接满足时间区间的 data segment。于是我们有 2 种解决方案，第 1 种方案是分别依次 refresh 更新 [2015-11-01，2015-11-02)， [2015-11-02，2015-11-04) 这 2 个 data segment 实现；第 2 种方案是先合并 (merge)[2015-11-01，2015-11-02)， (2015-11-02，2015-11-04) 这两个 data segment，合并后得到 [2015-11-01，2015-11-04) 这样 1 个 data segment，然后我们再拉取新数据后执行更新操作，即可满足需求。

问题 4：假设我们需要刷新 2015-11-01~2015-11-30 这 1 个月的数据，我们在另 1 套集群上基于 Kylin 1.1.1 对同一个 cube 进行测试，如果采用问题

3 中的第 1 种方案，我们需要逐步刷新 cube 的 23 个 data segment，大约耗时 17.93min X 30=537 分钟； 如果我们采用问题 3 中的第 2 种方案， 那么我们只需要将 23 个 data segment 合并成 [2015-11-01，2015-12-01) 这 1 个 data segment，计 1 次操作。然后再执行 1 次更新操作，共 2 次操作即可完成需求，总体上，耗时约 83.78 分钟，较第 1 种方法性能上提高很多。

基于上面的问题，目前我们平台对 Apache Kylin 进行了二次开发，扩展出了任务管理模块。

对于 cube 的计算 (build) 操作，假设数据仓库 2015-11-29 ～ 2015-12-02 的数据因故延迟，在 2015 年 12-03 天产出了 (T-1 天的数据)，如果不判断处理，就会例行计算生成一个时间区间为 [2015-11-29，2015-12-03) 的 data segment。所以，在每个 cube 计算前，我们的逻辑会自动检测跨自然月问题，并生成 [2015-11-29，2015-12-01) 和 [2015-12-01，2015-12-03) 两个 data segment.

对于 cube 的更新 (refresh) 操作，我们会采用问题 3、问题 4 中提到的第 2 种方案，自动合并 (merge)data segment 后再执行更新 refresh 操作。因为上面已经保证了不会有跨月 data segment 的生成，这里的自动合并也不会遇到生成跨自然月的情况。

对于 cube 的合并 (merge) 操作，如果每天都自动合并该自然月内前面日期已有的所有 data segment，假设我们想回溯更新 2015-11-11 这一天的数据，那么就需要回溯 (2015-11-01，2015-11-12)（因为这个时间区间的 data segment 每天都被自动合并了），其实，我们没有必要回溯 2015-11-01~2015-11-10 这 10 天的数据。所以，对于 1 个自然月内的 cube 的数据，在当月，我们先保留了 1 天 1 个 data segment 的碎片状态，因为在当月发现前面某几天数据有问题的概率大，回溯某个 data segment 小碎片就更加合理及性能更优。对于上个月整个月的数据，在下个月的中上旬时数据已经比较稳定，回溯的概率较小，通常要回溯也是上个月整月的数据。因此，在中上旬整体合并上 1 个月的数据而不是每

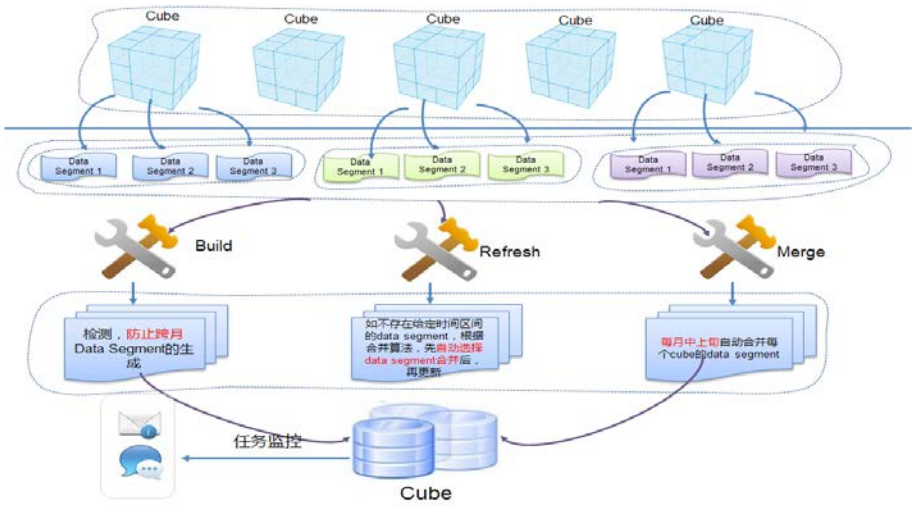


图 3 任务管理模块

天合并更合理（见图 3）。

4.3 平台监控模块二次开发

4.3.1 任务监控

通常，1 个产品对应多个页面，1 页面对应 1 个事实表，1 个事实表对应多个 cube，那么一个产品通常会包含多个 cube，上面提到的 cube 基于 data segment 的 3 种任务状态，很难人为去核查，所以对于任务执行的监控是非常必要的，当任务提交后，每隔一段时间检测一次任务的状态，任务状态中间失败或者最后成功后，则会发送邮件或者短信报警通知用户。

4.3.2 集群监控

由于我们的服务器是团队内部独自部署维护，为了高效监控整套 Hadoop 集群、Hive，HBase、Kylin 的进程状态，以及处理海量临时文件的问题，我们单独开发了监控逻辑模块。一旦集群出现问题，能够第一时间收到报警短信或者邮件（见图 4）。

4.4 资源隔离二次开发

由于我们以平台方式提供给各个业务线使用，当某个业务线的业务数据计算规模较大，会造成平台现有资源紧张时，我们会根据实际情况，要求业务方提供机器资源，随之而来的就是如何根据业务方提供的机器资源分配对应的计算

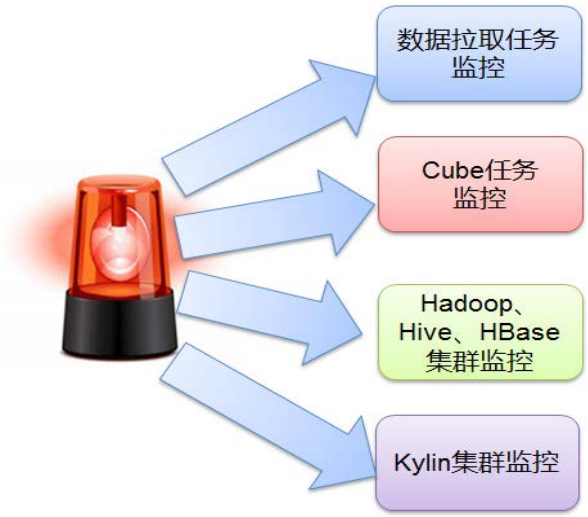


图 4 平台监控模块

队列的资源隔离问题。目前，官方的 Apache Kylin 版本对于整个集群只能使用 1 个 kylin_job_conf.xml，平台上所有项目的所有 Cube 的 3 种操作只能使用同一个队列。于是，我们基于 kylin-1.1.1-incubating 这个 tag 的源码做了相关修改，支持了以项目为粒度的资源隔离功能，并提交 issue 到 https://issues.apache.org/jira/browse/KYLIN-1241，方案对于我们平台管理员自身也参与项目开发的应用场景下非常适用。对于某个项目，如果不需要指定特定计算队列，无需在 $KYLIN_HOME 下指定该项目的 kylin_job_conf.xml 文件，系统会自动调用官方原有的逻辑，使用默认的 Hadoop 队列计算（见图 5）。

4.5 Hadoop及HBase优化

因独立部署的 Hadoop 集群硬件配置不高，内存十分有限，所以，在项目实践过程中也遇到不少问题。

4.5.1 Hadoop任务内存资源不够，cube计算失败

调整 MapReduce 分配资源参数：在 cube 计算过程中，会出现 mr 任务失败，根据日志排查，主要因 mr 的内存分配不足导致，于是，我们根据任务实际情况整体调整了 yarn.nodemanager.resource.memory-mb,mapreduce.map.memory.mb，mapreduce.map.java.opts，mapreduce.reduce.memory.mb 及 mapreduce.reduce.java.opts 等参数。

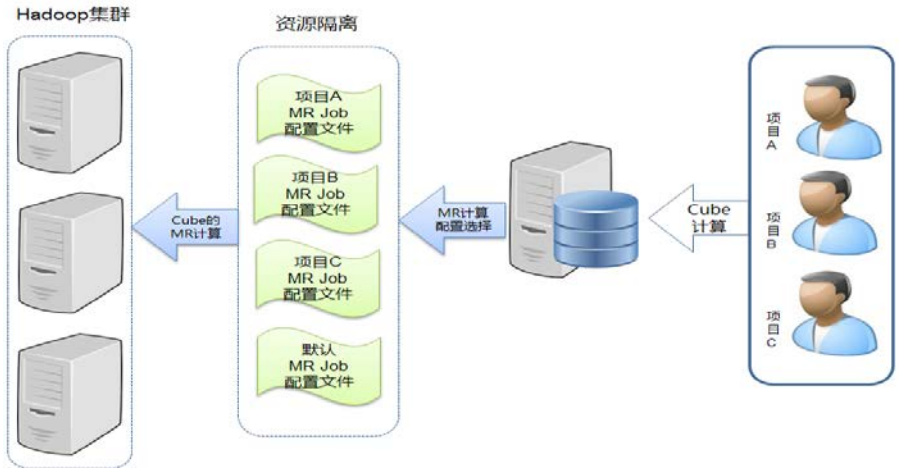


图 5 资源隔离

4.5.2 HBase RegionServer在不同节点随机down掉

由于机器整体资源限制，我们给 HBase 配置的 HBASE_HEAPSIZE 值较小，随着时间推移，平台承载的项目越来越多，对内存及计算资源要求也逐步提高。后来平台在运行过程中，HBase 的 RegionServer 在不同节点上出现随机 down 掉的现象，导致 HBase 不可用，影响了 Kylin 的查询服务，这个问题困扰了团队较长时间，通过网上资料及自身的一些经验，我们对 HBase 和 Hadoop 相关参数做了较多优化。

A. HBase 的 JVM GC 相关参数调优，开启了 HBase 的 mslab 参数：可以通过 GC 调优获得更好的 GC 性能，减少单次 GC 的时间和 FULL GC 频率；

B. HBase 的 ZK 连接超时相关参数调优：默认的 ZK 超时设置太短，一旦发生 FULL GC，极其容易导致 ZK 连接超时；

C. ZK Server 调优，提高 maxSessionTimeout：ZK 客户端（比如 Hbase 的客户端）的 ZK 超时参数必须在服务端超时参数的范围内，否则 ZK 客户端设置的超时参数起不到效果；

D. HBASE_OPTS 参数调优：开启 CMS 垃圾回收期，增大了 PermSize 和 MaxPermSize 的值；（见图 6）

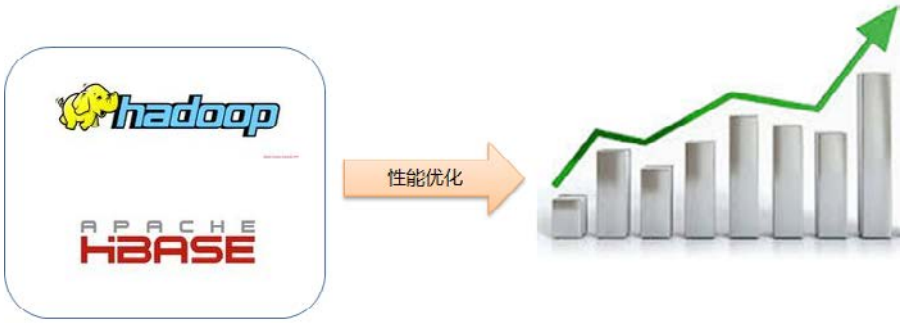


图 6 Hadoop 及 HBase 优化

5. Apache Kylin项目实践

5.1 基于仓库端join好的fact事实表建Cube，减少对小规模集群带来的hive join压力

对于 Cube 的设计，官方有专门的相关文档说明，里面有较多的指导经验，比如：cube 的维度最好不要超过 15 个，对于 cardinality 较大的维度放在前面，维度的值不要过大，维度 Hierarchy 的设置，等等。

实践中，我们会将某个产品需求分为多个页面进行开发，每个页面查询主要基于事实表建的 cube，每个页面对应多张维度表和 1 张事实表，维度表放在 MySQL 端，由数据仓库端统一管理，事实表计算后存放在 HDFS 中，事实表中不存储维度的名称，仅存储维度的 id，主要基于 3 方面考虑，第一：减少事实表体积；第二：由于我们的 Hadoop 集群是自己单独部署的小集群，MapReduce 计算能力有限，join 操作希望在仓库端完成，避免给 Kylin 集群带来的 Hive join 等计算压力；第三：减少回溯代价。 假设我们把维度名称也存在 Cube 中，如果维度名称变化必然导致整个 cube 的回溯，代价很大。这里可能有人会问，事实表中只有维度 id 没有维度 name，假设我们需要 join 得到查询结果中含有维度 name 的记录，怎么办呢？对于某个产品的 1 个页面，我们查询时传到后台的是维度 id，维度 id 对应的维度 name 来自 MySQL 中的维度表，可以将维度 name 查询出来并和维度 id 保存为 1 个维度 map 待后续使用。同时，一个页面的可视范围有限，查询结果虽然总量很多，但是每一页返回的满足条件的事实表记录结果

有限，那么，我们可以通过之前保存的维度 map 来映射每列 id 对应的名称，相当于在前端逻辑中完成了传统的 id 和 name 的 join 操作。

5.2 Aggregation cube辅助中高维度指标计算，解决向上汇总计算数据膨胀问题

比如我们的事实表有个 detail 分区数据，detail 分区包含最细粒度 os 和 appversion 两个维度的数据（注意： cuid 维度的计算在仓库端处理），我们的 cube 设计也选择 os 和 appversion，hierarchy 层次结构上，os 是 appversion 的父亲节点，从 os+appversion(group by os, appversion) 组合维度来看，统计的用户量没有问题，但是按照 os(group by os) 单维度统计用户量时，会从基于这个 detail 分区建立的 cube 向上汇总计算，设上午用户使用的是 android 8.0 版本，下午大量用户升级到 android 8.1 版本，android 8.0 组合维度 + android 8.1 组合维度向上计算汇总得到 os=android(group by os, where os=android) 单维度用户，数据会膨胀且数据不准确。因此我们为事实表增加一个 agg 分区，agg 分区包含已经从 cuid 粒度 group by 去重后计算好的 os 单维度结果。这样，当用户请求 os 维度汇总的情况下，Apache Kylin 会根据 router 算法，计算出符合条件的候选 cube 集合，并按照权重进行优选级排序（熟悉 MicroStrategy 等 BI 产品的同学应该知道这类案例），选择器会选中基于 agg 分区建立的 os 单维度 agg cube，而不从 detail 这个分区建立的 cube 来自底向上从最细粒度往高汇总，从而保证了数据的正确性。

5.3 新增留存类分析，如何更高效更新历史记录?

对应小规模集群，计算资源是非常宝贵的，假设我们对于某个项目的留存分析到了日对 1 日到日对 30 日，日对 1 周到日对 4 周，日对 1 月到日对 4 月，周对 1 周到周对 4 周，月对 1 月到月对 4 月。那么对于传统的存储方案，我们将遇到问题。

5.3.1 传统方案

假如今天是 2015-12-02，计算实际得到的是 2015-12-01 的数据。

日期	日对第1日留存	日对第2日留存	日对第3日留存	日对第30日留存	月对4月留存	新增用户
2015-12-01	0	0	0	0	0	1234
2015-11-30	123	0	0	0	0	2341
2015-11-29	234	432	0	0	0	4232
2015-11-28	242	2323	323	0	0	3292
. . .						
. . .						
. . .						

图 7

图 7 数据存储方案的思路是，当今天是 2015-12-02，那么 2015-12-01 可以计算活跃用户了，于是，我们会将 2015-11-30 的日对第 1 日留存， 2015-11-29 的日对第 2 日， 2015-11-28 的日对第 3 日等的这些列指标数据进行更新（如上红色对角线部分），这是因为每天数据的每 1 列都是以当天为基准，等今后第 n 天到了，再回填这 1 天的这些第 x 日留存，如此，对于 1 个任务会级联更新之前的多天历史数据，如上红色对角线的数据。

此方案的优势：

- 如果要查看某个时间范围内的某一个或者多个指标，可以直接根据时间区间，选择需要的列指标即可。
- 如果要查看某1天的多个指标，也可以直接选择那1天的多个指标即可。

此方案的缺点：

- 每天都需要更新历史数据，如上红色对角线的数据，造成大量MapReduce任务预计算cube，需要较多的机器计算资源支持。
- 如果今后增加新的留存，比如半年留存，年留存，那么对角线长度就更长，每天就需要回溯更新更多天数的历史数据，需要更多时间跑任务。
- 对于级联更新的大量的历史数据任务，其实依赖性很强，如何保证留存项目多个cube每一天的多个data segment级联更新正确，非常复杂，难以维护和监控，对于数据仓库端也易遇到如此问题。
- 对于需要批量回溯一个较大时间区间的历史数据时，问题3中涉及的任

务计算难点和困难尤为突出。

5.3.2 变通方案

假如今天是 2015-12-02，我们计算实际得到的是 2015-12-01 的数据（图 8 可和上面的结构对比）

此方案的思路是，当今天是 2015-12-02，实际是 2015-12-01 的数据，如上示例存储，但日对第 n 日的留存表示的是 n 日前对应的那个日期的留存量，相当于旋转了红色对角线。

此方案的优势：

- 如果要查看某个时间范围内的某1个指标，直接选择该范围的该列指标即可。
- 如果今后增加新的留存，比如半年留存，年留存等指标，不需要级联更新历史天数的数据，只需要更新2015-12-01这1天的数据，时间复杂度O(1)不变，对物理机器资源要求不高。

此方案的缺点：

- 如果涉及到某1天或者某个时间范围的多列指标查询，需要前端开发留存分析特殊处理逻辑，根据相应的时间窗口滑动，从不同的行，选择不同的列，然后渲染到前端页面。

目前，我们在项目中采用变通的存储方案。

日期	日对第1日留存	日对第2日留存	日对第3日留存	日对第30日留存	月对4月留存	新增用户
2015-12-01	123	432	323	2312	2341	1234
2015-11-30	234	2323	2131	1232	1243	2341
2015-11-29	242	1432	2323	3421	2321	4232
2015-11-28	242	2323	3233	3422	4322	3292
. . .						
. . .						
. . .						

图 8

6. 总结

目前，我们大数据 OLAP 多维分析平台承载百度地图内部多个基于 Apache Kylin 引擎的亿级多维分析查询项目，共计约 80 个 cube，平均半年时间的历史数据，共计约 50 亿行的源数据规模，单表最大数据量为 20 亿 + 条源数据，满足大时间区间、复杂条件过滤、多维汇总聚合的单条 SQL 查询毫秒级响应，较为高效地解决了亿级大数据交互查询的性能需求，非常感谢由 eBay 贡献的 Apache Kylin，从预计算和索引的思路为大数据 OLAP 开源领域提供了一种朴素实用的解决方案，也非常感谢 Apache Kylin 社区提供的支持和帮助。

作者简介

王冬，百度地图数据智能组成员，北京理工大学计算机本硕毕业，2012 加入 Microstrategy，负责 BI Server 核心组件 SQL Engine 相关开发。并于 2014 年加入百度地图数据智能组，主要负责大数据 OLAP 多维分析计算方向研究，热爱大数据离线、实时平台建设应用、Spark 生态应用等。

Apache Kylin
在网易云音乐的实践和调优

作者 **冯宇**

【编者按】本文来自网易杭州研究院在实际使用 Apache Kylin 中的经验总结，特别是对网易云音乐的分析需求，深入解构业务特性，加之团队对 Kylin 的深入研究，从业务和技术角度突破，转换思路，最终将查询性能从最初的上百秒提升到 1 秒以内，取得了非常好的效果。

背景

最近开始使用了新版本的 Apache Kylin，在此之前对于新版本的了解只是代码实现和一些简单的新功能测试，但是并没有导入实际场景的数据做分析和查询，线上 Hadoop 稳定之后，逐渐得将一些老需求往新的环境迁移，基于以前的调研，新版本（V2，版本为 1.5.2）的 Apache Kylin 提供了几个比较显著的功能和优化：

- 新的度量类型，包括TopN、基于bitmap的精确distinct count和明细数

据查询等。

- 自定义度量框架，用户可以定义一些特殊的度量需求。
- Fast Cubing算法，减少MR任务，提升build性能。
- 查询优化，Endpoint Coprocessor，HBase并行扫描和数据压缩，这部分对于查询性能提升还是比较显著的。
- Shard hbase存储，基于一致性哈希的rawkey分布，尽可能的将大的cuboid分散到不同的region上，增加并行扫描度。
- Spark计算引擎，in memory准实时计算，这两项目前还处于试验阶段。
- 新的aggregation group分区算法。

有了这么多新的特性和性能提升，经常拿新版本来应付用户的需求：新版本实现了 xxx 功能，肯定性能会很好，等到新版本稳定了再来搞这个需求吧。等到我们的新版本上线了，业务需求真正上来之后，才发现用起来没有相当的那么简单，对于 Apache Kylin 这种直接面向用户需求的服务，对于一些特殊的需求更是要不断打磨和调整才能达到更好的性能。本文整理在接入网易云音乐一个较为复杂的需求的实现和中间调优的过程，一方面开拓一下自己的思路，另外也使得读者能改更好的使用 Apache Kylin。

业务需求

数据通过日志导入到 Hive 中查询，经过一定的聚合运算，整理成如图 1 中的格式。这个数据源是针对歌曲统计的，每一首歌曲包含歌曲属性信息（歌曲名、歌手 ID，专辑 ID 等）、支付方式、所属圈子 ID、标签信息等，需要统计的指标包括每一首歌曲的各种操作的 PV 和 UV，操作包括播放、收藏、下载等 10 种。因此一共 20 左右个指标，然后需要按照这 20 个指标的任意一个指标计算 TOP N 歌曲和其他统计信息（N 可能是 1000,1W 等），除此之外，在计算 TOP N 的时候还需要支持字段的过滤，可以执行过滤的字段包括支付方式、圈子 ID、标签等。歌曲 ID 全部的成员数大概在千万级别，该表的数据每天导入到 hive，经过初步

图 1

的聚合计算生成图 1 中的格式，基本上保证了每天的每一个用户对于每一首歌曲只保留一条记录，每天的记录数大概在几亿条（聚合之后），查询通常只需要查询不同条件下的 TopN 的 song_id。查询性能尽可能的高（秒级），需要提供给分析人员使用（见图 1）。

基于kylin 1.x版本的实现

Apache Kylin 1.3.0 之前的版本对于这种需求的确有点无能为力，我们使用了最简单的方式实现，创建一个包含所有维度、所有度量的 cube，其中 10 个度量为 count distinct，使用 hyperloglog 实现的近似去重计数算法，这种做法能够可以说是最原始的，每次查询时如果不进行任何过滤则需要从 hbase 中扫描几百万条记录才能满足 TopN 的查询，每一条记录还都包含 hyperloglog 序列化的值（读取到内存之后还需要对 hyperloglog 进行反序列化），TOP 的查询速度可想而知，正常情况（没有并发）下能够在 1、2 分钟出结果，偶尔还会有一个查询把服务搞挂（顺便吐槽一下我们使用对的虚拟机）。查询以天的数据就这种情况了，对于跨天的数据更不敢尝试了。

这种状况肯定无法满足需求方，只能等着新版本落地之后使用 TopN 度量。

需求简化

面对这样的现实，感觉对于去重计数无能为力了，尤其像这种包含 10 个 distinct count 度量的 cube，需求方也意识到这种困境，对需求做出了一定的让步：不需要跨天计算 TopN 的 song_id 了，只需要根据每一天的 PV 或者 UV 值

计算 TopN，这样就不需要 distinct count 了，而且还能保证每天的统计值是精确的。如果希望计算大的时间周期的 TopN，则通过创建一个新的 cube 实现每个自然周、自然月的统计。对于这样的需求简化，可以大大提升 cube 的创建，首先需要对数据源做一些改变（这种改变通常是 view 来完成），将原始表按照歌曲 ID、用户 ID 和其它的维度进行聚合（group by，保证转换后的新表每一个用户对于每一首歌曲只保存一条记录），PV 则直接根据计算 SUM（操作次数），UV 通过表达式 if(SUM(操作次数) > 0, 1, 0) 确定每一个 userid 对每一个 songid 在当天是否进行了某种操作，这样我们就得到如图 2 中格式的表。

这个表中可以保证歌曲 ID+ 用户 ID 是唯一的（其它的维度是歌曲的属性，一个歌曲只对应相同的属性），对于 Kylin 来说原始数据量也变小了（还是每天几亿条记录），接下来新建的 Cube 由于数据源保证每一条记录不是同一个用户对同一首歌曲的操作，因此所有的度量都可以用 SUM 来实现（去重计数也可以通过对 sum(是否 xxx) 统计精确的计数值），这样对于 Kylin 的负担减少了一大部分，但是扫描记录数还是这么多（Cube 中最大的维度为歌曲，每次查询都需要携带，查询时扫描记录数始终保持在歌曲 ID 的个数），但是 Build 性能提升了，查询性能也由于扫描数据量减少（一个 SUM 度量占用的存储空间只有 8 字节）而提升，但是还是需要 30+ 秒。

此时，新版本终于在公司内部落地了，仿佛看到了新的曙光。

Apache Kylin新版本（1.5.x）的实现

新版本有了 TopN 度量了，但是目前只支持最大 TOP 1000，可以通过直接改

图 2

cube 的 json 定义改变成最大得到 5000，于是尝试了几种使用 TopN 的方案。

第一种是将所有维度放在一个 Cube 中然后对于每一个 PV 和 UV 创建一个 TopN 度量。这样 build 速度还是挺快的，但是查询时大约需要 25 秒 +，最坑爹的是，创建 TopN 时需要指定一个指标列（做 SUM，根据该列的 SUM 值排序）和一个聚合列（歌曲 ID），使用一个聚合列创建的多个 TopN 度量居然只有查询第一个度量时才有结果，使用其他指标查询 TopN 时返回的聚合值总是 null，难道我要为每一个指标都建一个 cube，每一个只包含一个 TopN 度量？

第二种方案是不使用全部维度，而只使用在计算 TopN 时需要做过滤的列，例如支付类型、圈子 ID 等维度建包含 TopN 的 Cube，这样通过抽取部分维度创建一个小 Cube 用来计算 TopN，然后再使用计算出来的一批批歌曲 ID 从大 Cube（包含全部维度的 cube）中取出这个 ID 对应的其它指标和属性信息。这样做的弊端是对于查询不友好，查询时需要执行两条 SQL 并且在前端进行分页，如果性能可以接受也就认了，但是现实总是那么残酷，当创建了一个这样的 Cube（包含歌曲 ID 和其他几个成员值很少的维度）进行 build 数据时，让人意想不到的事情发生了，计算第二层 Cuboid 的任务居然跑了 3 个小时才勉强跑完，接下来的任务就更不敢想象了，MR 任务的 task 使用 CPU 总是 100%，通过单机的测试发现多个 TopN 的值进行 merge 的性能非常差，尤其是 N 比较大的时候（而且 Kylin 内部会把 N 放大 50 倍以减少误差），看来这个方案连测试查询性能的机会都没有了。

尝试了这两个方案之后我开始慢慢的失去信心了，但是想到了 Kylin 新版本的并行扫描会对性能有所提升，于是就创建了一个小 cube（包含歌曲 ID 和几个过滤维度，和全部的 SUM 度量）用于计算 TopN，整个 cube 计算一天的数据只需要不到 1G 的存储，想必扫描性能会更好吧，查询测试发现在不加任何过滤条件的情况下对任意指标的 TopN 查询大约不到 15s，但是扫描的记录数仍然是几百 W（歌曲 ID 的数目），可见并行扫描带来的性能提升，但是扫描记录数仍然是一个绕不过去的坎。

在经历这么多打击之后，实在有点不知所措了，感觉歌曲 ID 的数目始终是

限制查询性能的最大障碍，毕竟要想计算 TopN 肯定要查看每一个歌曲 ID 的统计值，然后再排序，避免不了这么多的扫描，唯一的方法也就是减小 region 的大小，使得一天的数据分布在更多的 region 中，增大并行度，但这也不是最终的解决方案啊，怎么减小扫描记录数呢？

从业务出发

陷入了上面的思考之后，提升查询性能只有通过减少扫描记录数实现，而记录数最少也要等于歌曲 ID 的成员数，是否可以减少歌曲 ID 的成员数呢？需求方的一个提醒启发了我们，全部的 20 个指标最关键的是播放次数，一般播放次数在 TopN 的歌曲，也很有可能在其他指标的 TopN 中。那么是否可以通过去除长尾数据来减少歌曲 ID 的个数呢？首先查了一下每天播放次数大于 50 的个数数量，查询结果对于没接触过业务的我来说还是很吃惊的，居然还剩几十 W，也就意味着通过排除每天播放次数小于 50 的歌曲，可以减少将近 80%+ 的歌曲 ID，而这些歌曲 ID 对于任何指标的 TopN 查询都是没有任何作用的，于是通过 view 把数据源进行在过滤和聚合，创建包含全部维度的 cube，查询速度果然杠杠的，只需要 2s 左右了！毕竟只需要扫描的记录数只有几十万，并且都是 SUM 的度量。

终于达成目标了，但是这种查询其实是对原始数据的过滤，万一需求方需要查询这些播放次数比较少的歌曲 ID 的其他指标呢？完全不能从 Apache Kylin 中获取啊。此时，业务方的哥们根据数据特性想到了一个很好的办法：为播放次数建索引！根据播放次数的数值创建一个新的维度，这个维度的值等于播放次数所在的区间，例如播放次数是几位数这个维度的值就是多少，或者使用 case when sum（播放次数）自定义不同的范围。例如某一个歌曲在当天 SUM（播放次数）为千万级，那么这个歌曲对应的 index 维度值为 8，百万级则值为 7，以此类推。然后查询的时候根据 index 过滤确定播放次数的范围，这样可以大大减少歌曲 ID 的数目，进而减少扫描记录数。

此时需要对图 2 中的记录格式再进行整理，通过 group by 歌曲 ID,xxx（其

图 3

他维度）统计出每一首歌曲的操作次数和操作人数，然后再根据操作次数的值确定 index 的值。得到新的记录格式如图 3.

说干就干，整理完数据之后再添加一个 index 维度创建新的 Cube，查询的时候可以先查询 select 歌曲 ID, sum(指标) as c from table where index >= 7 group by 歌曲 ID order by c desc limit 100，使用着这种查询可以直接过滤掉所有播放次数小于百万的歌曲 ID，但是经过测试发现，这种查询时间还是 15s 左右，和上面的方案性能并无改善。整得我们开始怀疑人生了（见图 3）。

回归技术

接下来感觉是走投无路之后的各种猜测，例如猜测 shard by 设置 true or false 是否对性能有很大的影响，经过查看源码发现 shard by 只是为了决定该列的值是否参与一致性哈希中桶号的计算，只不过是为了更好的打散 rowkey 的分布，对于存储和查询性能提升不是太明显，思来想去还是不合理，既然加了过滤不应该还要扫描全部的歌曲 ID，是不是 Cube 中 rowkey 的顺序错了，查了一下 cube 的定义，发现 index 维度被放在 rowkey 的最后一行，这显然是不了解 Apache Kylin 是如何确定扫描范围的！ Kylin 存储在 hbase 中的记录是 rowkey 是根据定义 Cube 时确定的 rowkey 顺序确定的，把如果查询的时候对某一个维度进行范围过滤，Kylin 则会根据过滤的范围确定扫描的区间以减小扫描记录数，如果把这个维度放到 rowkey 的最后，则将这种过滤带来的减小扫描区间的作用降到了最低。

重新修改Cube把index维度在rowkey中的位置提升到最前面之后重新build，此后再执行相同的查询速度有了质的飞跃，降低到了1s以内。根据index的范围不同扫描记录数也随着改变，但是对于查询TopN一般只需要使用index >= 6的过滤条件，这种条件下扫描记录数可以降低到几万，并且保证了全部歌曲ID可以被查询。

总结

本文记录了网易云音乐的一个比较常见的TopN的需求的优化，在整个优化的过程中在技术和业务上都给与了充分的考虑，在满足查询需求的前提下，查询性能从最初的上百秒提升到1秒以内，这个过程让我充分意识到了解数据特征和业务的重要性，以前遇到问题总是想着是否能够进行算法的优化，是否可以加机器解决，但是当这些都不能再有改进的空间时，转换一下思路，思考一下是否可以从业务数据入手进行优化，可能会取得更好的效果。

随着Apache Kylin新版本的落地使用，对于业务的支持也更有信心了，尤其是并行查询和Endpoint Coprocessor对性能的提升，但是对于开源技术来说，如何正确的使用才是最困难的问题，我们也会在探索Apache Kylin使用的路上越走越远。

最后，通过本次调优，我发现了一个以前没有注意到的问题：薛之谦最近真的挺火，不相信的话可以查看一下网易云音乐热歌榜。

作者介绍

冯宇，网易杭州研究院后台技术中心大数据组，从市Apache Kylin的开发，运维等工作，对OLAP技术，SQL on Hadoop引擎等有浓厚的兴趣。

Apache Kylin
在 Hadoop 上的超高速数据查询

作者 **李扬**

Apache Hadoop 到今天已经十年，作为一个平台很好的解决了大数据的存储和运算的基本需求。伴随着技术的成熟，业界对大数据的需求也越来越高。从最早的NoSQL，最近几年又回归到了SQL-on-Hadoop，而在不断涌现的各种技术之中，有着不一样的设计理念和哲学。本文是在这样的背景下作者对大数据系统的一些思考。

什么是O(1)级的高速查询

大数据与传统技术最大的区别就在于数据的体量对查询带来的巨大挑战。从最早 NoQL 不支持 SQL 查询，到 Apache Hive 开始支持类 SQL(HQL)，技术在不断进步。现如今 SQL on Big Data 呈现百家争鸣的格局。除了 Apache Hive，还有 Apache Drill、Apache Impala、Apache Kylin、Apache Phoenix、Apache HAWQ、Presto、SparkSQL 等开源技术，以及 SAP HANA、Teradata、HP Vertica 等商业产品。查询的速度也从最早的几个小时缩短到了几分钟甚至几秒。面对各种五花八门的测评报告和各执一词的对比结果，要回答“谁是查询速度最快的大

数据系统”并不是一个简单的问题。

其实算法原理对程序的执行速度有很清晰的定义，称为时间复杂度。对应到大数据查询，可以概括为查询速度与被查询数据总量之间的关系。在给定硬件条件下，假设数据总量为 N，查询时间随数据量增长而线性增长，被称为 O(N) 时间复杂度；查询时间不随数据量的增长而增长，被称为 O(1) 时间复杂度（见图 1）。

当数据成倍增长时，O(1) 的查询速度有着绝对的优势。有可能在数据量较小时 O(N) 和 O(1) 系统的查询速度差别不大。但我们处在数据爆炸的年代，便携智能设备的普及和物联网的到来决定了数据的持续增长是不可逆转的趋势。假设未来数据量增长 100 倍，那 O(N) 系统的查询时间也将延长至从前的 100 倍。即使现在快到 1 秒的查询也会变成 100 秒的长时间等待（虽然有各种优化手段和技术，例如缓存，索引等可以降低一些这方面的查询延迟，但本质还是 O(N) 复杂度）。

因此以发展的眼光看，只有 O(1) 级别的查询速度才是唯一的出路。只有 O(1) 级的查询速度才能保证在数据成倍增长的同时保持不变的查询速度，才能称得上是超高速的大数据查询能力。

大数据查询的关键技术

那百家争鸣的大数据世界，谁拥有 O(1) 级的查询速度呢？

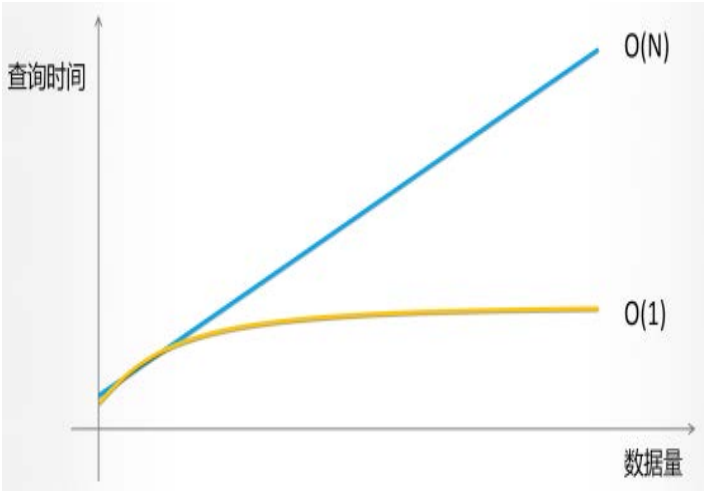


图 1

回答这个问题得先来回顾一下常见的大数据关键技术。上述众多的大数据系统，它们从本质上都是建立在这些关键技术之上的。这些关键技术也决定了数据查询的时间复杂度。

- 大规模并行计算：即Massive Parallel Processing，指将计算分布到多个处理器或计算机上并行处理。最理想的情况假设不考虑调度的额外开销，原来1台机器计算1小时的查询，分散到60台机器并行计算就能缩短到1分钟，达到提速的效果。
- 列式存储：即Columnar Storage，指将记录按列连续存储，在查询时按需读取，跳过不需要的列，达到减少存储访问时间，提高查询速度的效果。比如一个表有50列，一个查询只用到其中的10列。那么列式存储就能大约减少4/5的存储访问时间。
- 倒排索引：即Inverted Index，指建立从内容到存储位置的索引。查询时，根据过滤条件使用倒排索引能直接定位到符合条件的记录，而无需逐行扫描全表，缩短查询处理时间。
- 内存加速：指用内存代替硬盘，加速数据访问。内存比硬盘快十至十万倍，近年内存的持续降价，用内存来存储（部分）大数据正变得越来越可行。

考虑数据量增长 100 倍时，这些技术能否保持查询时间不变呢？首先排除内存加速，它不能减少任何新增的数据访问和计算。其次排除列式存储和倒排索引。它们都能减免不必要的存储访问，但如果新增的数据是有用数据，包含符合查询要求的记录，那也就必须访问这些新增记录，因此所需的处理时间也依旧成百倍增长。最后来看大规模并行计算，貌似通过 100 倍地扩容计算集群可以提升等比例的计算能力。但成百倍的任务调度会带来性能损失，更要命的是 100 倍的数据交换会让网络成为新的系统瓶颈，最后还有百倍的 IT 投资和维护预算。另外，如果数据的增长如果是 1000 倍，10000 倍呢？因此考虑现实的情况，大规模并

行计算也不能保持查询速度。

难道没有技术能在超大规模数据集上实现 O(1) 的查询速度吗？Apache Kylin 提出的“预计算”技术给了我们新的思路。

要查询百倍的数据，就必须投入百倍的计算，这是毋庸置疑的。但计算可以分为离线预计算和在线计算两部分。假设在用户查询开始之前，我们就能预判查询的大致方向，预先处理数据，完成最复杂的连接、聚合、排序等运算，那么当用户查询时，就只需完成剩余的少量计算，快速得到查询结果了。预计算并没能减少总的查询计算量，而是巧妙地利用时间差，在查询开始之前对数据做预处理，用离线预计算代替在线计算的方法，让用户感受到的查询速度变快了。

- 多维立方体：即OLAP Cube，是一种预计算技术，将数据分为维度和度量，预先按维度汇总度量的各级聚合值并保存。使用户可以交互式地对数据做多维分析（如上卷、下钻等）的一种技术。

除了提高查询速度，预计算还能极大的提升查询吞吐量，支持更多的并发查询和在线用户。因为在线计算被离线预计算代替，完成一次查询所需的CPU、内存、网络等计算资源都远远小于传统的大数据查询，所以同样的计算机集群，使用预计算后能支撑的并发查询数一般是原来的几十甚至几千倍。

O(1)级的高速查询系统

总结上述的大数据关键技术，一个能对无限数据提供常数级查询速度的 O(1) 高速查询系统大致如图 2 所示。

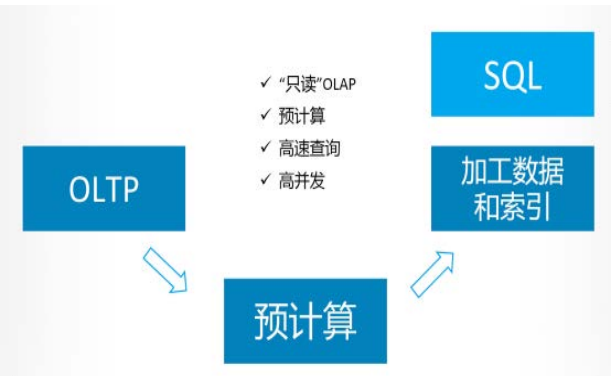


图 2

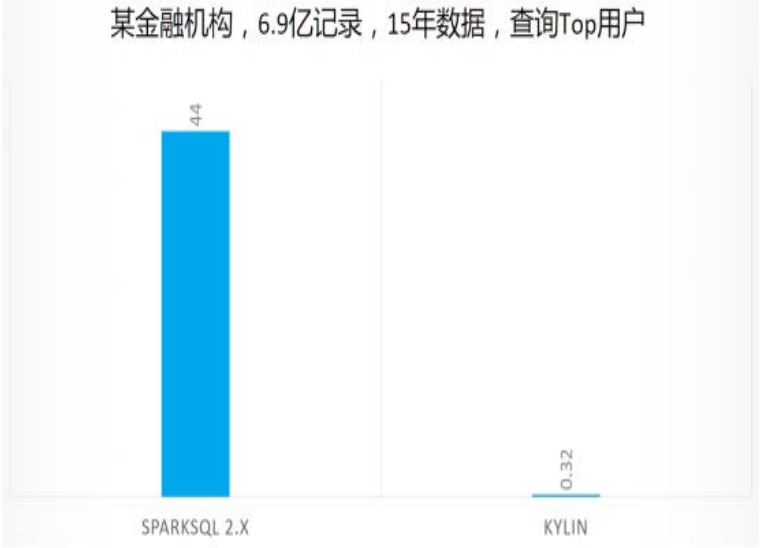


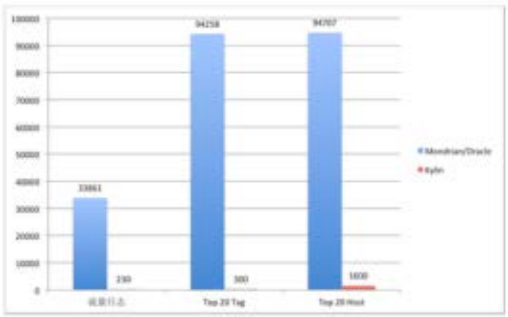
图 3

它是一个对查询做高度优化的 OLAP 系统。数据的导入一般通过批处理完成，导入过程也即是预处理过程。预处理后的加工数据和索引保存在分布式存储系统中，使用压缩、编码和列式存储优化存储访问。在线查询通过 SQL 引擎接入，利用并行计算完成少量的在线运算，快速得出查询结果。

Apache Kylin 是现有的大数据系统中，唯一一个着重于预计算技术，符合上述模型的系统。这也是为什么它在千亿乃至万亿的大数据场景中有那么出色的性能表现。几个简单的例子可以给我们更清晰的感性认识。

某金融机构曾将 Apache Kylin 与 SparkSQL 2.x 比较测试。在跨度 15 年、总量 6.9 亿的一个数据集上查询交易量最大的一组用户 (Top2000)。Apache Kylin 的查询响应时间是 0.32 秒，而 SparkSQL 是 44 秒。这就是 O(N) 与 O(1) 时间复杂度的差距（测试在同一个集群和数据集上进行，见图 3)。

另在网易的测试报告中，Apache Kylin 与 Oracle 数据仓库进行多维分析的对比。不仅在 SQL 查询上 Apache Kylin 要快 100 倍以上，更在吞吐率上通过线性扩容，轻易达到每秒 200 个查询以上。这对传统的基于大规模并行计算的数据仓库来说是难以想象的（见图 4）。



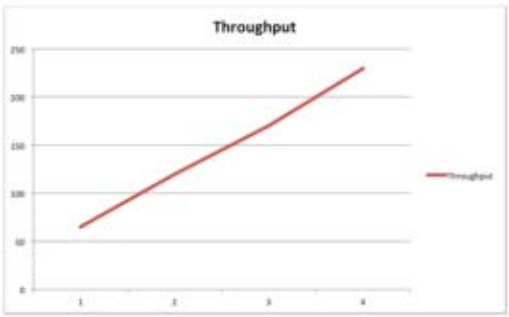


图 4

小结

在 SQL on Hadoop 百花齐放的今天，各类大数据系统层出不穷。然而在理论上可以清晰地看到，唯有“预计算”才能在数据爆炸式增长时保证常数 O(1) 级别的查询速度，实现真正的高速大数据分析。相信在不久的将来，随着物联网、智慧星球的到来，这一点将变得更加明朗。

这样的高速查询也为大数据在线服务打开了窗口。试想全球数据能够在手指轻轻点击下快速查询，交互分析。这将彻底改变大数据产业的生产力，激发数据潜能，改变未来。

作者介绍

李扬，Kyligence 联合创始人兼 CTO，Apache Kylin 联合创建者及项目管理委员会成员 (PMC)， 主创团队架构师和技术负责人，专注于大数据分析，并行计算，数据索引，关系数学，近似算法，压缩算法等前沿技术。曾任 eBay 全球分析基础架构部大数据资深架构师、IBM InfoSphere BigInsights 的技术负责人，负责 Hadoop 开源产品架构，“杰出技术贡献奖”的获奖者、摩根士丹利副总裁，负责全球监管报表基础架构。

专访韩卿：Kylin 是如何成为 Apache 顶级项目的

作者 **郭蕾**

2015 年 12 月 8 日，Apache 基金会批准 Apache Kylin 正式毕业成为 Apache 的顶级项目。Apache Kylin（麒麟）是由 eBay 研发并贡献给开源社区的 Hadoop 上的分布式大规模联机分析（OLAP）平台。Kylin 于 2014 年 10 月开源，并于当年 11 月成为 Apache 孵化器项目，是 eBay 第一个贡献给 Apache 软件基金会的项目，也是第一个由中国团队完整贡献到 Apache 的项目。为了了解 Apache Kylin 的开源发展历程以及经验，InfoQ 记者采访了该项目的负责人韩卿。

InfoQ：请简单介绍下 Apache Kylin 的开源发展历程以及目前的项目状态。

韩卿：Apache Kylin 是在 2014 年 10 月 1 日由 eBay 在 github.com 上开源，之后在业界获得了非常积极的反馈，相关 Hadoop 方面的资深成员提议让我们申请加入 Apache 孵化器以获得更好的发展，并愿意做我们的 Mentor。经一系列工作后于 2014 年 11 月 25 日正式加入 Aapche 孵化器项目，经过不到一年的时间，Apache Kylin 正式成为 Apache 顶级项目，其为 eBay 全球贡献至 Apache 软件基金会（ASF）的第一个项目，也是全部由在中国的华人团队整体贡献至 Apache 的第一个项目。

目前 Aapche Kylin 社区的有 5 位 mentor，13 位 PMC 成员（来自 eBay，美

团，明略数据等）及众多 contributor （GitHub 上显示有代码提交的有 31 位，还有更多在邮件列表，JIRA 及其他地方的贡献者），我们正在发展新的一批的 committer。目前 eBay 团队有 7 位成员全职参与此项目。

在 eBay，已经上线两个生产环境平台，有着诸多的应用，包括用户行为分析、点击分析、商户分析、交易分析等应用，最新的 Streaming 分析项目也已经上线。目前在 eBay 平台上最大的单个 cube 包含了超过 850 亿的数据，90% 查询响应时间小于 1.5 秒，95% 的查询响应时间小于 5 秒。

此外，社区中也有包括来自百度地图、京东、美团、唯品会、明略数据、Expedia 等各个公司的诸多应用已经上线使用。

InfoQ：开源一年的时间，就成为了 Apache 的顶级项目，并且全部由中国团队运营，这是中国开源历程里的一个重要里程碑。回顾这一年，你们在开源项目运营上，都做了哪些工作？

韩卿：非常感谢对 Apache Kylin 项目及我们团队和社区的褒奖。在项目伊始，我们就面临无数的挑战，包括技术、产品以及其他各个方面。除了在内部积极推动项目，吸引更多应用来使用 Kylin 平台外，开源项目的运营也是工作的重点。作为 Kylin 产品及开源社区负责人，在项目开源之前就定下了“发展社区”及”构建生态系统”两个重要的方向。

- 发展社区

正如 Apache 一直强调的：Community over Code。Apache Kylin 开源后，社区发展一直是对外的工作重点之一，通过各种渠道扩展线上线下社区，并且积极参与和组织各种活动和 Meetup，比如 Apache Kylin Meetup，Spark Meetup 等等，参加国际国内各个行业大会，包括 Strata + Hadoo World 伦敦，Hadoop Summit 硅谷，大数据技术大会，数据库技术大会，QCon，ApacheCon 等。通过媒体网站，社交媒体，微信等进一步扩展 Apache Kylin 的知名度，吸引了非常多对 Apache Kylin 技术有兴趣的公司和个人参与进来，之后整个团队非常积极的响应各种问题，及时修复各种 Bug，为赢得初期客户打下了坚实基

础，由此 Apache Kylin 的社区也初步建立起来。随着后续相应版本的稳定发布，各个不同案例的成功应用，Kylin 社区越来越活越，更多的 contributor 更多的 committer 不断加入，发展了包括来自美团，京东，明略数据等多位 committer，并正在发展新的一批 committer，同时也和其他开源社区形成了良好的互动，包括 Apache Zeppelin，Apache Calcite 等。

- 构建生态系统

一个应用很难单独的存在与一个企业中，不管是商业产品还是开源项目。从一开始，我们就定下了只关注核心功能，尽可能与整个产业链中的其它产品，项目及公司进行合作的方向。比如在前端展现方面和 Tableau 进行充分合作，在存储方面依靠 HBase 等。Apache Kylin 的生态圈图从第一版到现在没有太多的变化，只是增加了更多的朋友，例如 Apache Zeppelin 等，随着新版架构的改进，整个项目将与 Spark，Kafka， Excel/PowerBI， Docker 等形成更好的互补和整合，积极融入整个大数据生态圈并打造自己的生态系统。

- 积极学习Apache社区的运作方式

作为第一个 Apache 项目，从公司到团队到个人都没有特别多的这方面的经验。因此我们在加入 Apache 后积极学习和遵循 Apache 社区及项目的运作方式，特别是 The Apache Way、Community Over Code 等。另外整个团队与我们的 mentor 们形成了很好的互动，在各个方面获得了他们极大的指导和帮助，整个团队，社区一起不断进步和发展，为我们顺利毕业成为顶级项目提供了前提。

InfoQ：有人说过，开源社区也是混圈子，能详细阐述下这个『感受』吗？

韩卿：首先，社区是什么？技术社区不是一个在线论坛或者微信群，而是围绕一个产品或者技术，具有共同兴趣或者爱好而所形成的人与人之间的联系和互动。所以仅仅只是参与在线论坛或者邮件列表，提交代码，贡献补丁是远远不够的，需要不断的认识朋友，与不同的人就行交流，不断碰撞新的想法，积极与其他人进行互动等。在硅谷，经常有各种 meetup、user group 等组织这方面的活动，比如 Spark、Hive、Kafka 等等，通过线下活动将线上的 ID 映射到活生生的人的

时候，圈子自然形成了，朋友自然就交到了，如果你的技术或者产品很吸引人，自然会有更多人愿意来帮助。另外一方面，混熟悉后，不管如何当需要帮忙的时候就方便的多了。这方面我们国内的开发者社区还远远不够，不过已经看到了非常不错的交流氛围，也希望更多的朋友能够参与到国际社区的交流中，这样当你需要找人投票，找人帮忙 review code 的时候就容易的多，也是一个很好的机会向国际社区展现来自我们本土的技术力量和产品等。

InfoQ：目前国际上有几个大的基金会，如果要捐献，就基金会的选择方面，你有什么心得？

韩卿：目前国际上有 Apache 软件基金会，Linux Foundation 基金会，Open Stack 基金会，容器基金会等，由于本人较少参与其他基金会，这里仅给出一些简单的个人看法：

如果你要捐献你的项目至开源社区，为其选择合适的基金会很重要，比如大数据（Hadoop / Spark 等）相关，则 ASF 是比较好的地方，绝大部分的大数据相关项目目前都在 Apache 旗下，如果是云计算方向，则 OpenStack 基金会则是很好的地方，如果是微软技术系列的项目，则最新的 .NET 基金会是更好的选择。

选择合适的基金会将会为项目带来诸多便利，相关参与者及贡献者一般都活跃在特定的基金会及相关的社区中，在那里较容易获得更多的关注和帮助。比如在 Apache 中很容易找到 Hadoop、Spark 相关人士，但云计算方向就比较少。

InfoQ：Apache Kylin 项目做的很棒，在这段时间里，公司的支持一定是你们强有力的后盾吧。就公司的开源战略和投入方面，你有什么经验？ Apache Kylin 的成功对 eBay 有什么意义？

韩卿：Apache Kylin 项目的成功离不开 eBay 公司的支持，最初的立项、团队、客户案例等都是从 eBay 内部挖掘的，虽然我们在一开始就定了要开源的方向，但我们设定了内部必须有真实案例上生产环境才可以对外开源，事实上，Kylin 在 eBay 内部的生产环境正式于 2014 年 9 月 30 日上线，同时有三个应用案例，在隔天即 2014 年 10 月 1 日我将代码 push 到了 GitHub 上。此外管理层也给了巨

大的支持，从美国总部到 eBay 中国卓越技术中心（CCOE）都给予了充分的信任和支持，为团队能够非常专注在产品设计，技术研发，应用开拓上提供了必要的支持。

eBay 公司一直参与开源社区，贡献了很多项目至开源世界，特别是从 2013 年开始，公司决策层将采用，参与及贡献回开源社区作为战略指导，鼓励各个团队参与和贡献至开源社区，Apache Kylin 在项目一开始就获得 eBay 高级副总裁的直接 comment: Ready Open Source from Day One。

Apache Kylin 所获得的影响在公司内外部引起了积极反响，也获得了非常高的评价，从 CTO 到高级副总裁到部门 VP 等都给予了极大的赞赏和鼓舞。特别是 Apache Kylin 在今年的 InfoWorld Bossie 大奖中与 Apache Spark，Apache Kafka，Druid，Apache Flink 等一同荣获“最佳开源大数据工具”奖，是业界对整个项目的认可，也在 eBay 内部引起了巨大的影响，也为后续的其他项目进一步贡献至开源社区带来了一个好的开端，来自我们部门的另一个项目 Apache Eagle 已经于 2015 年 10 月正式成为新的 Apache 孵化器项目。

InfoQ：接触了这么长时间的 Apache 社区，有什么感想和读者分享吗？

韩卿：相对于直接在 github.com 或者其他地方开源方式，Apache 社区及基金会给人的感觉会更加“古板”和“官僚”，比如到目前为止，讨论社区都还是以邮件列表（文本形式）为主，还不能使用 Google Group 等“现代”工具。讨论，设计，决策，版本发布等都需要遵守一定的规则来完成，甚至很多时候以为一些细节问题不得不重新生成发布包并重新投票。

但是，这些方式，以及 ASF 一直推崇的 The Apache Way 为一个开源项目带来的是更好的治理（Governance）和品质，相信每一位朋友和我的感觉一样，一个来自 Apache 的项目或者 framework 是可以放心使用的，基本上不用担心有什么大的问题，更新上也会有一定的保证，这比在 GitHub 上看到一个好项目，但几个月甚至一年都没有更新，要靠谱的多的多。

而且 Apache 软件基金会会定期 review 不活跃的项目，重新寻找贡献者，或

者直接 retire 该项目，通过这样的方式保证了 Apache 社区中的项目都具有稳定的活跃度和较高的品质。

对于愿意参与开源项目，个人非常推荐多多参与 Apache 相关项目，一方面可以为开源项目做贡献，另一方面也是一个很好的机会可以学习西方文化，学习其他项目如何运作；对于有兴趣贡献项目的朋友，Apache 软件基金会是个不错的选择，如果你的项目希望加入 Apache 孵化器，可以联系我，希望我们在这方面的一些经验可以对大家有一些帮助。

InfoQ：接下来 Apache Kylin 有什么规划?

韩卿：我们正在准备 Apache Kylin 2.0 的发布工作，预计 2015 年年底前会发布一个稳定的beta版本,该版本将支持StreamingOLAP及可插拔架构(Plugable Architecture）。之后将更关注在实时 OLAP, Spark 生态整合，快速的明细数据查询，更多友好的客户端（包括 Zeppelin，Excel，PowerBI 及其他 BI 和展现工具），完善的 SQL 功能及高级函数支持，以及稳定性和易用性等方面。

更多的关于 Kylin2 的特性及计划，请关注 Apache Kylin 的微信公众号（ApacheKylin）或者 Twitter 帐号（@ApacheKylin）以获得最新的信息。

关于韩卿（Luke Han）

现任 eBay 全球分析基础架构部 (ADI) 大数据产品负责人， 负责包括 Apache Kylin，Apache Eagle 及其它大数据相关产品的设计，规划，战略和执行。并且作为 Apache Kylin co-creator & VP 管理和驱动 Apache Kylin 的愿景，路线图，特性及计划等，在全球各地不同部门中发展客户，开拓内外部合作伙伴及管理开源社区等，建立与大数据厂商，集成商及最终用户的联系已构建健壮的 Apache Kylin 生态系统。在此之前任 eBay BI 平台高级架构师，带领团队为 eBay 全球商务智能平台提供管理，架构，开发等，在大数据，数据仓库，商务智能等方面拥有超过十年的工作经验。

版权声明

InfoQ 中文站出品

架构师特刊：Apache Kylin 实践

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