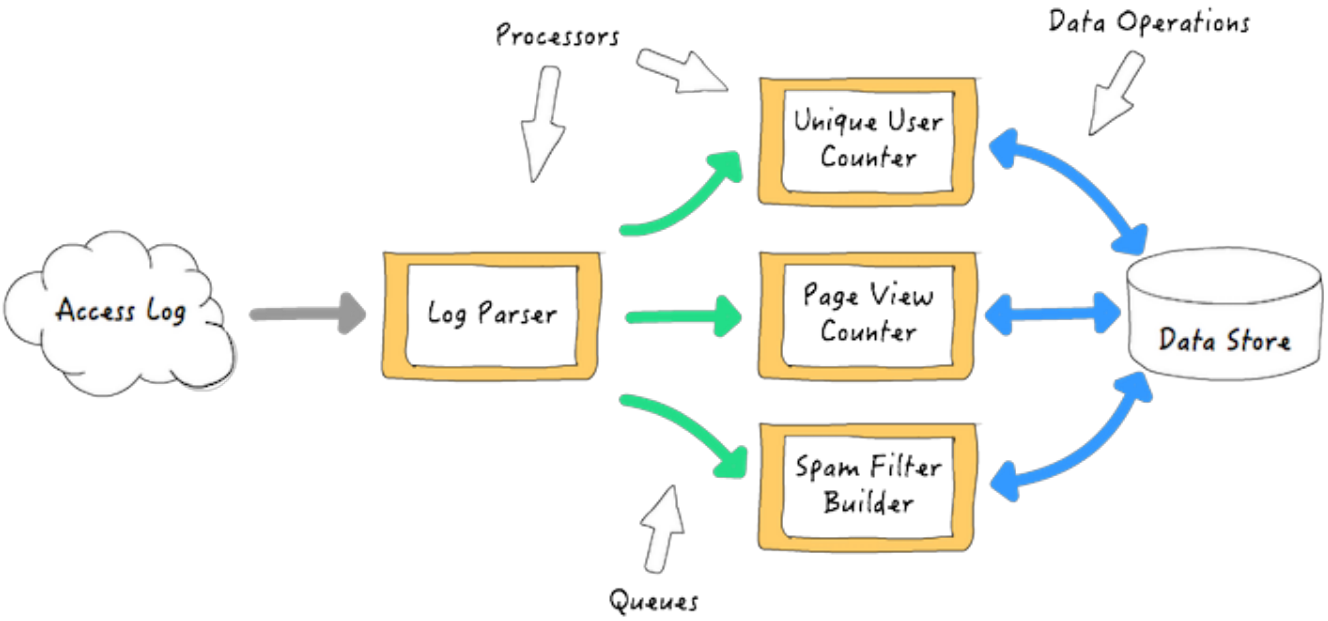
基于 HBase 构建可伸缩的分布式事务队列

一个实时流处理框架通常需要两个基础架构：处理器和队列。处理器从队列中读取事件，执行用户的处理代码，如果要继续对结果进行处理，处理器还会把事件写到另外一个队列。队列由框架提供并管理。队列做为处理器之间的缓冲，传输数据和事件，这样处理器可以单独操作和扩展。例如，一个web 服务访问日志处理应用，可能是这样的：



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

框架之间的主要区别在于队列语义，通常不同之处有以下几点：

- 调度保障机制: 至少一次，至多一次，只有一次。
- 容灾机制: 失败对用户和自动恢复是透明的。
- 可用性: 数据在出现错误后可以保存并重启。
- 可扩展性:产品/用户增加时的局限性。
- 性能Performance: 队列操作的吞吐量和延迟。

我们想在开源的 Cask Data Application Platform (CDAP)上提供一个动态可扩展，强一致性并且有一次性交易机制的实时流处理框架，在这个强大的机制保护下，开发者可以自由操作任何形式的数据操作而不用担心不一致性，潜在的返工和失败。它可以帮助开发者在没有分布式系统背景的情况下建立他们的大数据应用。此外，如果需要可以关闭这种强大的保护机制换取高性能。它总是比其他方式更容易使用。

可扩展队列

队列有两种基本操作：入队和出队。生产者将消息写到队头（入队），消费者从队尾读取数据（出队）。如果做为一个整体你添加更多生成者时的入队速度和添加更多消费者时的出队速度足够快，我们说这个队列是可扩展的。

理想状态下，扩展是线性的，这意味着两倍的生产者 /消费者，会产生两位速度的出队/入队，增长只受集群的规模限制。为了支持生产者的线性扩展，队列需要一个存储系统并且需要当前写入者的数量线性扩展。为了应对消费者的线性扩展，队列可以分区，例如一个消费者只处理队列中的一段数据。

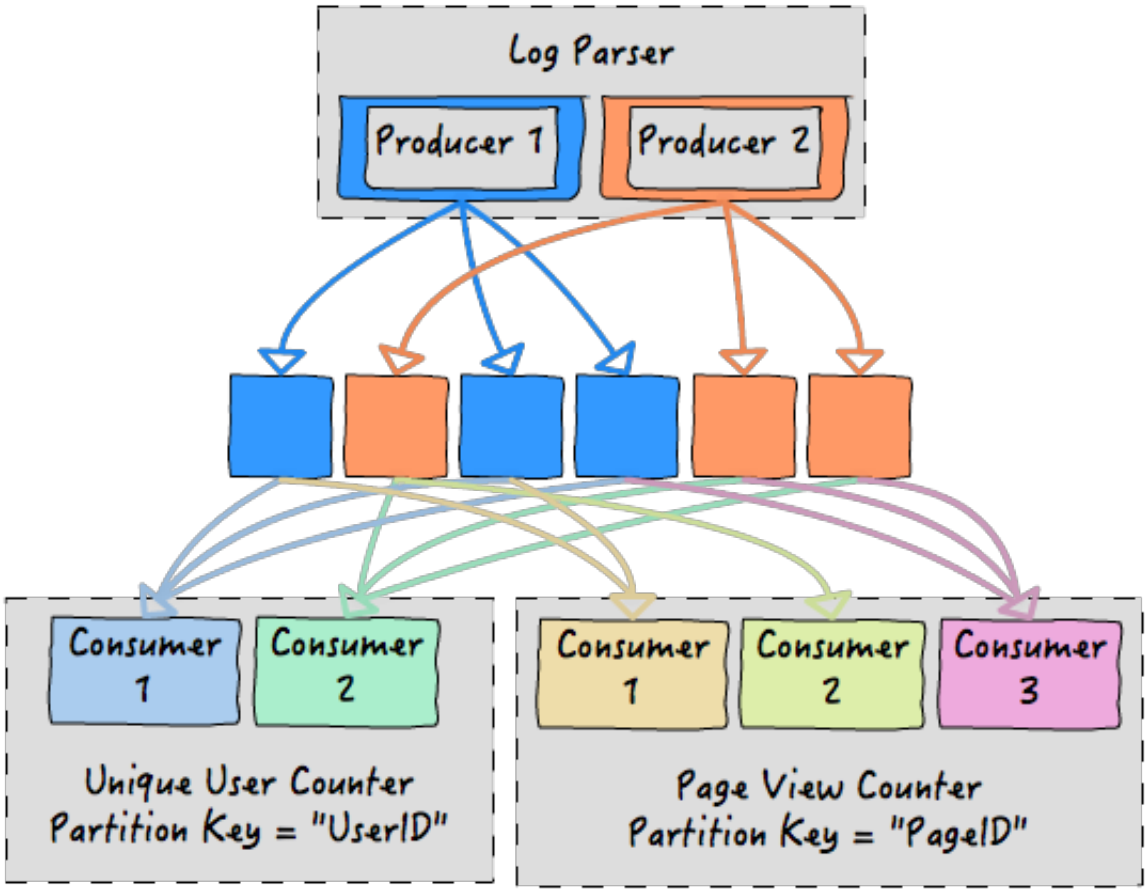
队列扩展的另一个方面是它应该可以横向扩展。这意味着队列性能的上限可以通过增加集群结点的方式来提升。这是很重要的，它可以保证队列不受当前集群大小限制根据数据的增长而扩展。

分区的 HBase 队列

我们选择 Apache HBase 做为队列的存储层。它为存储强一致性，可横向扩展的行数据做了设计和优化。它的并发写操作性能非常好，并提供了有序扫描以支持分区消费者。我们使用 HBase Coprocessors 的高效扫描滤波和队列清洗。为了在队列上使用一次性语义，我们用 Tephra's 为 HBase 提供传输支持。

生产者和消费者具有操作独立性。每个生产者通过 Hbase puts
批处理执行入队操作，消费者通过执行 Hbase Scans
执行出队操作。生产者和消费者的数量之间没有关联，他们可以分离。

此队列存在一个消费者组的概念。一个消费者组，是由相同的关键字划分的消费者集合，这样，每个发布到队列的事件，就会由此消费者组中的消费者去消费。使用消费者组，可以通过不同的关键字划分同一个队列，同时，也可以通过数据的操作性特点来拓展。按照上面访问日志分析的例子，生产者和消费者组可能看起来像这样：

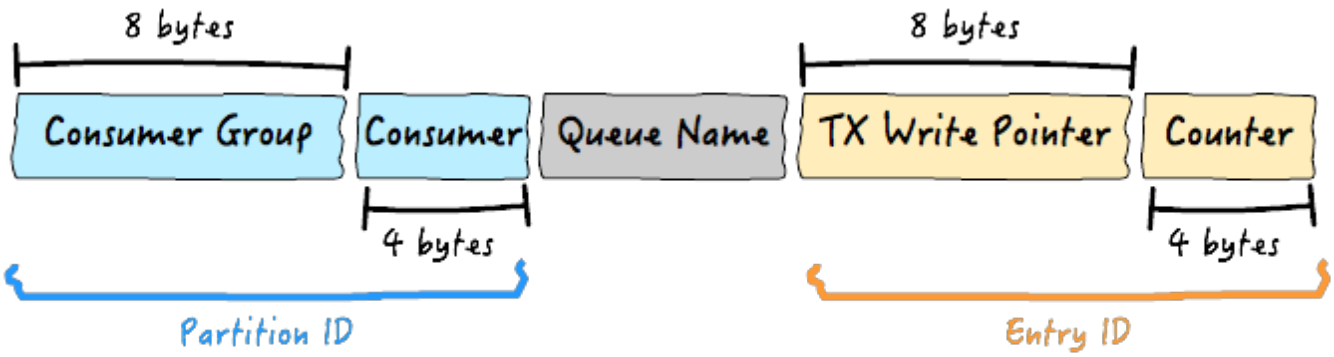


如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

对于Log Parser，这里有两个生产者在运行，它们并发的向队列写数据。在消费者这边，这里存在两个消费者组。 Unique User
Counter组有两个消费者，使用UserID作为划分（队列的）关键字。Page View
Counter组则有三个消费者，使用 PageID 作为划分（队列的）关键字。

队列行值格式

当一个事件通过一个生产者被发布出去，一个或多个消费者组合将收到消息，我们把事件写入HBase 表的一个或多个行上，那么这条记录就被设计成适用于每个消费者组。事件的有效负荷和元数据被存储在独立的列上，那么行的值就是下面这样的格式：



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

的值是分区 ID 和整个 ID。分区 ID 通过限定行值前缀再提供给消费者。消费者只被允许读数据，并在出队的时候使用前缀扫描。分区 ID 由两部分组成：一个消费者组 ID 和一个消费者 ID。生产者计算出每个消费者组的分区 ID，并通过入队写到那些行。

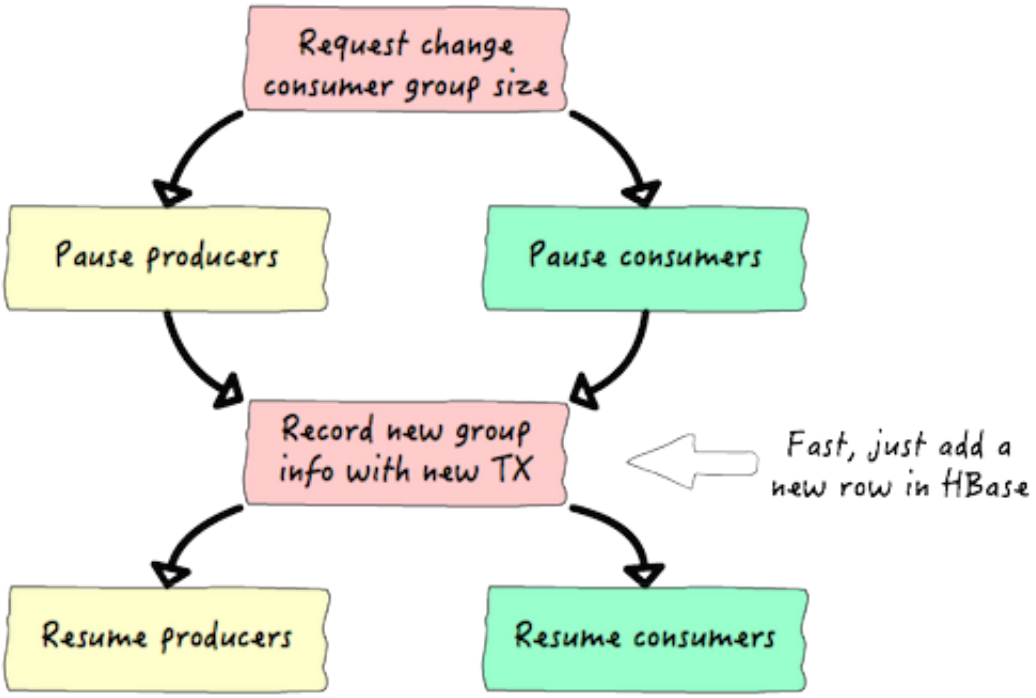
行关键字中的入口 ID（Entry ID）包含了事务信息。它由 Tephra 触发的生产者事务写指针和单向增长的计数器组成。这个计数器由本地的生产者生成，同时，针对事件，计数器需要让行关键字唯一，因为生产者可以在同一个事务中将多个事件加入队列。

出队列的时候，计数器会使用事务写指针来决定，队列入口是否已经提交，以及是否可以消费了。事务写指针和计数器的组合，使得行关键字总是唯一的。这让生产者可以独立的操作，而不会有写冲突。

为了生成分区 ID（Partition ID），生产者需要知道大小和每个消费者组的分区关键字。当应用程序启动，以及组大小发生任何变化的时候，消费者组信息都会被记录下来。

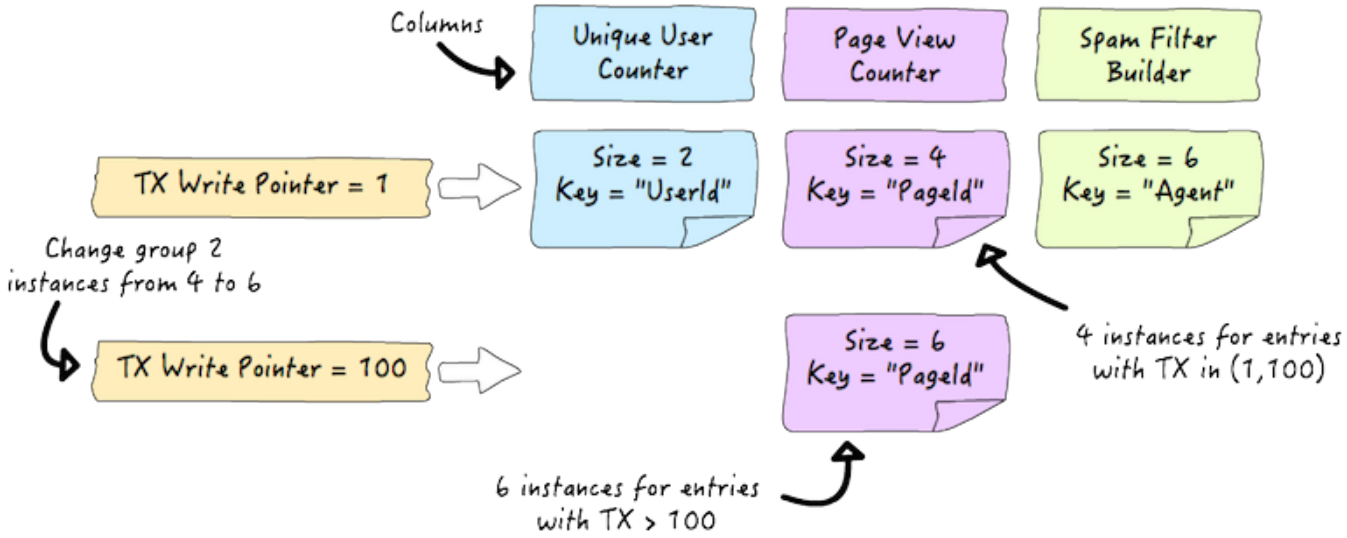
改变生产者和消费者

增加或减少生产者是很直接的，因为每个生产者都是独立操作的。增加或减少生产者进程就可以满足这个要求。然而，当消费者组的大小需要改变的时候，就需要协调来正确更新消费者组的信息。可以用下面的图表来概括所需的步骤：



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

由于暂停和恢复是由 Apache ZooKeeper
来协调的，同时它们也是并行执行的，所以它们是两个非常快速的操作。例如，之前我们提到的
Web 访问日志分析应用程序，改变消费者组信息的过程可能看起来像这样：



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

基于这个队列的设计，入队列和出队列的性能，与单独的批量 HBase Puts 和 HBase Scans
不相上下，这样也带来与 Tephra 服务器进行通讯的开销。通过在同一个业务处理中将多个事件

批量处理，可以大大降低这个开销。

最后，为了避免“热点聚焦（hotspotting）“，我们基于簇的大小提前分割了 HBase 表，同时，在行关键字（row key）上采用 加盐（salting）的方式来更好的分配写。否则，由于是单调的增加业务处理写指针，行关键字就会是连续的。

性能值

我们在小型的 10 节点的 HBase 集群上已经测试过性能，结果令人印象深刻。使用 1K 字节负载，以 500 个事件为一个批次大小，我们完成了生产和消费 100K 个事件/秒的吞吐量，其中运行了 3 个生产者和 10 个消费者。我们也观察到当我们增加消费者和消费者的时候，吞吐量线性增加：例如，当我们将生产者和消费者数量加倍的时候，吞吐量增加到 200K 个事件/秒。

在 HBase 的帮助下，结合最佳实践，我们成功的创建了一个线性可伸缩的，分布式事务队列系统。同时，在 CDAP 中使用这个系统提供实时流处理框架：动态可伸缩，强一致性，以及一次交付的传输保证。

英文原文：Scalable Distributed Transactional Queues on HBase