Spark on kubernetes：生产环境使用前需要解决的问题

Spark从2.3版本已经支持以云原生的方式运行在kubernetes上，这对于大数据和云原生来说，是二者的重要会师。相信未来会摩擦出更加精彩的火花。一个是大数据计算框架的翘楚，关注的是数据怎么变现(挖掘价值)，一个是容器编排事实上的标准，关注的是如何更加高效的使用资源。二者解决的是不同领域的问题，相信二者的结合，对于架构的统一，使得技术栈归一；资源利用率的提升，使得运维成本的降低。对于技术栈的统一，也是人力成本和心智成本的一大降低，使得我等码农可以聚焦技术本身，专心突破，不用在技术选型时将几个同类型的技术来个你死我活的比较。更重要的一点，对于大数据来说，云原生和大数据势必会走向统一，大数据借力云原生，使得资源的调度更加灵活有弹性，形成云原生大数据平台。想想都激动人心。

废话来了个开篇，下边言归正传。

其实早在Spark2.2版本的时候，Spark社区就开始以kubernetes容器集群作为Cluster Manager的实现，引入kubernetes来管理Spark集群，不过那时候，功能尚在襁褓之中，再加上kubernetes也没有像现在被普罗大众广泛接受用来做基础设施。不过幸运的是，经过三年的持续迭代发展，Spark on kubernetes已经茁壮成长，闪亮登场了。在此不得不佩服社区大神们当初的远见和见识。

但是当二者门不当户不对的spark和kubernetes走到一起的时候，像极了电视剧的爱情故事，可谓是经历了一波三折。毕竟Hadoop已经是大数据领域事实上的标准，Spark和YARN的结合更是一段人间佳话，一个负责计算数据，一个负责为计算节点提供资源，二者一个主内一个主外，配合默契。但是随着技术和通信基础设施的发展，网络性能得到的很大程度的提升，大数据处理的瓶颈逐渐从网络转向了CPU，以前移动计算不移动数据的架构的缺点也逐渐暴露出来。毕竟对于大部分业务场景来说，都存在波峰和波谷，不同时间段的计算需求是不一致的，对应的计算资源也是不一样的，但是传统的大数据平台是一个固定的资源，无法关闭闲置的资源和节点，造成了资源浪费。并且，公司业务多样化，不通的业务需要维护一套不通的平台，造成资源孤岛化，并且需要不通的人力去维护不通的系统，所以迫切需要一套统一的平台来广纳百川，能够部署多样化的应用，资源形成池化，统一调度，提升资源利用率。

在这样的大背景下，kubernetes横空出世，给IT从业者带来了曙光，他统一了开发、测试、生产三者环境的不对等性，并打着build once ,run every where的旗号到处抛媚眼。你给别人说你们公司的基础资源是kubernetes构建的容器云，一股自豪感就会油然而生，毕竟容器云代表着资源的灵活分配，代表着IT行业的共产主义，按需分配。

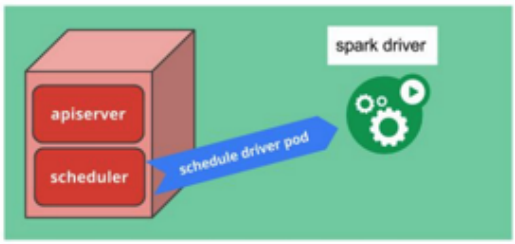
但是。。。

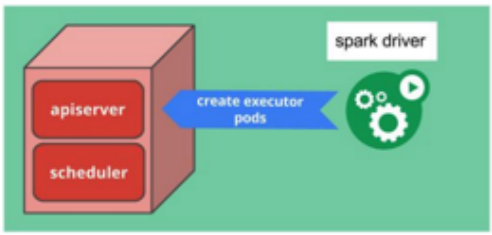
凡事都有一个但是，kubernetes怎么帅气。想要和美丽的Spark和谐幸福相处，也不是那么容易的。毕竟Spark社区目前的Spark3.0正式版本还一直没有发布出来。毕竟在生产中，有几个问题需要解决。

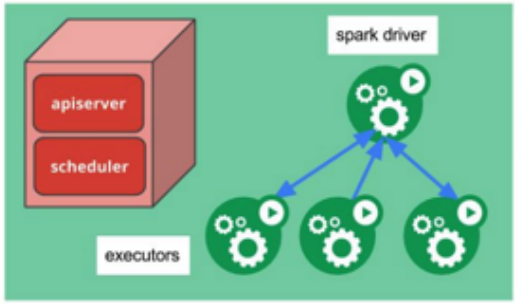
问题

1.external-shuffle-service

spark已经是以云原生的模式运行在kubernetes上了，也就是native模式。那什么是native模式呢？简单来说就是driver和executor都已经Pod化了。driver和executor的调度已经可以按照Pod的调度模式了。



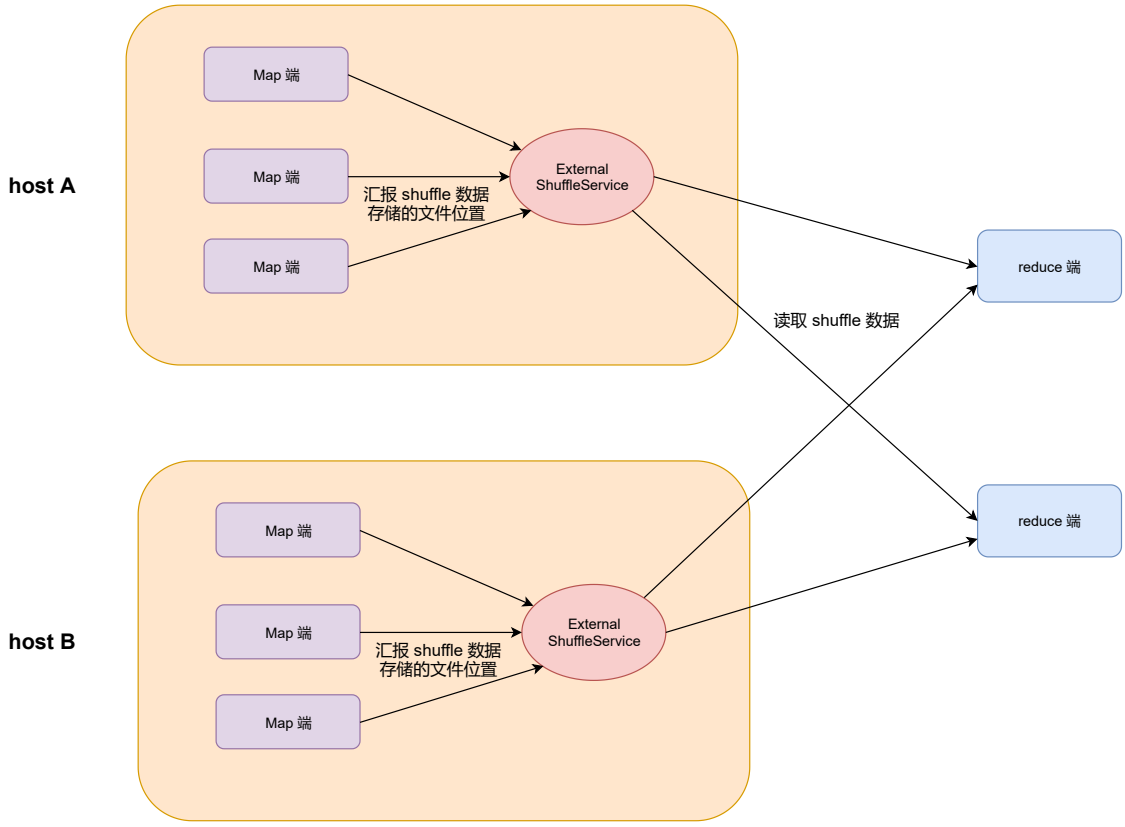




但是想要executor数量随着计算数据量的大小动态伸缩调整，提升资源利用率，就需要依赖external-shuffle-service功能了。在spark的shuffle过程中，一个executor需要到另外一个executor那里取数据，如果一个executor节点挂了，那么他就无法处理其他executor发过来的shuffle的数据读取请求了，上一个stage计算的数据也就没有意义了。

为了解决"取shuffle数据"和"目标executor是否运行"分开的问题，Spark引入了external-shuffle-service服务，也就是先把shuffle数据暂存到external-shuffle-service，然后下一个stage的executor再去external-shuffle-service那里取数据就可以。具体的原理可以参考文章：https://zhmin.github.io/2019/08/05/spark-external-shuffle-service/

在Spark架构中，external-shuffle-service是部署到每一个节点上的。



executor计算完之后告诉external-shuffle-service数据存到那里，然后external-shuffle-service记下来供下一stage查询。

shuffle数据是只支持HostPath的。executor容器跑在kubernetes节点上一个容器上，external-shuffle-service跑在另外一个容器上，二者要想共享相同的Path文件，就必须使用节点路径，要用hostpath就必须拥有节点的所有权，但是这对于多用户共享kubernetes集群来说，权限不安全，数据未隔离。

解决

那么怎么解决这个问题呢？社区对这个问题也有个专门的讨论。https://docs.google.com/document/d/1uCkzGGVG17oGC6BJ75TpzLAZNorvrAU3FRd2X-rVHSM/edit#heading=h.btqugnmt2h40

这个文档主要是表达当前external-shuffle-service的实现有缺点：

1. 多个Spark应用共用一个external-shuffle-service，如果external-shuffle-service出问题，多个Spark应用都受影响，即隔离性差。
2. 一个节点一个external-shuffle-service，导致不同节点间压力不均衡。同时如果节点挂了，external-shuffle-service也就没了，这个节点上面的所有executor都受影响，可靠性差。
3. 在当前较火热的Docker容器环境下，executor写入的shuffle数据（在一个容器内）。不一定就能被external-shuffle-service读取到（在另一个容器内）。因为有些Kubernetes集群中，管理员出于安全考虑，会强制隔离不同用户的容器，禁止任何共享。

所以提出了**改进方向**：即executor保存shuffle数据时，不限定非得是保存在本地Path中。

具体实现方案可以有多种：

1. 保存shuffle数据时，通过external-shuffle-service上传的方式。
2. external-shuffle-service支持shuffle数据为远端uri地址，而不仅仅是主机路径。
3. 由Driver来维护所有的shuffle数据信息，取消external-shuffle-service组件。
4. 将shuffle数据保存到分布式存储中。
5. 将shuffle数据上传到external-shuffle-service，然后由Driver跟踪文件路径。

总体思路就是：以前external-shuffle-service是本地写，远程读，**调整为远程写，远程读。**

其实要在Kubernetes上实现executor数量动态调整（dynamic resource allocation），还有另一条小路（即不通过external-shuffle-service的方式）。并且这条路已经实现了，在如下PR里：

https://github.com/apache/spark/pull/24817

实现原理：

当发现executor里面是shuffle数据没有用了，则可以删除该executor。如果这个executor里面的shuffle数据，还会被其他job读取，那么就保持这个executor存活着不被删除。从而实现executor数量可以动态调整。

缺点：

可以看出，这种方式其实是缓兵之计。

- 删除部分暂时不被使用executor，但是必须保留那些还会被使用的executor。所以动态效果并不是最优的。
- 一个executor也许最近不被使用，被删除了。但是后续其他Stage又有可能去访问那个shuffle数据。结果发现找不到（被动态删除嘛），这个时候又得重新计算，浪费性能。

PR里面的讨论也表示这是无法用来完整替代external-shuffle-service的。

通过上面的分析，可以基本了解在kubernetes上面跑external-shuffle-service的困难和思路。

所以要达到目的的路径为：

- external-shuffle-service支持远端保存shuffle数据
- executor和external-shuffle-service共享云端shuffle数据
- executor数量可以动态调整，不影响功能
- 在kubernetes上支持了executor数量动态调整（dynamic resource allocation）。

看Spark的规划是在 3.0.0 版本提供完整能力,让我们期待这个版本早日面世。

https://issues.apache.org/jira/browse/SPARK-24432

2.动态资源调度(Dynamic Resouce Allocation)

Spark on YARN开始支持动态资源调度。并且目前也很完美，在on YARN模式下，开启Dynamic Resouce Allocation，在官方文档上也给出了方案：

```
spark.dynamicAllocation.enabled=true
spark.shuffle.service.enabled=true
```

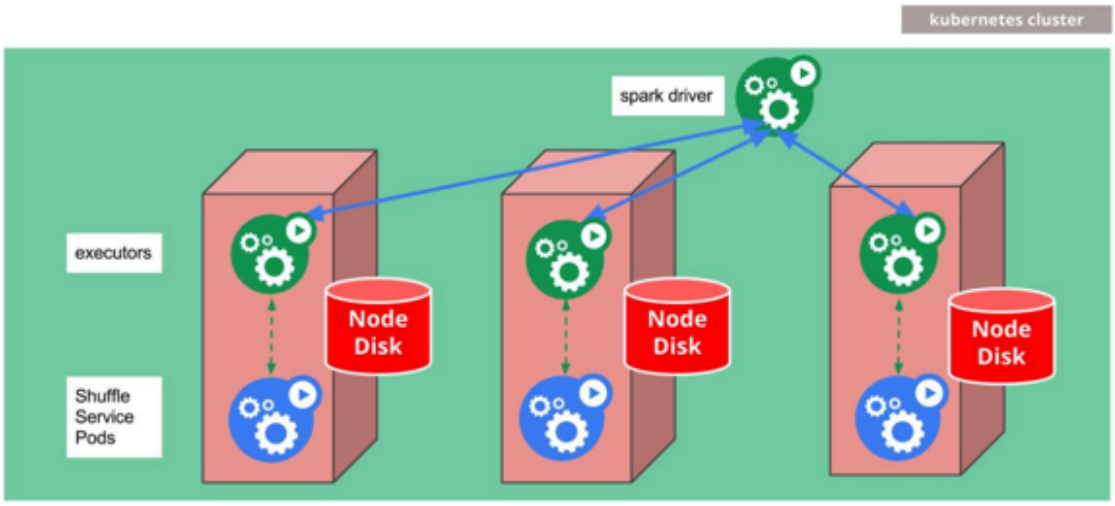
在没有Dynamic Resouce Allocation之前，在计算过程中，不管数据集的大小，一旦spark-submit启动之后，都只有固定的executor来工作。但是实际场景是，业务有闲忙，在凌晨或者业务闲时，会释放大量的资源，但是这部分资源因为固化给了spark任务而不能被其他任务使用。这是一种资源浪费。好在社区在spark1.2版本之后引入了Dynamic Resouce Allocation，他能够根据当前集群的资源和工作的负载，来动态的调整executor的数量。

但是到目前spark2.4.5，社区官方也没有支持on kubernetes的Dynamic Resouce Allocation。但是社区也有这方面的讨论：

https://issues.apache.org/jira/browse/SPARK-25299

```
1.每个Node上的External Shuffle Service 进程如果挂了，那么Shuffle数据依然需要被recompute。
2.External Shuffle Service必须 Long Running的，而且当不需要Shuffle的时候会导致了另一种方式的资源浪费。
3.External Shuffle Service的数据写在本地磁盘，这不利于一些容器环境下的 IO 隔离。
```

下图是 on Kubernetes 的 External Shuffle Service 在 Kubernetes 部署的原型图。可以看到，Executor Pod 和 External Shuffle Service Pod 在同一个 Node 上运行，共享 Node 上的 Disk。



尽管官方仍未正式支持 on Kubernetes 的 Dynamic Resouce Allocation，但是目前 Master 分支合进去了一个很有趣的 commit https://github.com/apache/spark/pull/24817。我们留意到，这个 commit 做的事情是可以不通过 External Shuffle Service 来开启 Dynamic Resouce Allocation。

当遇到 ShuffleMapStage 的时候，会记录下产生 shuffle 数据的 Executor Id，并且会让这些 Executor 在这个 Stage 对应的 Job 没有结束的时候不被移除，这样使得后面如果有 Stage 需要对 ShuffleMapStage 的数据进行读取的时候，相关的数据可以不需要 recompute。

如果这些 shuffle file 一直没有被其他 Stage 使用呢，那占用的 Executor 是不是很浪费？其实还可以通过设置一个 timeout 参数 spark.dynamicAllocation.shuffleTimeout，当超时的时候，不管这些 shuffle 文件还是否有对应的 Active Job，也会把这些 Executor 移除释放资源。

按照这个逻辑，当一个 Stage 需要1000个 Executor 的时候，再下一个 Stage 计算得到只需要10个 Executor，比较理想的情况是可以很快释放990个 Executor 占用的资源。这可能有点困难，因为 shuffle file 往往会被持有一段时间以确保下面的 Stage 确实不需要了才会被标记，而下一个 Stage 什么时候才需要前面的 shuffle 文件是无法确定的，所以如果马上移除990个 Executor，可能也不是一个很好的方案。