spark源码阅读笔记

harli

2015/1/19

# org.apache.spark.scheduler

## Task

1. Task
2. ResultTasks – 最后一个 stage – 输出到 driver
3. ShuffleMapTasks – 最后stage之前的stage – 输出到多个 buckets（基于task的 partitioner）

**case class** Aggregator[K, V, C]

根据配置：

**private val** *externalSorting* = SparkEnv.*get*.conf.getBoolean("spark.shuffle.spill", **true**)

false ： AppendOnlyMap

true ： ExternalAppendOnlyMap

*/\*\*  
 \* :: DeveloperApi ::  
 \* An append-only map that spills sorted content to disk when there is insufficient space for it  
 \* to grow.  
 \*  
 \* This map takes two passes over the data:  
 \*  
 \* (1) Values are merged into combiners, which are sorted and spilled to disk as necessary  
 \* (2) Combiners are read from disk and merged together  
 \*  
 \* The setting of the spill threshold faces the following trade-off: If the spill threshold is  
 \* too high, the in-memory map may occupy more memory than is available, resulting in OOM.  
 \* However, if the spill threshold is too low, we spill frequently and incur unnecessary disk  
 \* writes. This may lead to a performance regression compared to the normal case of using the  
 \* non-spilling AppendOnlyMap.  
 \*  
 \* Two parameters control the memory threshold:  
 \*  
 \** `*spark.shuffle.memoryFraction*` *specifies the collective amount of memory used for storing  
 \* these maps as a fraction of the executor's total memory. Since each concurrently running  
 \* task maintains one map, the actual threshold for each map is this quantity divided by the  
 \* number of running tasks.  
 \*  
 \** `*spark.shuffle.safetyFraction*` *specifies an additional margin of safety as a fraction of  
 \* this threshold, in case map size estimation is not sufficiently accurate.  
 \*/*@DeveloperApi  
**class** ExternalAppendOnlyMap[K, V, C](

# Shuffle

<https://issues.apache.org/jira/browse/SPARK-2045>

runTask –-getWriter

ShuffleMapTask -> runTask – ShuffleManager->writer --> shuffleBlockManager-*ShuffleWriterGroup*

consolidateShuffleFiles

--ShuffleFileGroup(*ShuffleBlockId*)

--*ShuffleBlockId*

blockManager - DiskBlockObjectWriter

## Hash-based Shuffle

文件名：shuffle+shuffle\_id+map\_id+reduce\_id

## Sort-based Shuffle

sort-based shuffle的一个map task只生成一个shuffle文件，后续的reduce task是如何从这一个shuffle文件中得到自己的partition呢，这个时候就需要引入一个新的文件类型即index文件。

其具体实现步骤如下:

1. Map Task在读取自己输入的partition之后，将计算结果写入到ExternalSorter
2. ExternalSorter会使用一个map来存储新的计算结果，新的计算结果根据partiton分类，如果是有combine操作，则需要将新的值与原有的值进行合并
3. 如果ExternalSorter中的map占用的内存已经超越了使用的阀值，则将map中的内容spill到磁盘中，每一次spill产生一个不同的文件
4. 当输入Partition中的所有数据都已经处理完毕之后，这时有可能一部分计算结果在内存中，另一部分计算结果在spill的一到多个文件之中，这时通过merge操作将内存和spill文件中的内容合并整到一个文件里
5. 最后将每一个partition的在data文件中的起始位置和结束位置写入到index文件

## ShuffleManager

在SparkEnv中初始化

配置："spark.shuffle.manager", "sort"

|  |  |  |
| --- | --- | --- |
| 配置值 | 实现类 | 备注 |
| "hash" | "org.apache.spark.shuffle.hash.HashShuffleManager" |  |
| "sort" | "org.apache.spark.shuffle.sort.SortShuffleManager" | 默认 |

## ShuffleMemoryManager

在SparkEnv中初始化

|  |  |  |
| --- | --- | --- |
| 配置值 | 默认值 | 备注 |
| " spark.shuffle.memoryFraction " | 0.2 |  |
| " spark.shuffle.safetyFraction " | 0.8 |  |

*ExternalAppendOnlyMap*

*ExternalSorter*

## ShuffleClient

在SparkEnv中初始化

配置："spark.shuffle.blockTransferService", "netty"

|  |  |  |
| --- | --- | --- |
| 配置值 | 实现类 | 备注 |
| "netty" | NettyBlockTransferService | 默认 |
| "nio" | NioBlockTransferService |  |

# storage

## BlockManagerMessages

BlockManagerMessages ：定义 BlockManagerSlave 与 BlockManagerMaster 之间传递的消息类的工厂对象。