下载APP

39 | 十年一梦，一起来看Facebook的数据仓库变迁（一）

2022-01-26 徐文浩

《大数据经典论文解读》 课程介绍 >

讲述：徐文浩

时长 14:45 大小 13.52M

你好，我是徐文浩。

前面两节课，我们是从方法论和具体的实践这两方面，一起了解了 Twitter 的大数据系统是怎么样的。而在过去的整个课程里，我们也看过大量的来自 Google 的论文。

发表了大量论文的 Google，是开创整个大数据时代的引领者，我们有太多可以学习他们的地方，却很难去模仿他们。那么，团队规模比起来小很多的 Twitter，则是一个我们可以直接“抄袭”的对象，我们不仅可以借鉴他们的直接经验，在选择开发什么系统上，也可以模仿他们的整体思路。

而有一家公司，则介于他们两者之间，那就是今天我们要介绍的 **Facebook**。比起 Twitter，随着逐渐发展壮大的过程，Facebook 为开源社区提供了更多广泛使用的开源项

领资料

目。比起 Google，Facebook 在大数据系统的开创性工作上却做得不多。

而如果你所在的团队，正好要从各种实践方法的优化，进一步提升到**通过开发新系统来让大数据体系更加易用**，那么从 Facebook 的数据基建的发展历程中，你可以学到很多东西。

所以今天，就请你和我一起来读一读发表在 2010 年的《Data Warehousing and Analytics Infrastructure at Facebook》这篇论文。在这篇论文里，Facebook 还处于自己发展的早期阶段，而在下节课里，我们还会看到 Facebook 的数据基建是如何进一步进化的。

那么，通过这节课，我希望你能够学到一点：大数据系统在真实环境里应该如何去架构、部署。

多个“集群”的职责划分

前面我们读了那么多论文，这些论文也大都变成了一个个的开源系统。看起来，似乎我们只要把 Hadoop 和 Hive 简单地部署一下，我们就有一个大数据系统了。不过，当我们真的开始用上了这些系统，就会发现事情并没有那么简单。

Facebook 面临的**第一个问题**，就是在大数据系统上跑着各种要求完全不同的程序。有些是有严格 SLA 的，比如每天晚上计算的广告计费报表，第二天早上需要呈现给客户一个准确的数据。有些没有 SLA，但我们总是希望它们能够跑得快一点，比如数据分析师临时跑的分析脚本。

但是，在一个 Hadoop 集群上，这些任务是在一个资源池里运行的。如果有数据分析师，不小心写了一个复杂的 Hive SQL，需要占用大量的资源，那么它就会影响甚至阻塞到我们有 SLA 需求的生产任务。而且，不仅仅是 Hadoop 上运行的分析任务会占用资源，本身前端的业务应用的日志，想要落地到 Hadoop 上，一样会占用 Hadoop 集群的磁盘 IO。

拆分集群来确保各集群的 SLA

针对这个问题，Facebook 的做法比较简单直接，那就是拆分集群。在 Facebook 的整个生产环境中，一共有三个 Hadoop 集群。

第一个集群，Facebook 称之为 **Scribeh 集群**。这个集群，只负责接收日志。Web 应用服务器的日志，通过 Scribe 这个日志收集系统，直接落地到这个集群的 HDFS 上。并且，对应的 Scribe 服务也是直接运行在这个集群的服务器上的。

要注意，这里的**日志都是未压缩的**。你可能会问，为什么不压缩一下呢？毕竟论文里一开始的数据中，就能看到日志压缩之后，存储空间只有原来的 1/6。

这是因为我们需要确保日志落地是**低延时**的，这个延时并不是压缩算法本身会占用多少时间。而是，启用压缩的话，我们需要将一个特定的 Buffer 填充满，才能在压缩后把数据从 Web Server 发送到 Scribe 里。

但是，有些 Web Server 和有些类型的日志数量很少，并不会在几秒钟甚至几分钟内填充满整个 Buffer 区。这样，一旦启用压缩，这些日志会等待很长时间，才能最终落地到 Scribe 的集群上。

当日志落地到了 Scribe 集群上之后，Facebook 会通过一个程序，定期地往生产的 **Hive-Hadoop 集群**复制数据。这个定期的时间间隔通常在 5~15 分钟左右，也就是说在数据层面，Facebook 实际从日志产生，到日志可以在一个生产环境被分析，通常有 5~15 分钟的延时。这个延时，其实也是 Storm、Flink 这样的系统随后出现在历史舞台的原因。

对于生产的 Hive-Hadoop 集群来说，除了 Web Server 的日志数据之外，它还需要**对应的元数据**。没有这些元数据，我们其实没法儿分析这些日志数据。比如，在广告投放日志里，固然有是哪一个客户，投放哪一条广告。但是，通常它并不会有用户设置的广告是投放给哪些用户群，以及投放到哪些地区的信息。

这些广告投放设置的数据，往往是在一个 MySQL 数据库里。所以，生产的 Hive-Hadoop 集群，需要定期每天去同步读取 MySQL 里面的数据，然后转换成一系列的 Hive 表。如果你熟悉数据仓库的话，你会发现原始日志，就成了数据仓库的“事实表”，而 MySQL 同步过来的数据，就成了数据仓库的“维度表”。

直接使用历史数据作为容错手段

为了避免给生产环境的 MySQL 带来太大的负担，生产的 Hive-Hadoop 集群会读取这些数据库的一个单独的副本。

不过，从 MySQL 复制数据，到生产的 Hive-Hadoop 集群可不是一个简单的事儿，这是因为在 Facebook，需要复制的 MySQL 数据库有数千个之多。而在这么多的 MySQL 数据库里，每天总有几个是会坏掉的。如果你有三千个数据库，你的系统可用性有 99.9%，那你每天平均也有 3 台数据库是无法访问的。

Facebook 的解决方案非常简单粗暴，也就是如果一个数据库不可用，那么就用这个数据库前一天同步过来的数据。考虑到同步抓取的都是“维度表”，那么最坏情况也无非是在分析数据的时候，会缺失很少量的维度信息，倒也不是不能接受。

而在实际进行数据分析的集群里，Facebook 把集群拆成了两个，一个是我们刚才说过的生产的 Hive-Hadoop 集群，另一个则是 **Adhoc 的 Hive-Hadoop 的集群**。有严格完成时间要求的高优先级任务，都会在生产集群上运行，而其他的任务都在 Adhoc 的集群上运行。Adhoc 集群上的任务，既有生产型每天定时运行的任务，也有数据分析师临时运行的真正“Adhoc”的任务。不过，这里的生产任务的完成时间没有生产集群要求那么高。

在 Adhoc 集群上的生产任务，虽然没有绝对严格的完成时间要求，但是它们仍然需要在特定时间之前跑完。所以，在 Adhoc 集群里，我们仍然面临 SLA 要求不同的任务之间争抢资源的问题。

对于这个问题，Facebook 的解决办法是改造了 Hadoop 的 FairScheduler，让所有的硬件服务器的资源**分成两个不同的资源池**。虽然两边的资源可以共享，但是可以始终保证生产类型的任务至少占有一定的资源，即使 Adhoc 的 SQL 写得太糟，也不会占满整个集群的资源，导致生产类型的任务跑不完。

通过监控进程和 Hook 来同步数据和元数据

Facebook 有两个 Hadoop 的集群，而且在 Adhoc 的集群里，我们一样需要生产集群跑出来的报表数据。因为大量的分析工作，并不需要从原始日志开始进行，而是可以基于生产系统已经生成的报表来进行。所以，Facebook 通过一个**进程**去监控生产的 Hive-Hadoop 集群，把对应的数据表都同步到 Adhoc 的集群里来。

同时，当用户在生产集群上运行命令，去修改 Hive 的元数据的时候，Facebook 通过 Hive 的 pre-execution 的 Hook，一样会同步在 Adhoc 集群上执行相同的命令，确保两边的数据和元数据都是同步的。

合并小文件来减轻 NameNode 压力

在有了这些动作之后，我们基本可以保障整个大数据的数据仓库，能够做到“分工明确、各司其职”了。不过，随着数据的快速增长，Facebook**还会面临一个问题**，那就是似乎无论加上多少服务器都不够用。

我们在论文里给出的数据可以看到，Facebook 在 2010 年的时候，就已经每天要新增 60TB 的数据了，并且每天线上有 1 万个任务在运行。随着数据的不断增长，存储本身对于他们的 Hadoop 集群来说，已经是一个巨大的压力了。

这个压力体现在两个方面。

第一个是**对于 NameNode 的压力**。不知道你还记得GFS的论文吗？Hadoop 的 NameNode 也好，GFS 的 Master 也好，它们本身是要在内存里，存储所有文件和对应所在的 Block 的元数据的。也就是说，随着文件数量的不断增长，NameNode 一定会不够用。

而且，因为线上有大量的 Adhoc 的任务，而很多这些任务，就是通过一个 Map 函数快速过滤出自己想要看到的数据。这些任务的输出结果很小，但是文件数却和原始表或者原始日志的文件数一样多。面对这样大量的小文件，我们的 NameNode 被占用了大量的内存空间，压力也就越来越大。

针对这个问题，Facebook 的解决办法，是在 Hive 里自动加上了一个**合并文件**的步骤，让任务的完成时间增加，但是文件数量减少。

不同集群和策略来存储冷热数据

回过头，其实不光是文件数越来越多，随着日志的不断增长，本身**集群需要的硬盘空间也会越来越大**，而且里面大量的都是历史的日志和数据表。这些数据虽然并不常用，但是又时不时会被分析到。在各种生产系统的报表里，我们用不上这些数据。但是，每当我们有一个新想法，想要通过历史数据去验证分析的时候，我们又会把这些数据拿出来跑一些分析程序。

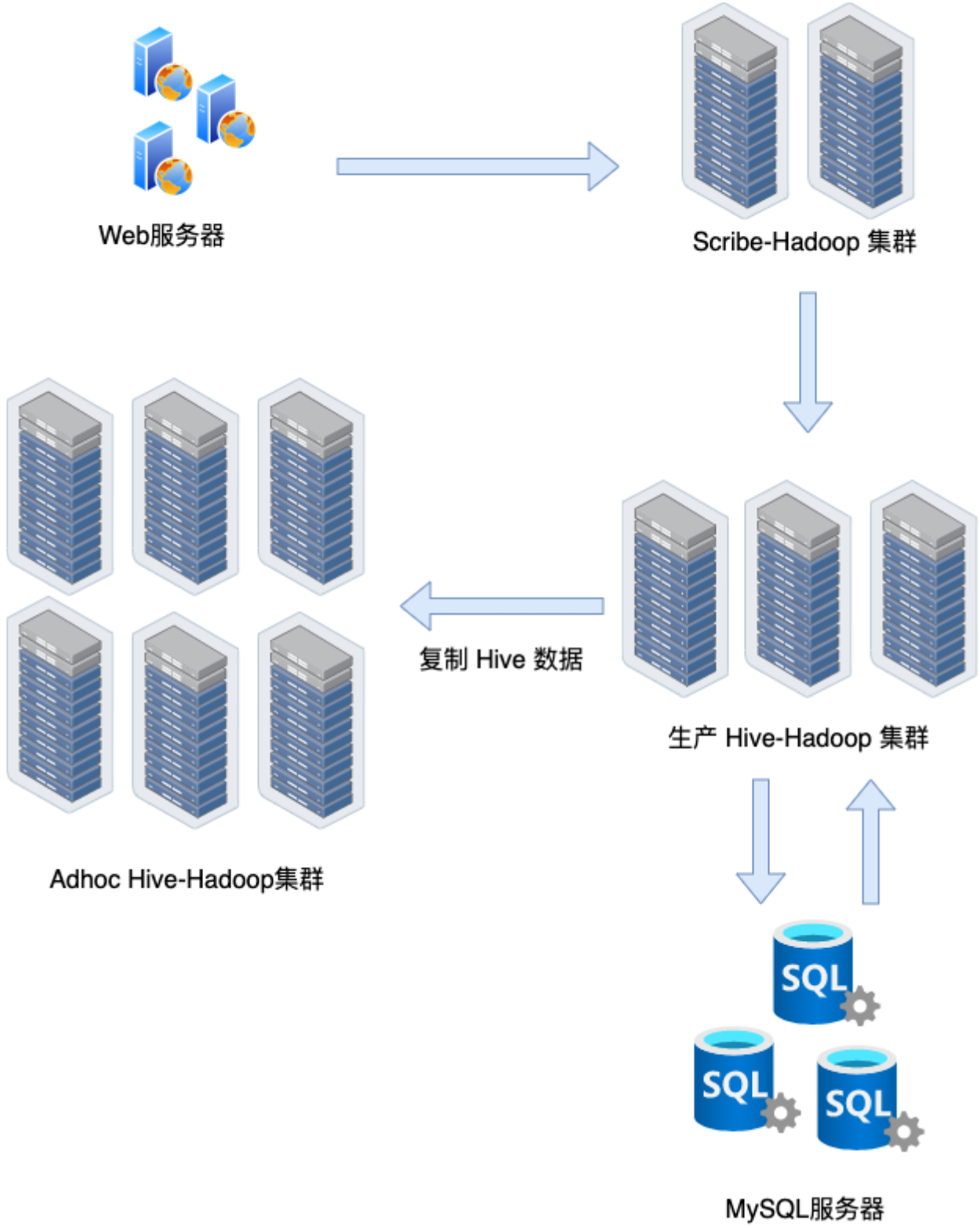
那么 Facebook 的解决方案，就是**对冷热数据进行分离**。生产的 Hadoop 集群，只需要维护最新的一个月的数据，而 Adhoc 的集群则会保留更多的历史数据。并且，无论是哪一边

的数据，都和 Scribe 的集群不一样，它们是进行过压缩的。所以，从 Scribe 集群，到生产 Hive-Hadoop 集群，再到 Adhoc 的 Hive-Hadoop 集群，Facebook 就形成了从热到冷的三层的数据体系。而这个策略，你也可以参考来用一下。

更进一步地，对于冷数据，我们并不会频繁地进行读取。那么，Hadoop 默认的三份拷贝的策略是不是可以变一下呢？

Facebook 采用的方法是，**存放两份数据以及对应的 Erasure Code**。这样，它占用的硬盘空间就是 2.2 倍，而不是 3 倍了。而且，任何一份数据损坏，我们都可以恢复过来。

可是这样就带来了一个新问题，那就是 MapReduce 运行的时候，“数据本地性”不一定能够得到保障了。不过，根据 Facebook 的经验可以看到，虽然我们不一定可以保障从同一台机器上读取数据，但是**从同机架去读取数据，并不会有太多的性能问题**。而对于最新经常会被读取到的数据，我们仍然可以复制 3 份，确保它们总是能在运行 MapReduce 任务的时候，从本地读取到。



小结

好了，这节课到这里就结束了。今天我们一起读完了论文里，关于 Facebook 搭建数据仓库的整体架构，也就是论文的 1~3 部分。

可以看到，一旦我们想要一个稳定、高效用于生产的大数据系统，我们需要考虑的具体问题就多了很多。通过多个职责分离的集群，Facebook 既能够确保数据及时落地，也避免了不同类型的任务之间抢占资源，相互影响。

Facebook 的线上集群，分成了**承载日志落地的 Scribeh 集群**，只运行有严格时间期限任务的**生产 Hive-Hadoop 集群**，以及运行没有那么严格的生产任务和临时分析程序的**Adhoc 集群**。Scribeh 集群里的数据都是没有压缩的，以确保所有的日志都能够低延时地落地，而另外两个集群都会通过 gzip 压缩数据，以尽量减少磁盘占用。

生产的 Hive-Hadoop 集群，不只要同步日志，也要从数千个 MySQL 数据库里去同步业务数据库的数据。并且最终生成的很多报表，也需要写回到 MySQL 集群里去。而为了应对必然会发生的 MySQL 服务器不可访问的问题，Facebook 采用了**直接使用历史版本数据的方案来进行容错**。

Adhoc 的 Hive-Hadoop 集群，会通过同步进程和 Hook，来同步生产 Hadoop 集群的数据。在内部，它也会把实际的资源池，分成运行生产任务和 Adhoc 任务的，以此确保生产任务始终有一定的资源运行。

在实际应用的过程中，我们的“热”数据始终会在生产集群里，而“冷数据”则只放在 Adhoc 集群里。对于冷数据中，我们也可以通过两份拷贝和 Erasure Code，来进一步减少它们占用的空间。

可以看到，Facebook 面对各种具体问题的时候，解决方案本质上都是采用 **容错 + 分层 + 优化** 的三板斧。

> 容错是为了确保在数千台服务器的集群里，总会出现软硬件的错误，不能容错的话，我们的系统就是脆弱而不可用的。
>
> 分层，则是在这么大的数据量下，总是有些数据是被频繁使用，有的数据则很少被读到。通过对数据、任务进行分层，确保大家都能够有合适的资源进行处理，并且不会相互抢占或者相互阻塞。
>
> 而优化，则是在前面两个方法解决不了问题的时候，通过压缩、合并等等的技术手段，来减少占用的资源，以降低整个集群运行的成本。

那么，如果你也要基于 Hadoop 或者是今天的 Spark 去搭建一个大型的数据仓库，Facebook 的这些经验其实是可以好好借鉴的。

最后，在下节课里，我们还会一起来看看，为了让系统易用，以及能够让系统不断向前迭代、进化，Facebook 还做了哪些工作。这些工作，既有和 Twitter 非常实战的风格相似的

地方，也有更多更系统性地考虑问题、推动系统迭代的地方。

推荐阅读

在这篇在 2010 年的论文里，Facebook 的系统还是一个完全离线定时运行的系统。而在实践应用中，我们始终希望数据系统能够是**接近于实时**的。在 2011 年，流式系统还没有正式登上历史舞台的时候，Facebook 还发表了一篇《Apache hadoop goes realtime at Facebook》。你可以从这篇论文中，找到 Facebook 在那个时代是如何让系统更加实时的，我推荐你去读一读。

除了论文之外，Facebook 其实在很多会议上，也做了很多针对自己数据基础架构的讲座。我推荐你去看看 Facebook 在 2015 年，F8 开发者大会上给出的数据基础设施的讲座。你可以和这篇论文对照一下，看看在 5 年之后，Facebook 的数据基建又有了哪些进化。我把对应视频讲座的链接放在这里了。

思考题

我们看到，在 Facebook 的大数据系统里，数据是进行了分层存放的。那么，在你接触使用的大数据系统中，也是这样做的吗？无论是否是这样做的，你能讲讲你们系统分层或者不分层的考虑是怎么样的吗？

欢迎你在留言区分享出来，也欢迎你把今天的内容分享给更多的朋友。

分享给需要的人，Ta购买本课程，你将得 20 元

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 38 | 当数据遇上AI，Twitter的数据挖掘实战（二）

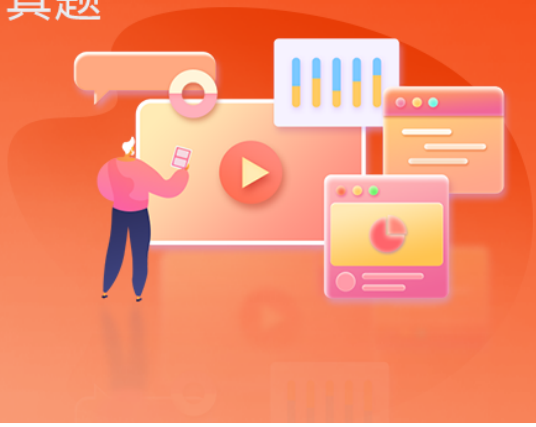
下一篇 40 | 十年一梦，一起来看Facebook的数据仓库变迁（二）

更多学习推荐

190 道大数据高频面试真题

涵盖 11 个核心技术栈 + 4 套大厂真题

免费领取



精选留言 (2)

写留言

vkingnew

2022-01-26

有机会请详细的讲解下Erasure code的原理和应用？有些数据不用是否可以销毁？应该有销毁的策略

共 1 条评论 >

1

尚春

2022-02-05

我们厂目前一些分层的思路：
- 数据湖：主要存放一些对于读取性能要求不那么高的数据（Delta 格式，全量长周期数据），以及 ML 中用到的半结构化数据
- 数仓：主要存放一些需要快速读取的数据用于分析（AWS Redshift，只有近期详细数据和周期性汇总数据）

展开