**DAGScheduler**

高层次面向stage的调度，为每一个job划分stage，跟踪每一个RDD和stage输出的物化，并且找到一种最小代价调度作业。

然后提交stage作为TaskSet给底层的**TaskScheduler实现者**去在集群中运行他们。一个TaskSet保存独立的任务可以基于集群的数据（或上一个map的输出）立即执行，但是数据如果成为不可获得的话(数据不存在或者删除，丢失等)，那么task将会失败。

Spark Stage的划分依据**Shuffle**的界限。RDD的操作的窄依赖，例如 map filter将会放在一个task中(言外之意 map filter不是划分依据)

但是Shuffle动作会划分Stage的。最后划分完Stage之后，只有不同的Stage之间存在shuffle，stage内部不允许存在shuffle（言外之意就是遇见Shuffle就切分Stage,Shuffle是划分的Stage的依据）每一个Stage中可能有多个操作（例如一个Stage中有多个map），这些操作组成的流水线当RDD调用compute时候会被执行。(Saprk算子分为transform和action)

除了切分Stage之外，DAGScheduler还会依据当前的Cache状态尽可能的Locations执行task，传递这些状态低层的TaskScheduler。DAGScheduler还会处理Shuffle数据文件丢失的错误，这种情况丢失数据的Stage需要再次提交，

如果一个Stage的错误**不是**由于文件丢失的话将会由TaskScheduler处理，TaskScheduler将会在取消该整个Stage之前重试少数几次每一个task.(言外之意文件丢失的话由DAGScheduler负责执行)

**---------------------------------------------------------------------------------------------------------------------------------**

**关键概念：**

Jobs ：

**Jobs是顶层的单元提交给Scheduler**。例如：当用户调用action(例如count())，一个Job将会使用submitJob提交。每一个Job可能需要多个Stages构建中间数据(使用Stage的输出数据)

Stages ：

**Stage是Task的集合**，用来在Job中计算中间数据，每一个task在同一个RDD的多个分区上执行相同的计算逻辑。Stage是根据Shuffle界限划分的，这将引入一个界限(后面的stage必须等待前面的stage执行完毕去拉取数据)。这里有**两种类型的Stage**：ResultStage是执行Action的最终stage，ShuffleMapStage是map为shuffle写输出文件的Stage，

Stage通常可以被多个job共享，如果这些Job都是在相同的RDD上执行计算。

Tasks ：

**Task是Work的独有的单元**，每一个Task都会发送到一台机器上运行

**Cache Tracking：**

DAGScheduler会解决哪些RDD被缓存，避免重新计算和 避免执行shuffle输出的Satge再次执行

**Preferred Locations：**

**尽可能本地化**：DAGScheduler将会依据RDD所有的位置偏好计算在哪里执行每一个Stage中的Task，或者在哪里缓存或者输出Shuffle Data

**Cleanup：**

当Job依赖的数据使用完毕之后将会清除这些数据，避免app长期运行导致内存泄露

**---------------------------------------------------------------------------------------------------------------------------------**

为了失败恢复，相同的stage可能需要执行多次，这称为"attempts"。如果TaskScheduler汇报一个task由于上一个map输出

文件丢失的话，DAGScheduler将会重新提交丢失的stage，这个可以通过完成的FetchFailed或ExecutorLost时间侦查到。

DAGScheduler会等一小段时间去检查是否其他节点也存在这种情况或者是Task失败，然后重新提交TaskSet给丢失的stage计算

丢失的Task。在这个过程中，我们必须从新创建Stage Object，因为之前的Stage Object对象已经被清除。失败的stage有可能由于attempt

还会执行，必须在正确的Stage Object中考虑事件(翻译的不是很得体)

Here's a checklist to use when making or reviewing changes to this class:

当做改变或者回顾时候需要看的清单：

- All data structures should be cleared when the jobs involving them end to avoid indefinite

accumulation of state in long-running programs.

当job执行完毕之后该job的所有的数据必须清除掉，避免无限的积累

- When adding a new data structure, update `DAGSchedulerSuite.assertDataStructuresEmpty` to

include the new structure. This will help to catch memory leaks.

添加一个新的数据结构时，` dagschedulersuite assertdatastructuresempty `来更新。包括新结构。这将有助于捕捉内存泄漏。