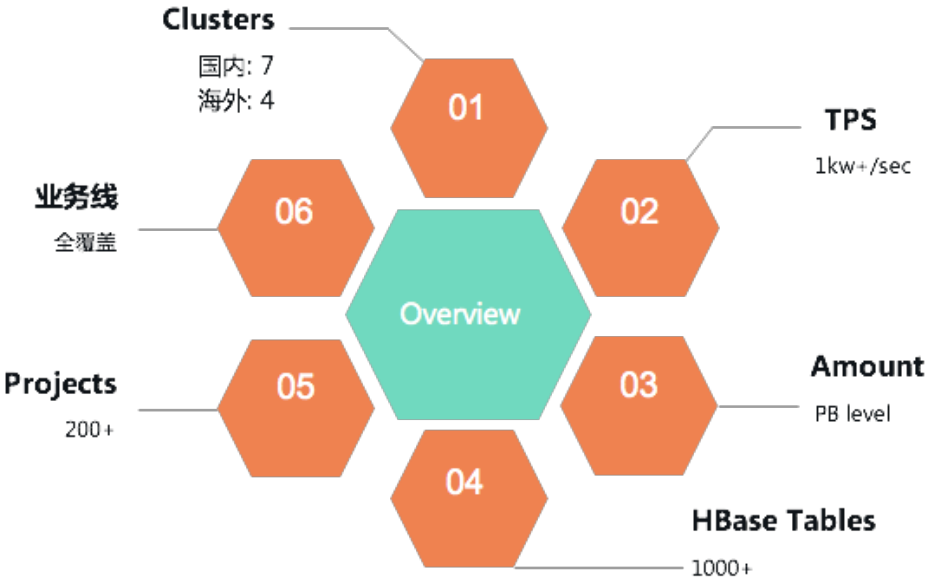
滴滴 HBase 大版本滚动升级之旅

滴滴HBase团队日前完成了0.98版本 -> 1.4.8版本滚动升级，用户无感知。新版本为我们带来了丰富的新特性，在性能、稳定性与易用性方便也均有很大提升。我们将整个升级过程中面临的挑战、进行的思考以及解决的问题总结成文，希望对大家有所帮助。

背景

目前HBase服务在我司共有国内、海外共计11个集群，总吞吐超过1kw+/s，服务着地图、普惠、车服、引擎、金融等几乎全部部门与业务线。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

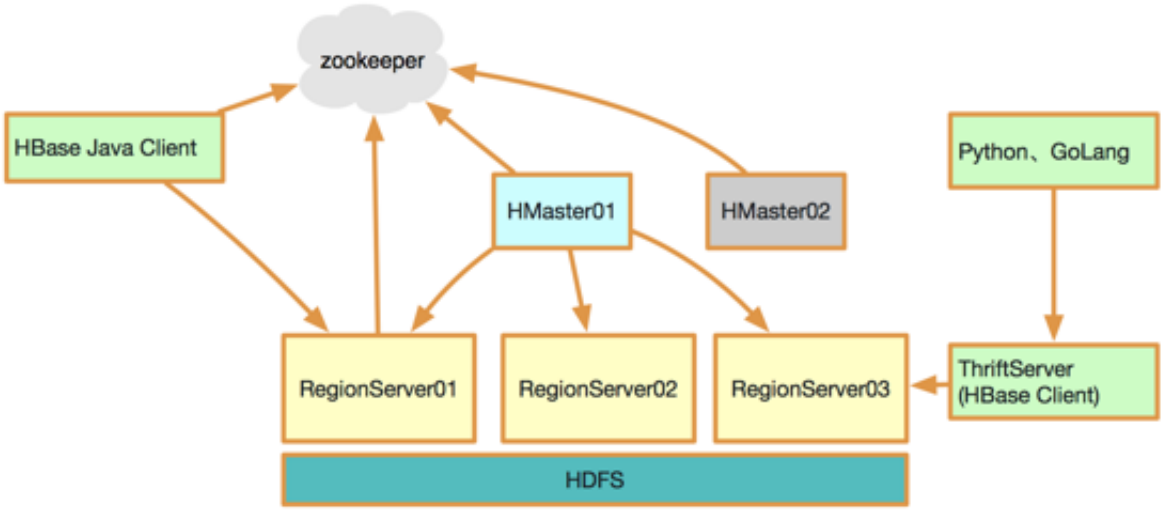
然而有一个问题持续困扰着我们：版本较社区落后较多——HBase线上集群使用0.98版本，而社区目前最新的release版本为2.3。这为我们的工作带来了很多额外的掣肘与负担，主要包括以下几点：

- 新特性引入成本极高；
  0.98版本可以算是HBase第一个稳定版本，但过于老旧，社区已经不再维护。想要backport新特性难度越来越大。
- 自研patch维护成本较高；
  我们基于0.98版本有数十个大大小小的自研patch，涵盖了从label分组、ACL鉴权等大的feature到监控体系建设、审计日志优化等Improvement以及各种bug fix。这些patch或是新版本中已支持但和我们实现有差异，或是由于版本差异过大无法合入社区，而且随着时间线的拉长，这种问题只会进一步恶化。
- 上层组件对于HBase存在一定需求；

> 得益于活跃的HBase生态圈，目前我们的用户使用形态也比较丰富，OLAP（Kylin）、时空索引（GeoMesa）、时序（OpenTSDB）、图（JanusGraph）等等场景不一而足。然而这些上层引擎无一例外，最新版本没有任何一款是依赖0.98版本HBase的。

因此对于HBase团队而言，升级迫在眉睫刻不容缓。

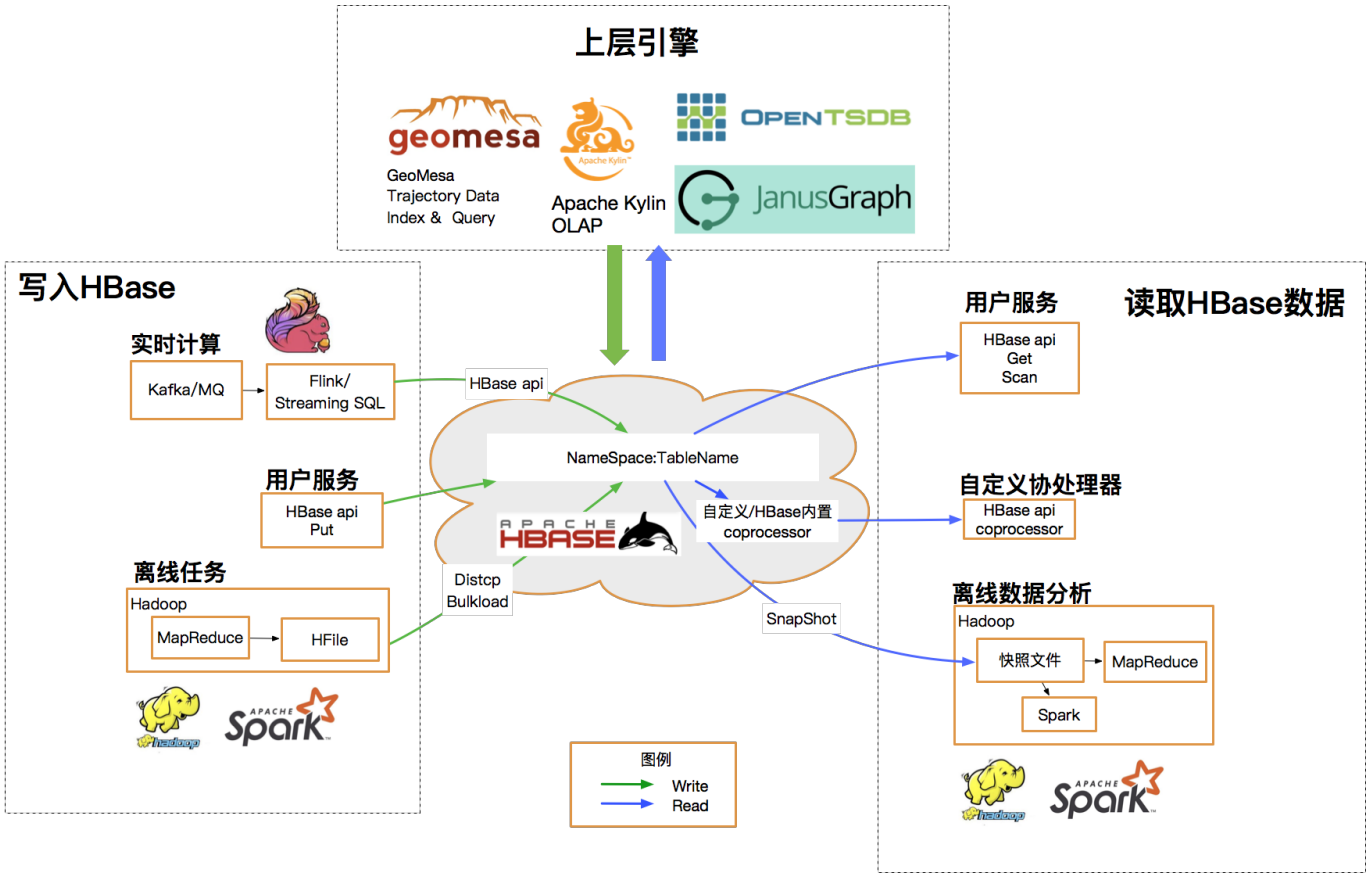
挑战



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

首先简单介绍一下HBase的架构。HBase是一个典型的主从结构——主备Master用于管理集群；RegionServer用于响应处理用户读写请求；使用ZooKeeper保障集群内的一致性；节点间通过RPC通信；底层数据文件HFile存储于HDFS中。

此外HBase具有相当丰富的上下游生态，从以StreamingSQL为代表的实时任务到Spark、MR等批处理任务，通过下图可以得窥一二：



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

基于以上对集群架构和上下游生态的梳理，可以明确升级过程中我们主要面临的挑战有：

- RPC接口兼容性问题；
  升级不可能一蹴而就，因此我们需要确保所有RPC通信接口新旧版本完美兼容。
- HFile兼容性问题；
  不同版本中底层文件的数据格式有差异。1.4.8版本默认使用HFile v3，幸运的是0.98版本虽然使用HFile v2，但已经可以兼容v3。这样我们就不需要再额外backport相关patch到0.98版本。此外也是基于这方面的因素，官方文档中说明0.98版本想要升级到2.x版本，必须以1.x版本作为过渡——这也是我们本次升级选择1.4.8版本的原因。
- 自有patch兼容性问题；
  如上文所述，我们需要全量梳理自研的数十个patch——高版本是否有替代方案、替代方案是否兼容、是否需要移植到新版本、移植后的功能/性能/兼容性测试等。
- 上下游兼容性问题；
  这一点大家看上图就好不需再赘述，每一种引擎的应用都需确保完全兼容。
- 可能引入的新问题；
  HBase的社区release版本迄今仍然非常活跃，但同时这也意味着可能存在很多潜藏的问题（事实上我们在升级过程中也遇到了，解决了并反哺给社区）——这一点其实没有什么太好的办法，只能要求我们随时紧密跟进社区，洞察新进展，即时发现问题修复问题。

基于以上这些挑战点，其实不难得出一个结论：我们需要设计并实施大量的前置准备工作以保证

升级过程的可靠性，但这并不是什么坏消息，因为只要我们的准备工作足够细致完善，顺利升级的把握和信心也就越强——这个思路在我们今后的工作中也同样适用。

下面简单列举了我们完成的准备工作：

- Release Note review
- 自研patch移植与测试
- 基础功能测试、性能测试
- 高阶功能测试（Bulkload，Snapshot，Replication，Coprocessor等）
- 社区后续小版本patch梳理跟进（截止目前实际合入的有100余个）
- 跨版本兼容性测试、RPC接口兼容性梳理
- 全量测试集制定与实施，涵盖HBase，Phoenix，GeoMesa，OpenTSDB，Janusgraph等所有使用场景
- 软件包及配置文件准备

升级方案

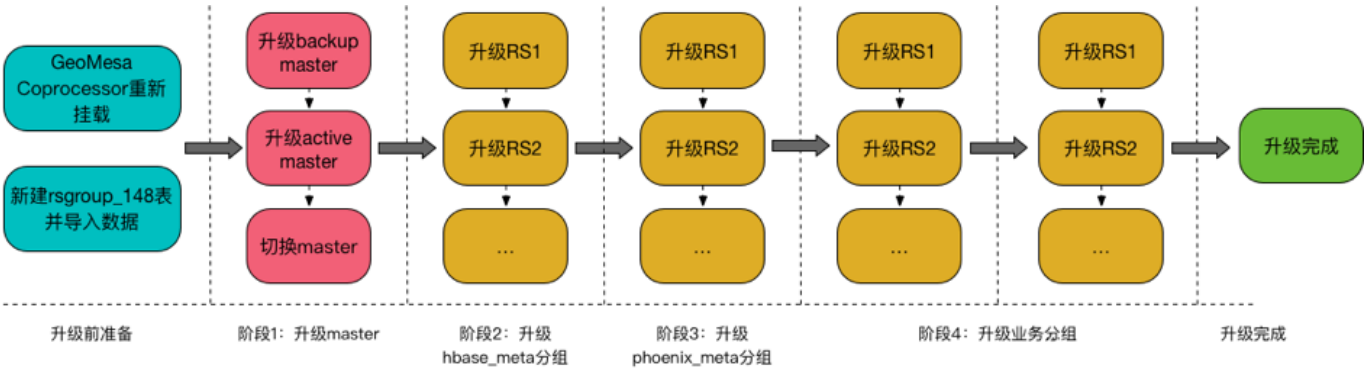
升级方案主要有两种：新建集群+数据迁移 和 滚动升级。

	优点	缺点
新集群+数据迁移	双集群可快速降级，风险较小	需用户配合切换，体验较差
		升级周期较长，预计半年以上
		操作复杂，成本高
滚动升级	用户无感知	发现故障需要即时回滚，挑战相对较高
	升级周期短	
	操作简单，成本低	

实际上滚动升级的方案一定是最优选，主要是出于对“release版本仍然不够稳定”的担忧，我们一度有所犹豫。但最终基于“充分的准备与测试”带给我们的信心，最终我们仍然选择了滚动升级。

简单说明滚动升级的大致步骤：

- 解决兼容性问题，主要体现在新建rsgroup元数据表并重写数据、挂载新的coprocessor等；
- 升级master节点；
- 升级meta分组；
- 依次升级业务分组；



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

实操及问题

我们从19年下半年启动了这一轮滚动升级的调研与准备，今年3月下旬正式开始线上操作，截至5月初已完成了国内外共计9个集群的升级工作，用户无感知。

在此期间我们也遇到了不少未解问题，这里摘取一个Critical问题做简单介绍：
region split过程中叠加RS宕机引发数据丢失：
region split是一个相当复杂的事务过程，大体可分为以下几步：

- RegionServer开始执行split事务，在ZK region-in-transition下创建该region的节点，标记为SPLIT；
- Master监听到新的split节点，开始做一些初始化工作，并修改内存状态，完成后将节点状态改为SPLITING；
- RS监听到节点状态变为SPLITING，开始正式执行split——关闭父region、创建子region文件夹并添加引用文件、修改meta表记录两个子region并将父region下线；
- 子region上线；

当父region下线、子region还未正式上线时RegionServer宕机，master上的ServerCrashProcedure线程开始进行回滚，会将子region删除；此外master上还有一个CatalogJanitor线程做数据清理，正常split过程中由于ZK上存在对应节点，这个线程会被阻塞；然而由于RS宕机，临时节点也随之消失，该线程正常工作，判断meta表中父region已经下线，从而将父region删除——至此父子region皆被删除，导致数据丢失。
修复方案已提交社区：HBASE-23693

其它patch list：

- HBASE-22620 修复replication znode积压问题；
- HBASE-21964 支持按Throttle Type取消quota设置；
- HBASE-24401 修复 hbase.server.keyvalue.maxsize=0 时append操作失败的问题；
- HBASE-24184 修复只使用simple ACL时snapshot相关操作的鉴权问题；
- HBASE-24485 Backport，优化堆外内存初始化时间；
- HBASE-24501 Backport，去除ByteBufferArray中非必要的锁；
- HBASE-24453 Backport，修复挪动表分组时缺少校验逻辑的问题；

总结

本次升级工作从立项到完结耗时近一年，能够成功完成非常开心。一方面本次升级极大拉进了内部版本与社区release版本的距离，为更加良性的版本迭代及社区互动夯实了基础；另一方面新版本引入了诸多新特性，在稳定性、易用性方面都为我们带来了更为广阔的成长空间；更重要的是在这个过程中我们自身也沉淀出了一套系统的工作思路与方法论，期待后续可以更好的为业务赋能，为公司创造价值。

作者简介
唐天航，滴滴资深软件开发工程师，专注于HBase内核研发，滴滴HBase服务及上下游生态的建设与维护。

本文转载自：https://mp.weixin.qq.com/s/d6Lca7Y5SHBY-1sDBohEKA