Apache Spark 将支持 Stage 级别的资源控制和调度

背景

熟悉 Spark 的同学都知道，Spark 作业启动的时候我们需要指定 Exectuor 的个数以及内存、CPU 等信息。但是在 Spark 作业运行的时候，里面可能包含很多个 Stages，这些不同的 Stage 需要的资源可能不一样，由于目前 Spark 的设计，我们无法对每个 Stage 进行细粒度的资源设置。而且即使是一个资深的工程师也很难准确的预估一个比较合适的配置，使得作业启动时设置的参数适合 Spark 每个 Stage。

我们来考虑这个这样的场景：我们有个 Spark 作业，它总共有两个 Stages。第一个 Stage 主要对我们输入的数据进行基本的 ETL 操作。这个阶段一般会启动大量的 Task，但是每个 Task 仅仅需要少量的内存以及少数的核（比如1个core）。第一个 Stage 处理完之后，我们将 ETL 处理的结果作为 ML 算法的输入，这个 Stage 只需要少数的 Task，但是每个 Task 需要大量的内存、GPUs 以及 CPU。

像上面这种业务场景大家应该经常遇到过，我们需要对不同 Stage 设置不同的资源。但是目前的 Spark 不支持这种细粒度的资源配置，导致我们不得不在作业启动的时候设置大量的资源，从而导致资源可能浪费，特别是在机器学习的场景下。

不过值得高兴的是，来自英伟达的首席系统软件工程师 Thomas Graves 给社区提了个 ISSUE，也就是 SPIP: Support Stage level resource configuration and scheduling，旨在让 Spark 支持 Stage 级别的资源配置和调度。大家从名字还可以看出，这是个 SPIP（Spark Project Improvement Proposals 的简称），SPIP 主要是标记重大的面向用户或跨领域的更改，而不是小的增量改进。 所以可以看出，这个功能对 Spark 的修改很大，会对用户产生比较大的影响。

作者提完这个 SPIP 之后给社区发了一份邮件，说明这个 SPIP 的目的，解决的问题等，然后让大家进行投票决定这个 SPIP 要不要开发下去。值得高兴的是，已经社区的一轮投票，得到6票赞成1票反对的结果。那就说明这个 SPIP 通过了，将进入开发状态。

From	Thomas graves <tgra...@apache.org>
Subject	[RESULT][VOTE] [SPARK-27495] SPIP: Support Stage level resource configuration and scheduling
Date	Sat, 14 Sep 2019 03:02:08 GMT

Hi all,

The vote passed with 6 +1's (4 binding) and no -1's.

+1s (* = binding) :
Bobby Evans*
Thomas Graves*
Dongjoon Hyuni*
Felix Cheung*
Bryan Cutler
Ryan Blue

Thanks,
Tom Graves

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

设计

前面扯了一堆，下面让我们来看看这个方案是如何设计的。为了实现这个功能，需要在现有的 RDD 类里面加上一些新的 API，用于指定这个 RDD
计算需要用到的资源，比如添加以下两个方法：

```
def withResources(resources: ResourceProfile): this.type
def getResourceProfile(): Option[ResourceProfile]
```

上面的 withResources 方法主要用于设置当前 RDD 的 resourceProfile，并返回当前 RDD 实例。ResourceProfile 里面指定的资源包括
cpu、内存和额外的资源（GPU/FPGA/等）。我们还可以利用它实现其他功能，比如限制每个 stage 的 task 数量，为 shuffle 指定一些参数。不过为了设计实现的简单，目前只考虑支持 Spark 目前支持的资源，针对 Task 可以设置 cpu 和额外的资源（GPU/FPGA/等）；针对 Executor 可以设置 cpu、内存和额外的资源（GPU/FPGA/等）
。执行器资源包括cpu、内存和额外资源(GPU、FPGA等)。通过给现有 RDD 类添加上面的方法，这使得所有继承自 RDD 的演变 RDD
都支持设置资源，当然包括了输入文件生成的 RDD。

用户在编程的时候，可以通过 withResources 方法来设置 ResourceProfile 的，当然肯定不可以设置无限的资源。可以通过 ResourceProfile.require 同时设置 Executor 和 task 需要的资源。具体的接口如下所示：

```
def require(request: TaskResourceRequest): this.type
def require(request: ExecutorResourceRequest): this.type
```

```
class ExecutorResourceRequest(
 val resourceName: String,
 val amount: Int, // potentially make this handle fractional resources
 val units: String, // units required for memory resources
 val discoveryScript: Option[String] = None,
 val vendor: Option[String] = None)

class TaskResourceRequest(
 val resourceName: String,
 val amount: Int) // potentially make this handle fractional resources
```

之所以用 ResourceProfile 包装 ExecutorResourceRequest 或 TaskResourceRequest 是因为后面如果我们需要添加新功能可以很容易的实现。比如我们可以在 ResourceProfile 里面添加 ResourceProfile.prefer
方法，来实现程序申请到足够就运行这个作业，如果没有申请到足够资源就使作业失败。

当然，这个功能的实现需要依赖 Spark 的 Dynamic Allocation 机制。如果用户没有启用 Dynamic Allocation（spark.dynamicAllocation.enabled=false）或者用户没有为 RDD 设置 ResourceProfile，那么就按照现有的资源申请那套机制运行，否则就使用这个新机制。

因为每个 RDD 都可以指定 ResourceProfile，而 DAGScheduler 是可以把多个 RDD 的转换放到一个 stage 中计算的，所以 Spark 需要解决同一个 stage 中多个 RDD 的资源申请冲突。当然，一些 RDD 也会出现跨 Stages 的情况，比如 reduceByKey，所以针对这种情况 Spark 需要将 ResourceProfile 的设置应用到这两个 Stage 中。

如何使用

那么如果 RDD 添加了上面的方法，我们就可以想下面一样设置每个 Task 的资源使用情况：

```
val rp = new ResourceProfile()
rp.require(new ExecutorResourceRequest("memory", 2048))
rp.require(new ExecutorResourceRequest("cores", 2))
rp.require(new ExecutorResourceRequest("gpu", 1, Some("/opt/gpuScripts/getGpus")))
rp.require(new TaskResourceRequest("gpu", 1))

val rdd = sc.makeRDD(1 to 10, 5).mapPartitions { it =>
  val context = TaskContext.get()
  context.resources().get("gpu").get.addresses.iterator
}.withResources(rp)

val gpus = rdd.collect()
```

上面 ResourceProfile 指定 Executor 需要 2GB 内存、2个 cores 以及一个 GPU；Task 需要一个 GPU。

总结

本文只是介绍了这个功能的简单实现，在真实的设计开发中会有很多需要考虑的问题，具体可以参见 SPARK-27495，对应的设计文件参见 Stage Level Scheduling SPIP Appendices API/Design 。因为这是个比较大的功能，所以可能需要花费数个月的时间去实现。相信有了这个功能之后，我们会更合理的使用集群的资源。