40 | 十年一梦，一起来看Facebook的数据仓库变迁（二）

2022-01-28 徐文浩

《大数据经典论文解读》 课程介绍 >

讲述：徐文浩
时长 13:55 大小 12.75M

你好，我是徐文浩。

上节课里，我们一起学习了 2010 年 Facebook 的数据仓库的整体架构。我们看到，Facebook 是采用了 **容错 + 分层 + 优化** 这样的三重手段，来搭建自己的数据仓库体系。他们部署了多个不同职责的 Hadoop 集群，这些集群分工明确、各司其职，让数据分析师的 Adhoc 任务，重要但又不那么重要的生产任务，以及有着严格的完成时间的生产任务，共同运转在自己的数据仓库上。

不过，只有这强大的基础设施还是不够的。对于大数据系统的构建来说，还有两个重要的问题值得关注。

对于海量的数据表，我们如何让内部用户能够快速把我们的数据用起来？

对于这样一个数据系统，我们如何来判断那么多用户到底遇到了什么问题和困扰？

那么今天这节课，我就带你来看看 Facebook 是如何回答这两个问题的。回答了这两个问题之后，整个 Facebook 数据仓库体系也就完整地呈现在你面前了。

基于“事实”数据的协同方式

虽然在开发 Hive 这个系统的时候，Facebook 其实已经考虑了如何让用户能够把数据用起来。所以，在我们研读 Hive 的论文的时候，就聊过 Hive 相比于 Pig 的一大优势，就是有着**良好的元数据管理**。而我们在看 Twitter 的大数据整体架构的时候，也看到他们是通过 HCatalog 来补上 Pig 缺失的这一环的。

尽管在 Hive 上运行 SQL，本质上是执行一系列的 MapReduce 任务，它也仍然是一种高延时的数据批处理的方式。只不过，数据分析师们仍然习惯于传统 BI 工具那种，有界面可以交互的形式。所以，Facebook 也直接提供了一个类似的交互界面，这个界面在 Facebook 内部叫做 **HiPal**。

通过 HiPal，数据分析师一方面可以查询到当前 MapReduce Job 的进度、检查 SQL 运行的结果、并且可以上传小数据集，或者 SQL 查询结果本身不大的话，也可以以 CSV 格式下载下来。尽管在性能体验上，Hive 和传统的 BI 工具体验不太一样，但是在其他方面，Facebook 通过 HiPal，就让数据分析师们的体验尽量接近传统的 BI 工具。

在交互体验之外，Facebook 也遇到了和 Twitter 类似的问题，那就是**如何让使用内部数据仓库的用户知道有哪些数据可以用**。

在 2010 年，Facebook 的 Adhoc 集群里有超过 2 万张表，想要找到有什么数据可以用，算得上是一个巨大的挑战。这个问题，在论文中被称之为数据发现问题（Data Discovery）。这些数据是由公司内部不同的团队生成、维护、使用的，而为了公司本身能够快速向前推进，这种把“使用数据”的权限给到每一个小团队和每一个人，是不可或缺的手段。

那么，Facebook 解决问题的思路，其实和 Twitter 也是非常类似的，就是直接在系统内提供各种协同功能。Facebook 在内部创建了几个工具。

首先是能够直接针对数据表的 Metadata 创建 **Wiki**，用户可以添加、修改、纠正数据表，以及对应各个列的描述信息，并且可以为这些表加上能够搜索的标签和项目名称。这样，使用这些数据表的工程师和分析师们可以不断更新信息。而且即使表的 Schema 和数据集本身发生了变化，对应可以搜索的信息也会不断有用户去修改，使得文档会适配于最新的信息。

其次，Facebook 内部写了一系列的工具，来提取不同数据表之间的“**谱系**”，也就是某一个数据集是通过哪些其他的数据集计算生成出来的。这个表明了上下游数据集依赖关系的“谱系”信息，也让使用数据表的用户可以直接浏览。

进一步地，Facebook 还直接让你可以看到数据集中的一小部分真实的数据，让你可以更容易地在上面撰写 SQL。

相信经常要写 SQL 的同学一定会觉得这个功能非常实用，因为很多时候我们通过 Schema 还是不知道 SQL 该怎么写。比如订单表里有一个 order_time 字段，即使我们知道它是一个表示下单时间的整型字段，我们也不知道它究竟是用秒数还是毫秒数来表示。而直接提供一些表里面的真实数据，会让我们很容易知道字段实际到底是什么含义。

更有意思的一点是，Facebook 还会通过工具**自动分析**每一张表。把最近查询这张表比较频繁的用户找出来，把他们标记为“专家用户”。并且，你可以直接在这个数据仓库系统内联系这些用户，去询问他们相关的问题。

而所有这些工具，都是**内置**在上面所说的 HiPal 这个交互式 Web 查询界面里的。我前面也说了，这个思路其实和我们之前看到的 Twitter 的解决方案是类似的，Facebook 直接把数据仓库的“文档”这件事情，内置到了系统内部，并且能够让所有的用户一起来协同更新。

只不过，比起 Twitter 来，Facebook 更进了一步。在协同层面也不仅仅提供 Wiki 这样的异步协作方式，还会**自动挖掘**出每一张表的“专家”是谁，让数据分析师能够在系统内直接同步沟通，更快地得到反馈。

如何管理定时任务的依赖关系？

除了由数据分析师主动发起的 Adhoc 的查询分析之外，我们日常的大数据系统里，还会有大量每天定时运行的任务。在论文里，这样的任务被称之为 Periodic Batch Jobs。不同的

任务之间也有相互的依赖关系，而 Facebook 通过了一个叫做 **Databee** 的系统来管理依赖关系。

不过这个倒是没有那么稀奇，在 2010 年，开源世界里已经有很多这样的任务调度管理工具了，包括 Apache 体系里的 Oozie、LinkedIn 开源出来的 Azkaban 等等。而到了今天，Apache 旗下的 Dolphin 和 Airflow 也都是这样的工具。

这些工具都是为了解决一个问题，那就是很多数据报表，需要一系列的任务组合起来。以我们之前讲解过的 Twitter 为例，一个任务 A 会从原始日志生成 session_sequence 这样的 session 数据。然后有各种不同的任务 B、C、D 都是分析各种不同的漏斗数据，它们都要依赖 session_sequence 数据的生成。也就是说，B、C、D 都依赖于 A 任务先完成。

那么，最简单但是也是最不可靠的办法，自然是依赖于不同任务执行的时间。我们让任务 A 在凌晨 2 点运行，然后让 B、C、D 在凌晨 3 点运行。我们的假设，是任务 A 一定会在 1 个小时内运行完成。

但是，这个假设总会遇到程序出现 Bug、系统出现故障的时候。当任务 A 在 3 点没有运行完成，那么 B、C、D 要么会读到不完整的数据，给出错误的计算结果，要么干脆会运行失败。

当然，这个问题并不难解决，我们可以在 A 的任务完成后，往系统里**写入一个标记**。这个标记可以是数据库里的一条记录，也可以是 HDFS 上的一个空文件。然后，3 点运行的 B、C、D 任务可以去检查这个标记。如果标记已经生成了，那么 B、C、D 这些后续任务就可以照常运行。而如果标记没有生成，那就等待一段时间，再去检查标记。

当我们把这样的机制直接内置到系统里，只要申明任务之间的依赖关系，各个任务就会自动生成对应的标记，以及去检查标记，其实就是 Databee 或者其他开源的调度依赖系统的功能了。而且，有了依赖关系之后，我们也不用再去估算 B 这个任务应该在几点运行了，只要申明 B 要等待 A 完成就好了。一旦 A 完成了，B 这个任务就会启动。

这样，我们不仅解决了依赖关系的管理，还会让 B 的任务的完成时间尽可能地早。顺带地，这样就让我们各个报表的完成时间和延时，变得尽可能地小了。

而除了管理任务之间的依赖关系之外，这些调度工具还会去统计和监测各个任务花费的时间，以及延时。这些数据，也会帮助我们在整个数据处理流程出现 Bug 或者延误的时候，能够做好 debug 工作。

我们应该监控些什么数据？

我们通过 HiPal 这个系统，解决了用户如何发现有什么数据可以使用，通过 Databee 这个系统，解决了用户的任务之间的依赖管理问题。这两个问题被解决，也就意味着我们已经解决了数据仓库日常使用最显性的问题了。那么，在这之后，我们如何进一步地发现内部用户在使用 Hive-Hadoop 集群中，会遇到哪些问题呢？

答案就是**监控**。

第一层的监控自然包含传统的对于集群的硬件和使用资源的监控。这里面既包括系统级的 CPU 占用、I/O 活动，以及内存使用这样的监控；也包括特定的对于 Hadoop 集群的监控，像是 Job Tracker 里的 JVM 的堆的使用率，以及 GC 发生的频率。相信只要是用上了 Hadoop 集群的公司，都有这样的基础监控。

不过，这些监控只是给 Hive-Hadoop 集群的运维人员看的。而 Facebook 进一步地统计了一系列的数据，来给各个使用 Hive-Hadoop 集群的用户去看。而这些监控包含了两个方面。

第一个方面，是每一个你负责的 Job 到底消耗了多少资源。Facebook 开发了一个叫做 **htop** 的工具，来统计每一个 Job 在整个集群里，一共花费了多少 CPU、内存资源。这件事情，其实也是每一个使用大数据系统的团队都应该去做的。

我自己在搭建的大数据系统里，也会按人、按组、按任务类型去统计这些信息。因为当我们把大数据系统做得很好用的时候，就会面临一个甜蜜的烦恼：越来越多并不了解和熟悉 MapReduce 底层机制的用户，可以简单地写一个 SQL 就分析大量的数据。而因为系统方便好用，通过 SQL 去分析海量的数据也很容易被滥用。往往一个数据分析师原本只是为了做一些简单的统计查询工作，可是回头一看，一个月下来消耗的硬件资源就高达好几百万人民币。

这样的情况是在我的身边真实发生过的。有工程师写了一个复杂的嵌套了多个表关联的 SQL，这个 SQL 跑在 Google Cloud 的 BigQuery 上。然而第二天一看，就这一句

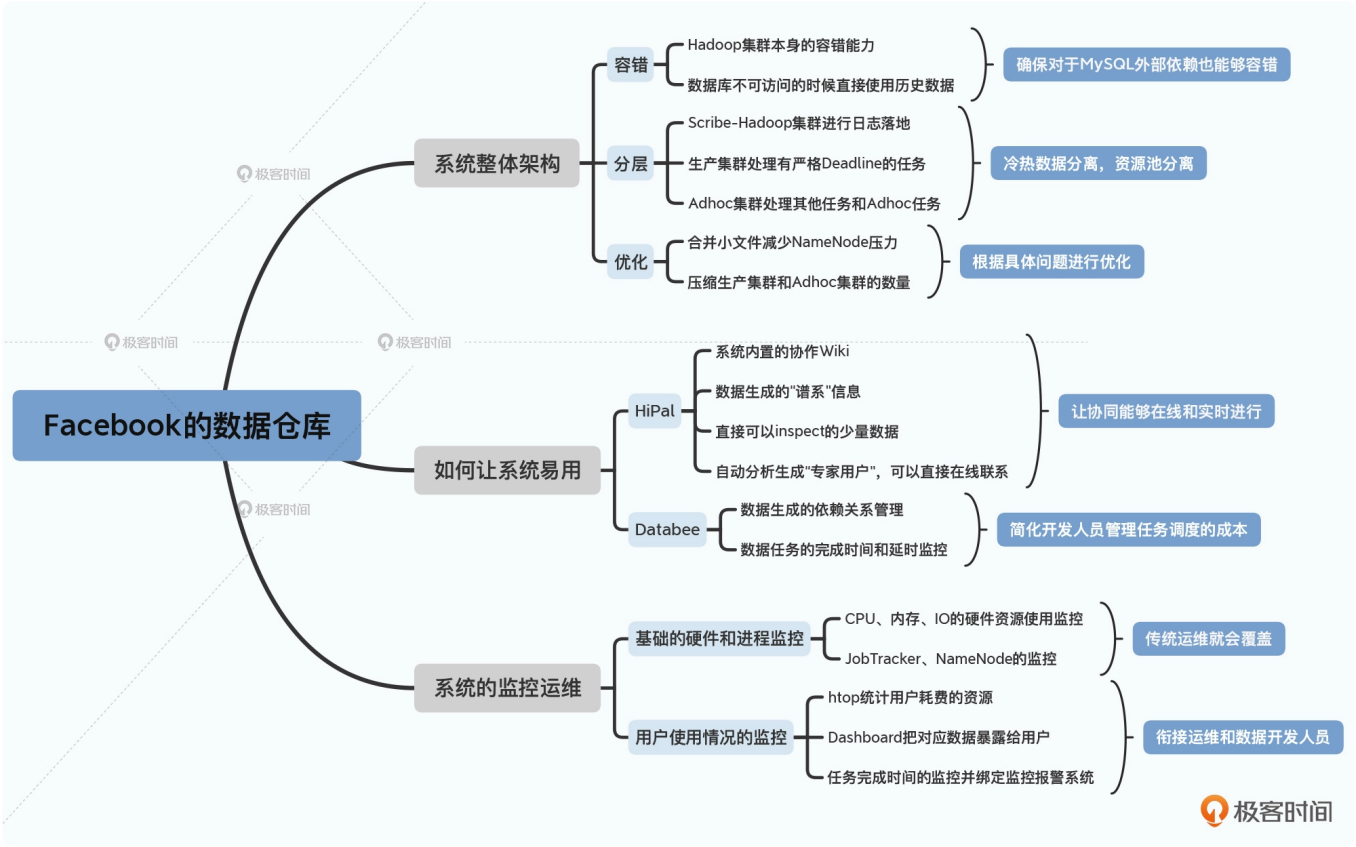
SQL，就带来了 15 万美元的一张账单。而去统计每一个分析师使用的硬件资源，并将这些数据反馈在数据看板上，可以让数据分析师们对于实际占用的硬件资源有感知，也可以在消耗大量资源的任务刚刚冒头的时候，就尽量优化。

第二个方面，则是去监控每一个数据集，特别是那些定期生成的数据集的发布时间。需要这样一个监控程序，也是因为越来越多的数据分析师而不是工程师，参与到了数据体系的建设中来。比起习惯了添加监控程序、衡量系统 SLA 的工程师们，数据分析师可以快速写出 SQL，把各类报表发布上线。

不过，他们往往对于让这些报表能够稳定地、每天自动按时生成，缺少了一些职业习惯带来的强迫症。而与其让数据分析师们都去变成一个个工程师，不如就直接在监控体系中带上这些功能。通过一个运维层面的自动化手段，来减少无论是工程师、还是数据分析师不得不去花费时间，以保障系统可用性的问题。

可以看到，在真实的大数据数据仓库的场景中，固然传统的对于集群的硬件和运行状况的监控很重要，而每一个用户如何使用系统、消耗了多少资源，以及他们的任务是否真的按时完成、让这个数据链路是稳定可用的，也同样重要。前者往往很容易受到运维团队的重视，但是后者常常在很多团队中是割裂或者缺失的。

这些事情，运维团队可能会觉得不归自己管，工程师和分析师团队也觉得不归自己管。常常会出现，每个人都觉得自己做好了自己的工作，但总是感觉系统的资源不够用、数据报表生成总是延时的情况。这个，也是我们在搭建大数据体系的过程中，需要额外重视的一点。



Facebook数据仓库体系的拆解图

小结

好了，到了这里，相信你也了解 Facebook 的数据仓库的全貌了。虽然这是一篇 10 年前的论文，不过却并不过时。因为它所讲解的，并不是某一个具体的分布式数据库的内部原理，而是在一个数千个工程师、数万张表的数据仓库里，人们该如何去运营、维护和管理。

回顾过去两节课的内容，Facebook 给出的解决方案的思路也很清晰。整个系统的搭建没有什么奇技淫巧，而是针对每个问题都给出了常规的工程解决方案。

在上节课里，我们看到的是尽量考虑**容错**、采用**分层**来提升性能，以及针对具体问题进行**优化**的方案。而这节课里，我们看到的则是直接在系统**内置**用户协同工具、把常用的依赖关系**抽象**到系统里，以及**监测**运行数据，以确保理解用户是如何使用系统的方案。

两个方案都可谓是“善战者无赫赫之功”，每一个动作和方法都不是独属于大数据系统的，而是在工程和应用开发中，常见的解决问题的方案。而且，这些方案中，大量考虑的不只是系统本身有多复杂和前沿，而是**如何让大量的用户能够真正把系统用起来和用好**。

其实，回到我们日常的工程开发中，理解各种系统复杂艰深的底层原理，自然非常重要。但是如何在内部真正把系统用起来和用好，也是一个重大的挑战。有着精英技术团队的公司我见到过不少，但也见过很多这样的团队效率和产出并不高。有时候，我们会沉醉于有趣先进的技术，却忘了这些系统需要被好好地使用起来，才能发挥出真正的价值。

推荐阅读和思考题

在这个课程的最后一讲，我就不再给你推荐一篇非常具体的阅读材料了。相信经过这一整门的课程学习，你应该也已经学会了如何自己寻找论文和资料，通过这些资料自发学习和成长了。

那在今天这节课的最后，就请你为参与这个课程的所有同学，推荐一篇你觉得有价值的阅读材料。它可以是一篇论文，也可以是一个视频，只要是你自己从中学到有价值的大数据知识就好。

大数据这个领域仍处在快速发展的过程中，也希望你在学完了这个课程之后，能够坚持不断学习，追踪最新的论文运用到自己的工作中去。

好了，我们结束语再见吧。

上一篇 39 | 十年一梦，一起来看Facebook的数据仓库变迁（一）

下一篇 结束语 | 长风破浪会有时，直挂云帆济沧海

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。