大规模 Hadoop 升级在 Pinterest 的实践

Monarch 是 Pinterest 的批处理平台，由30多个 Hadoop YARN 集群组成，其中17k+节点完全建立在 AWS EC2 之上。2021年初，Monarch 还在使用五年前的 Hadoop 2.7.1。由于同步社区分支（特性和bug修复）的复杂性不断增加，我们决定是时候进行版本升级了。我们最终选择了Hadoop 2.10.0，这是当时 Hadoop 2 的最新版本。

本文分享 Pinterest 将 Monarch 升级到 Hadoop 2.10.0 的经验。为了简单起见，我们将 Hadoop 2.10.0 简称为 Hadoop 2.10；将 Hadoop 2.7.1 简称为 Hadoop 2.7。

挑战

自 Pinterest 的批处理平台开始（大约 2016 年）以来，我们一直在使用 Hadoop 2.7。随着时间的推移，我们的平台处理的工作负载不断增长和发展。为了满足这些需求，我们进行了数百次内部更改。这些内部补丁中的大多数都是 Pinterest 特有的，需要大量时间投入才能将它们移植到 Hadoop 2.10。

大多数最关键的批处理工作负载是在 Monarch 上运行，因此我们的首要任务是以不会对这些工作负载造成集群停机或性能/SLA 影响的方式执行升级。

升级策略

由于许多用户定义的应用程序与 Hadoop 2.7 紧密耦合，我们决定将升级过程分为两个独立的阶段。第一阶段是平台本身从 Hadoop 2.7 升级到 Hadoop 2.10，第二阶段是用户自定义应用升级到 2.10。

在升级的第一阶段，我们允许用户的作业继续使用 Hadoop 2.7 的依赖项，同时我们专注于平台升级。这增加了额外的开销，因为我们需要使 Hadoop 2.7 作业与 Hadoop 2.10 平台兼容，但它会让我们有更多的时间来处理第二阶段。

由于平台和用户应用程序的规模，上述两个阶段都需要逐步完成：

- 我们需要一个一个地升级 Monarch 里面的集群；
- 我们需要将用户应用程序批量升级到与 2.10 绑定，而不是 2.7。

当时，我们没有一个灵活的构建管道来允许我们构建两个不同版本的作业，这些作业具有单独的 hadoop 依赖项。 同样，我们要求所有用户（公司中的其他工程师）单独验证从 2.7 到 2.10 的 10,000 多个单独的工作迁移是不合理的。
为了支持上述增量升级，我们需要在迁移之前运行许多验证，以确保使用 Hadoop 2.7 构建的绝大多数应用程序将继续在 2.10 集群中工作。

我们为升级过程制定的步骤如下:

- Hadoop 2.10 发布准备：将 Hadoop 2.7 内部分支上的所有补丁移植到 Apache Hadoop 2.10 上；
- 一个一个的升级 Monarch 集群到 Hadoop 2.10 ；
- 批量升级用户应用程序以使用 Hadoop 2.10。

Hadoop 2.10 发布准备

Pinterest 使用的 Hadoop 2.7 版本包括在开源 Hadoop 2.7
之上进行的许多内部更改，这些更改需要移植到 Hadoop 2.10。但是，Hadoop 2.7 和 Hadoop 2.10 之间发生了重大变化。 因此，将 Pinterest Hadoop 2.7 的更改应用到社区的 Hadoop 2.10 上并非易事。

下面是一些我们在 Hadoop 2.7 上做的内部补丁，然后移植到 Hadoop 2.10 的例子：

- Monarch 建立在 EC2 之上，并使用 S3 作为持久存储，Task 的输入和输出通常在 S3 上。Pinterest 实现了 DirectOutputFileCommitter 以使 Task
  能够直接将结果写入目标位置，以避免在 S3 中复制结果文件的开销；
- 添加 application master 和 history server 的 endpoint
  以获取给定作业的所有任务的特定计数器的值；
- 在为 container log 提供服务时实现了范围查找，它允许获取指定 container log 的一部分；
- 为日志聚合添加 Node-Id 分区，使得集群不同节点的日志可以写入不同的 S3 分区，避免达到 S3 访问速率限制。
- 新创建的 namenode 可能有不同的 IP 地址，我们实现了一个功能，以解决NN RPC套接字地址故障转移。
- 如果分配的 mappers 数量与总 mappers 的比率超过配置的阈值，则禁用 preempting reducers。
- 将磁盘使用监控线程添加到
  AM，这样如果磁盘使用超过配置的限制，应用程序将被终止。

升级 Monarch 中的集群到 Hadoop 2.10

集群升级方法探索

我们评估了将 Monarch 集群升级到 Hadoop 2.10
的多种方法。每种方法都有其优缺点，我们将在下面概述。

方案一：使用 CCR（cross-cluster routing，跨集群路由）

正如在《Efficient Resource Management》
一文提到的，我们开发了跨集群路由（CCR）来平衡不同集群之间的工作负载。
为了尽量减少对现有 2.7 集群的影响，一种选择是构建新的 Hadoop 2.10 集群，并逐步将工作负载转移到新的集群。如果出现任何问题，我们可以将工作负载路由回原来的集群，修复问题，然后再次路由回2.10集群。

我们开始使用这种方法，并在一些小的生产和开发集群上进行评估。没有什么大问题，但我们发现了一些缺点:

- 我们必须为每个集群迁移构建一个新的并行集群，对于大型 YARN 集群（多达数千个节点），这会变得很昂贵
- 需要批量迁移工作负载，这非常耗时。因为 Monarch 是一个非常大的平台，这个升级过程可能需要很长时间才能完成。

方案二：滚动升级（Rolling Upgrade）

理论上，我们可以尝试滚动升级 Worker 节点，但滚动升级可能会影响集群上的所有工作负载，如果我们遇到任何问题，回滚将是昂贵的。

方案三：原地升级（In-place Upgrade）

我们利用类似的方法将集群从一个实例类型升级到另一个实例类型，我们：

- 将几个新实例类型的 canary 主机作为新的自动缩放组（canary ASG）节点插入集群
- 评估相对于 base ASG（现有实例类型）的 canary ASG
- 扩大 canary ASG
- 缩小 base ASG

一般来说，这对于没有服务级别更改（service level change）的小型基础设施非常有效。作为探索，我们想看看 Hadoop 2.10 的升级是否也能做到这一点。我们必须做出的一个关键假设是 Hadoop 2.7 和 2.10 组件之间的通信是兼容的。这种方法的步骤是:

- 在 Hadoop 2.7 集群中添加 Hadoop 2.10 worker 节点（运行 HDFS datanode 和 YARN NodeManager）；
- 发现并解决出现的问题；
- 增加 Hadoop 2.10 worker 节点的数量，减少 Hadoop 2.7 worker 节点的数量，直到2.7节点完全替换为2.10节点
- 升级所有管理节点（namenode, JournalNodes, resourcemanager, History Servers等）。这个过程的工作原理类似于用 Hadoop 2.10 节点替换 worker 节点。

在将这种有风险的方法应用到生产集群之前，我们对生产环境的 Monarch 集群进行了广泛的评估。这是一个无缝的升级体验，除了一些小问题，我们将在后面描述。

最终升级方案

如前所述，业务作业最初是用 Hadoop 2.7 依赖项构建的。这意味着它们可以将 Hadoop 2.7 jar 文件携带到分布式缓存中。然后在运行时，我们将用户类路径放在集群中存在的库路径之前。这可能会导致 Hadoop 2.10 节点的依赖问题，因为 Hadoop 2.7 和 2.10 可能依赖不同版本的第三方 jar。

在使用方法一对一些小集群进行升级后，我们认为这种方法将花费太长时间来完成所有 Monarch

集群的升级。此外，考虑到我们最大的 Monarch
集群的规模（多达3k个节点），我们无法在这么短的时间内获得足够的 EC2 实例来替换这些集群。我们评估了优点和缺点，并决定采用方法三，因为我们可以大大加快升级过程，并且可以快速解决大多数依赖问题。如果我们不能快速解决某些作业的问题，我们可以使用 CCR
将作业路由到另一个 Hadoop 2.7 集群，然后花时间修复问题。

遇到的问题及解决方案

在我们最终确定了方法三之后，我们的主要关注点就变成了识别任何问题并尽快解决它们。从广义上讲，我们遇到了三类问题：由于 Hadoop 2.7 和 Hadoop 2.10
之间的不兼容导致的服务级别问题、用户定义的应用程序中的依赖性问题以及其他各种问题。

不兼容的行为问题

- 重启 Hadoop 2.10 NM 会导致容器被杀死。我们发现 Hadoop 2.10 引入了一个默认值为 FALSE 的新配置 yarn.nodemanager.recovery.supervised。为了防止容器在重新启动 NMs 时被杀死，我们需要将其设置为TRUE。当启用此配置时，运行中的 NodeManager 不会尝试清理容器，因为它会假设立即重启并恢复容器。
- 当 AM 在 2.10 节点上调度时，作业可能会卡死：MAPREDUCE-6515 中添加的 Application Priority 假设该字段总是在 PB（protobuf）响应中设置。在多版本的集群（2.7.1 ResourceManager + 2.10 worker）中则不是这样，因为 RM 返回的 PB 响应将不包含 appPriority 字段。我们检查这个字段是否在 protobuf 中，如果不在，则忽略更新 applicationPriority。
- HADOOP-13680 使 fs.s3a.readahead.range 从 Hadoop 2.8 开始使用 getLongBytes，并支持“32M”格式的值（内存后缀 K、M、G、T、P）。 但是，Hadoop 2.7 代码无法处理这种格式。这会破坏混合 Hadoop 版本集群中的作业。我们为 Hadoop 2.7 添加了一个修复程序，以使其与 Hadoop 2.10 行为兼容。
- Hadoop 2.10 不小心在 io.serialization 配置的多个值之间引入了空格，这导致了 ClassNotFound 错误。 我们进行了修复以删除配置值中的空格。

依赖问题

当我们执行 Hadoop 2.7 到 2.10 的就地升级时，我们面临的大多数依赖问题是由于 Hadoop 服务和用户应用程序之间共享的不同版本的依赖关系造成的。 解决方案是修改用户的作业以与 Hadoop 平台依赖项兼容，或者在作业或 Hadoop 平台分发版中设置版本号。以下是一些例子：

- Hadoop 2.7 jars 被放入分布式缓存并导致 Hadoop 2.10 节点上的依赖问题。我们在 Hadoop 2.7 版本中实现了一个解决方案，以防止将这些 jars 添加到分布式缓存中，以便所有主机都使用已部署到主机的 Hadoop jars。
- woodstox-core 包：Hadoop-2.10.0 依赖于 woodstox-core-5.0.3.jar，而一些应用程序依赖于 wstx-asl-3.2.7.jar 的模块。woodstox-core-5.0.3.jar 和 wstx-asl-3.2.7.jar 之间的不兼容导致了作业失败。我们的解决方案是在 Hadoop 2.10 中遮蔽 woodstox-core-5.0.3.jar。
- 我们有一些基于 Hadoop 2.7 实现的内部库或类。它们不能在 Hadoop 2.10 上运行。例如，我们有一个名为 S3DoubleWrite 的类，它同时将输出写到 s3 的两个位置

。它的开发是为了帮助我们在3个桶之间迁移日志。因为我们不再需要那个类了，所以直接删除它即可。
- 一些 Hadoop 2.7 库被打包到用户的 bazel jar 中，在运行时导致一些依赖问题。我们采取的解决方案是将用户应用程序与 Hadoop jar 解耦，更多的细节可以在后面的相关章节中找到。

各种各样的其他问题

- 我们在开发集群上执行的验证之一是确保在升级过程的中可以回滚。当我们试图回滚 NameNode 到 Hadoop 2.7 时，出现了一个问题。我们发现 NameNode 没有收到来自升级的 datanode 的块报告。我们确定的解决方法是手动触发块报告。我们后来发现了潜在的问题 HDFS-12749 (DN 可能不会在 NN 重启后向 NN 发送阻塞报告)，并将其移植到我们内部分支。
- 当 Hadoop streaming 作业与 Hadoop 2.7 jar 捆绑部署到 Hadoop 2.10 节点时，预期的 2.7 jar 不可用。这是因为我们使用集群提供的 jar 来满足大多数用户作业的依赖关系，从而减少作业的大小。然而，所有的 Hadoop 依赖都在 jar 名称中编码了版本。解决方案是让 Hadoop streaming 作业包的 Hadoop jar 不带版本字符串，这样提供的 Hadoop 依赖项在运行时总是在类路径中，而不管它运行在 Hadoop 2.7 或 2.10 的节点上。

将用户程序升级到 Hadoop 2.10

为了将用户应用程序升级到 Hadoop 2.10，我们需要确保在编译时和运行时都使用 Hadoop 2.10。第一步是确保 Hadoop 2.7 jar 不是随用户 jar 一起提供的，以便在运行时使用部署到集群中的 Hadoop jar（2.7 版本的节点有 2.7 版本的 jar, 2.10 版本的节点有 2.10 版本的jar）。然后我们将用户应用程序构建环境改为使用 Hadoop 2.10 而不是2.7。

将用户应用程序与 Hadoop jar 解耦

在 Pinterest，大多数数据管道都使用 Bazel 构建的 fat jars。这些 jar 包含了升级前的所有依赖关系，包括 Hadoop 2.7 客户端库。我们总是优先使用那些 fat jar 中的类而不是本地环境中的类，这意味着在使用 Hadoop 2.10 的集群上运行这些 fat jar 时，我们仍将使用 Hadoop 2.7 类。

为了解决这个问题（在2.10集群中使用 2.7 jar），我们决定将用户的 Bazel jar 从 Hadoop 库中解耦；也就是说，我们不再将 Hadoop jar 放在 fat user Bazel jar 中，已经部署到集群节点的 Hadoop jar 将在运行时使用。

Bazel java_binary 规则有一个名为 deploy_env 的参数，它的值是表示此二进制的部署环境的其他 java_binary 目标的列表。我们设置这个属性是为了从用户 jar 中排除所有 Hadoop 依赖及其子依赖。这里的挑战在于，许多用户应用程序对 Hadoop 所依赖的库有共同的依赖关系。这些常见的库很难识别，因为它们没有明确指定，因为它们已经作为 NodeManager

部署的一部分在 Hadoop worker
本身上提供了。在测试期间，我们花了很多精力来识别这些类型的情况，并修改用户的 bazel 规则，以显式地添加那些隐藏的依赖项。

将 Hadoop bazel targets 从 2.7 升级到 2.10

在将用户应用程序与 Hadoop Jars 解耦后，我们需要将 Hadoop bazel targets 从 2.7 升级到 2.10，以便我们可以确保构建和运行时环境中使用的 Hadoop 版本是一致的。
在这个过程中，Hadoop 2.7 和 Hadoop 2.10 之间又出现了一些依赖冲突。
我们通过构建测试确定了这些依赖项，并相应地将它们升级到正确的版本。

总结

将 17k 多个节点从一个 Hadoop 版本升级到另一个 Hadoop 版本，同时又不会对应用程序造成重大破坏，这是一个挑战。我们以高质量、合理的速度和合理的成本效益做到了这一点。我们希望以上分享的经验对社区有益。

本文翻译自：Large Scale Hadoop Upgrade At Pinterest