2018/03/10

参考：https://www.csdn.net/article/2015-07-10/2825184

SPARK的定位

能够进行批处理、交互查询、流式计算的全能工具？

Apache Spark的安装

略

通过pip安装

pip install pyspark

核心概念

RDD

弹性分布式数据集，它是由数据组成的不可变分布式集合，其主要进行两个操作：transformation和action（理解成一個數據分片？）。

Transformation是类似在RDD上做 filter()、map()或union() 以生成另一个RDD的操作，而action则是count()、first()、take(n)、collect() 等促发一个计算并返回值到Master或者稳定存储系统的操作。

Transformations一般都是lazy的，直到action执行后才会被执行。Spark Master/Driver会保存RDD上的Transformations。这样一来，如果某个RDD丢失（也就是salves宕掉），它可以快速和便捷地转换到集群中存活的主机上。这也就是RDD的弹性所在。

示例：從文本中發現五個最長用的詞

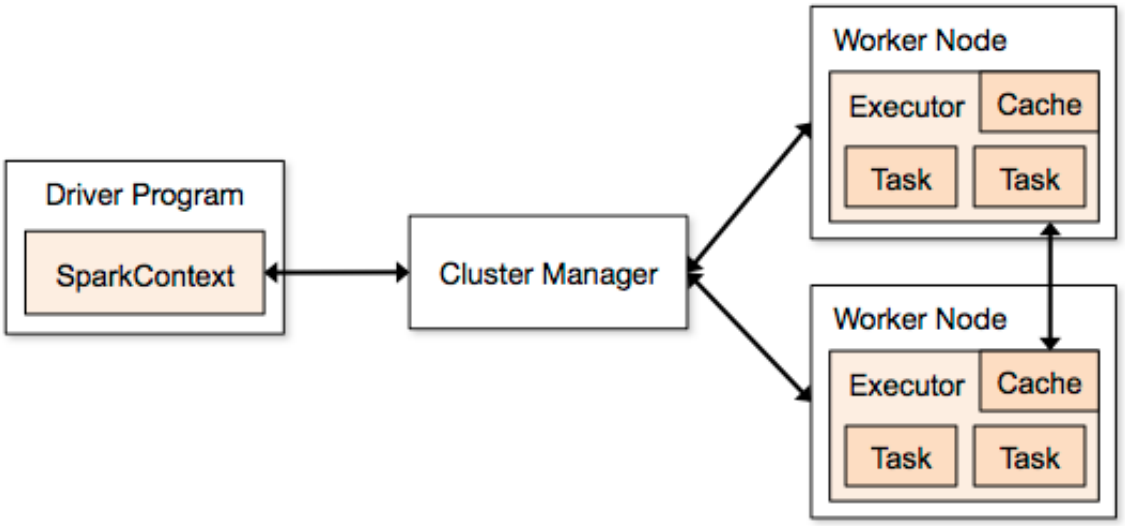
2018/03/12

QuickStart

略

2018/03/13

Overview



位于客户端上的SparkContext，会连接Cluster Manager（可以是Mesos/YARNMesos/YARN），请求在工作节点上创建Executor进程。

Executor进程进程管理工作节点上的计算和存储资源。当Executor建立之后，SparkContext会分发Jar和Py文件到工作节点，并且通知Executor执行任务。

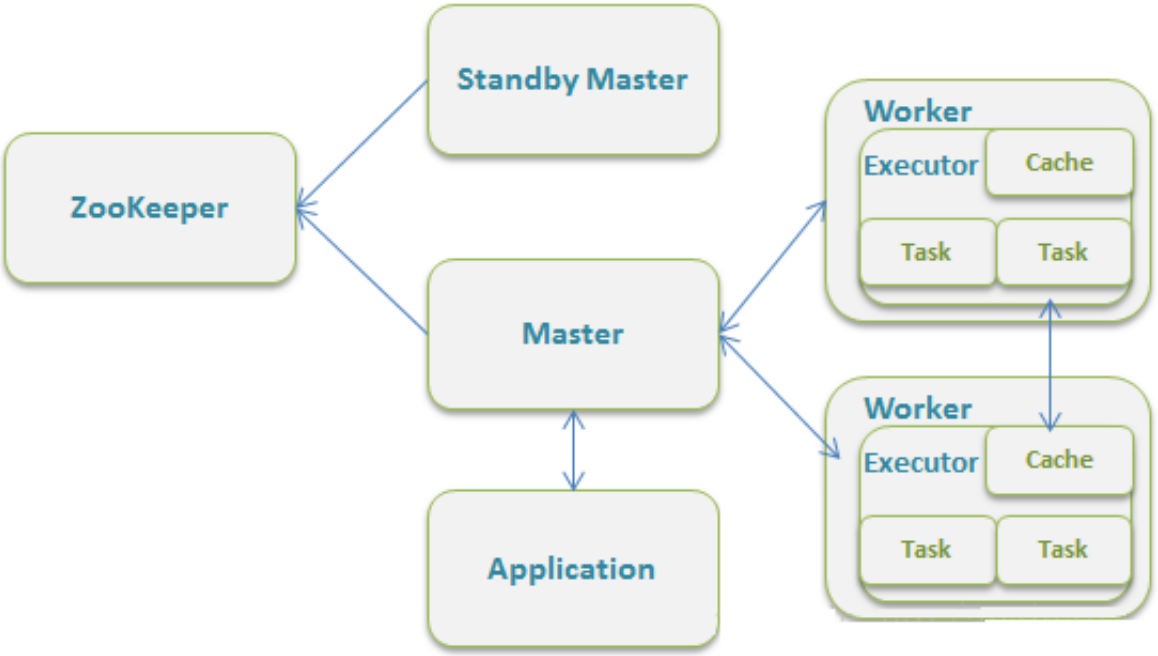
1. 不同Executor之间没有通信机制，只能用共享存储进行交互；
2. Spark本身是不能感知Cluster Manager的；

Spark Standalone Mode

https://spark.apache.org/docs/1.0.1/spark-standalone.html
不通过Yarn或者Mesos管理的标准集群。

在Standalone Mode下存在Master和Slaver两类节点，Master的作用类似其他Mode中的Cluster Manager，Client通过Master暴露的7077端口提交作业。通过Zookeeper可以实现对Master节点的高可用配置。

Standalone模式下的架构图：



要提交任务到标准集群时可以使用如下方式

spark-submit --class --master

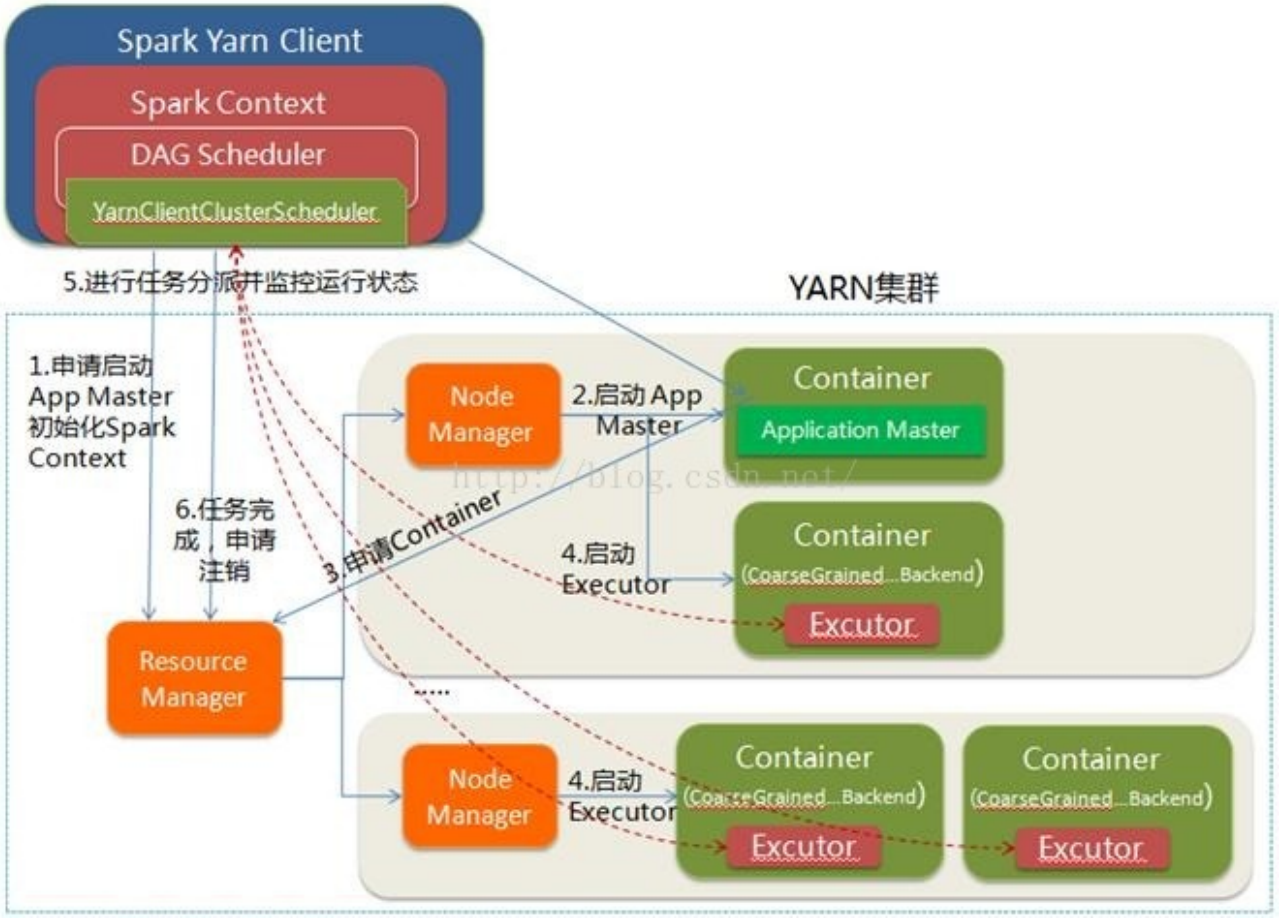
例如：

spark-submit --class com.ruijie.idata.spark.SparkDemo --master spark://172.24.10.63:7077 ruijie-idata-spark-1.0-SNAPSHOT.jar

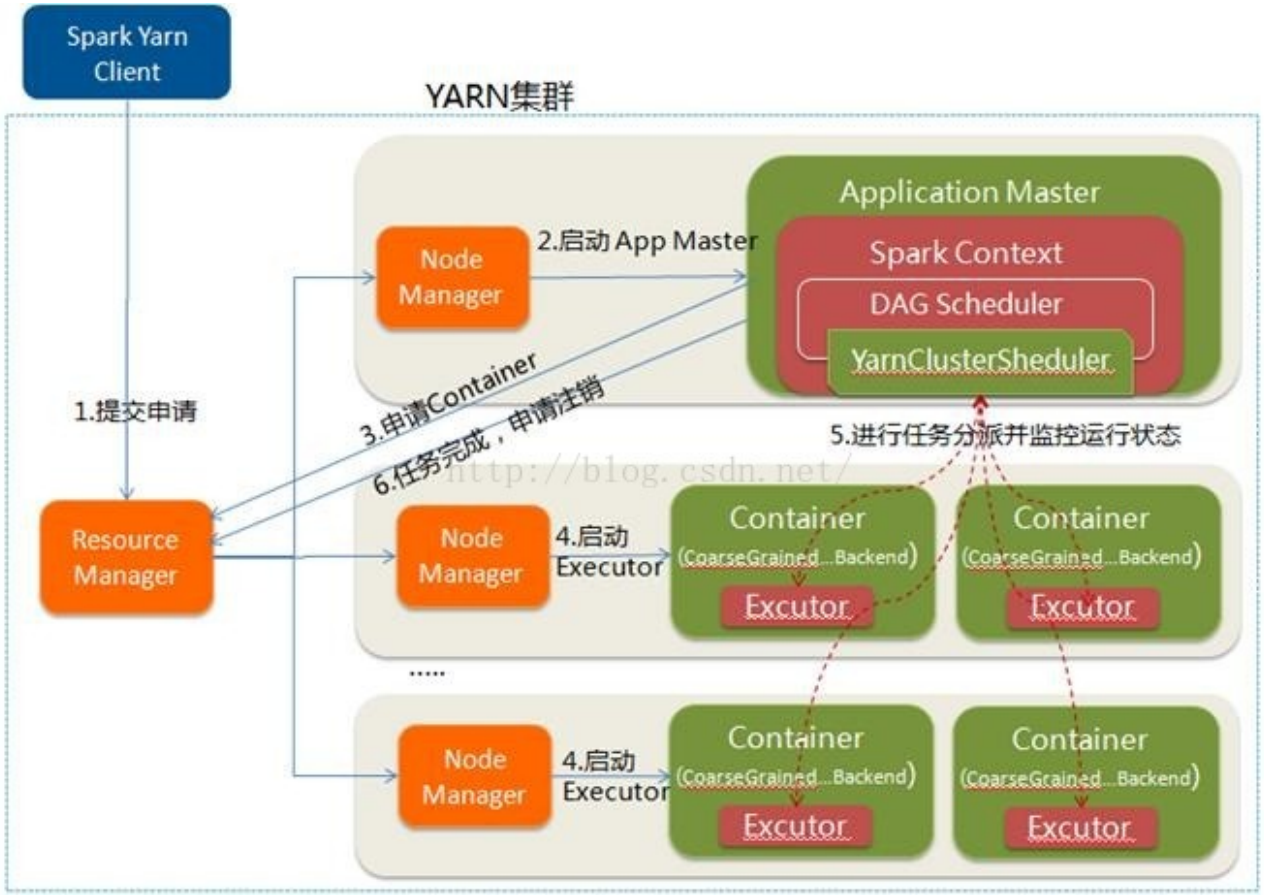
---

Running Spark on YARN

Yarn-Client模式基本架构：



Yarn-Cluster模式基本架构：

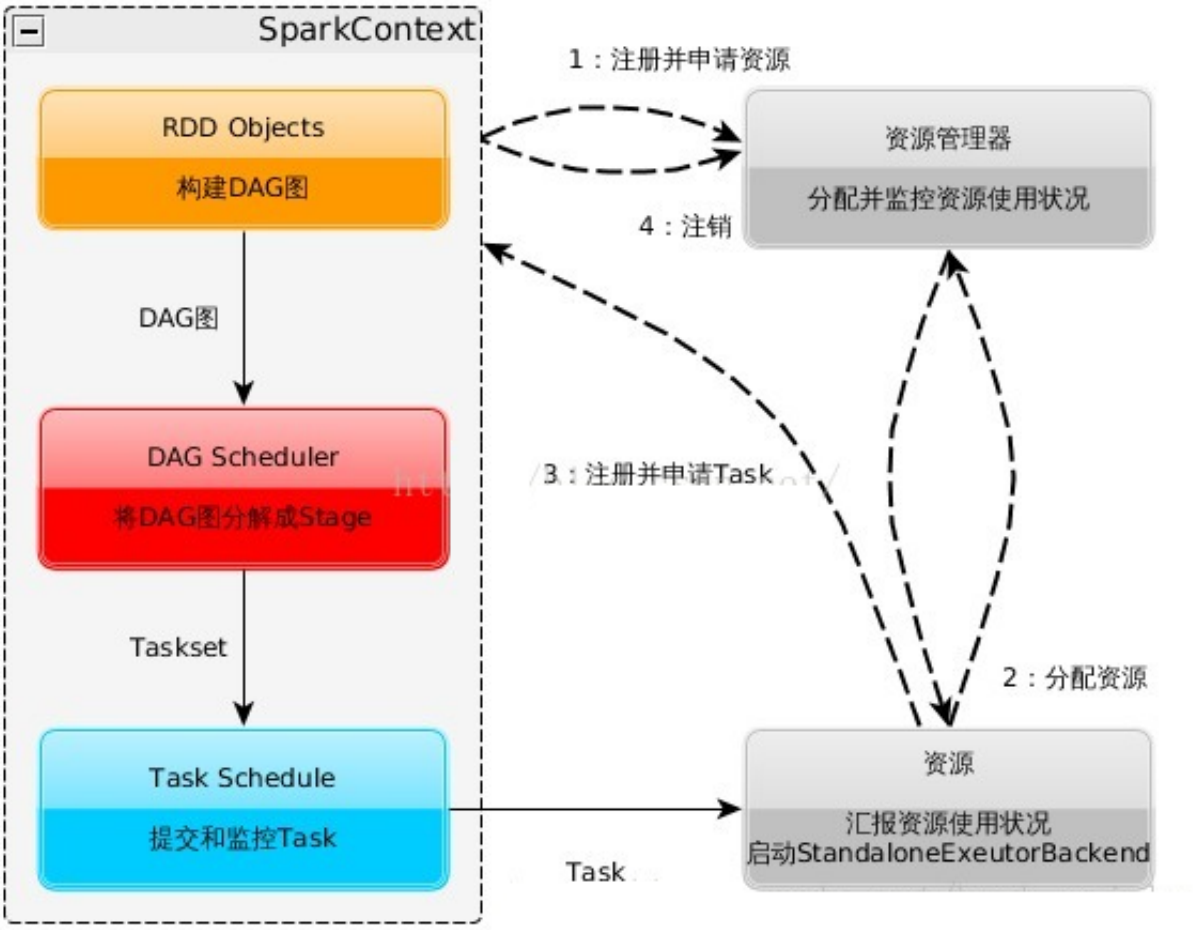


1. 运行在Yarn上的Spark是编译成Yarn支持版本的（参考），

2. Spark的jar包可以放在node节点本地，也可以放在HDFS存储上（通过配置项SPARK_JAR）
3. 提交任务时，需要配置环境变量HADOOP_CONF_DIR或者YARN_CONF_DIR指向到yarn和hdfs的配置文件
4. 两种提交任务的模式，yarn-client（Spark driver运行在Client本地）和yarn-cluster（Spark driver运行在yarn上，客户端在初始化完成之后可以随时退出）
5. 在Spark on Yarn模式下NodeManager节点就是worker，而RM就是Cluter Manager

---

Spark架构

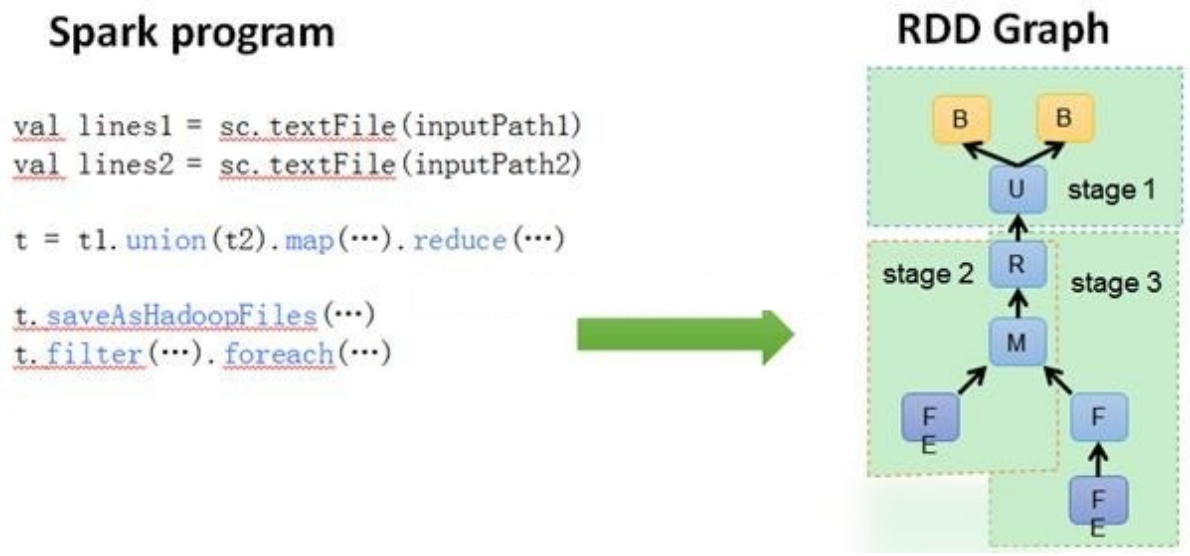


关键术语

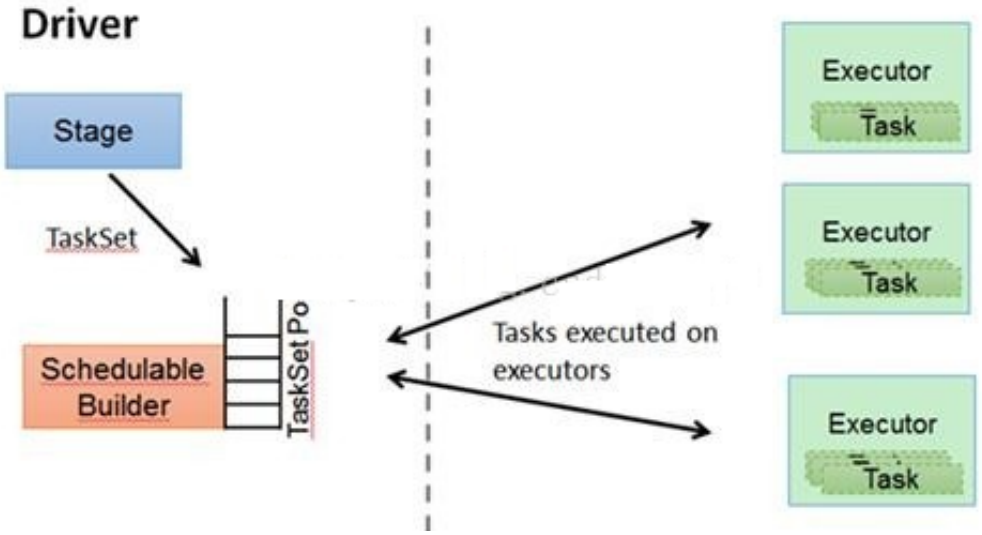
1. Application:包括一个Driver功能的代码(主要是main函数)，和分布在集群中多个节点上运行的Executor代码（可以是原生的RDD接口，也可以是自己定义的？从

代码上看可能是一个StandaloneExecutorBackend或者CoarseGrainedExecutorBackend）

2. Worker、Cluter Manager
3. Job --> Stage(TaskSet) --> Task(被送到Work节点上执行，一个Executor进程下可能管理了多个Task线程)
4. DAGScheduler：作用是将DAG图划分成若干Stage



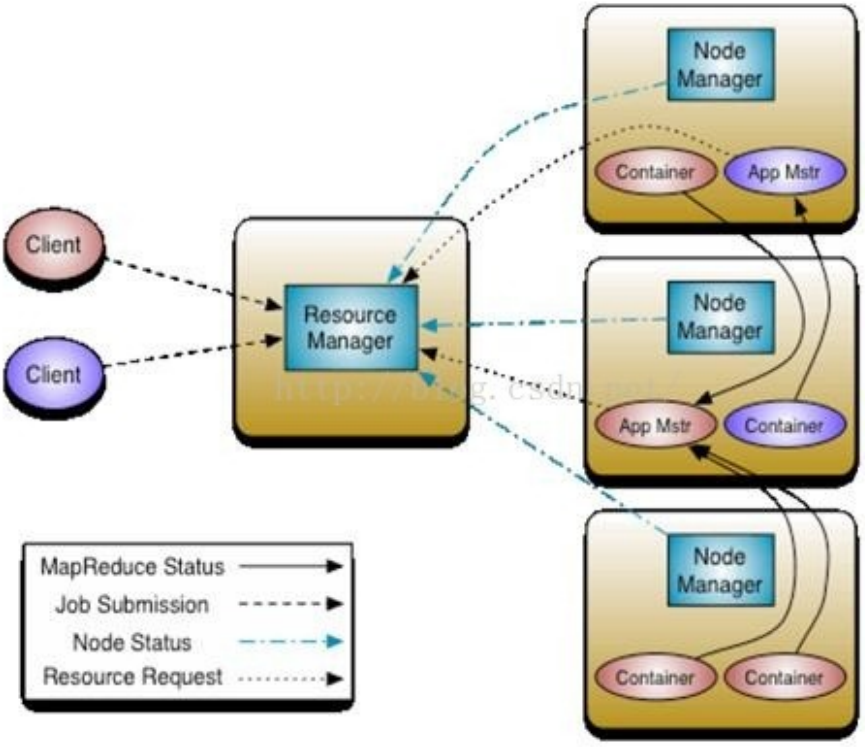
5. TASKSedulter：运行在Driver上，负责调度Task在哪个Work上运行，并且管理Task状态，进行Task的失败重试。在不同运行模式中任务调度器具体为：TaskScheduler（Standalone）、YarnClientClusterScheduler（YARN-Client）、YarnClusterScheduler（YARN-Cluster）



参考

配置项

yarn的基本调度模式：需要注意的一点是APP Master是进程的第一个Container，负责和后续通信的Container进行通信，对于Spark任务来说，Yarn-Cluster模式下Driver运行在AppMaster中，而Yarn-client模式下，Driver运行在客户端上和AppMaster进行通讯。



Spark资源动态调整

参考

```
spark.dynamicAllocation.enabled true    #开启动态配置
```

相关配置

参考

External Shuffle Service

Spark系统在运行含shuffle过程的应用时，Executor进程除了运行task，还要负责写shuffle 数据，给其他Executor提供shuffle数据。

当Executor进程任务过重，导致GC而不能为其他Executor提供shuffle数据时，会影响任务运行。

利用Yarn的External Shuffle Service可以提升Spark APP的提升性能，External shuffle Service是长期存在于NodeManager进程中的一个辅助服务。

通过该服务 来抓取shuffle数据，减少了Executor的压力，在Executor GC的时候也不会影响其他 Executor的任务运行。

启用方法：

在NodeManager中启动External shuffle Service。

```
<!--  在“yarn-site.xml”中添加如下配置项 -->
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>spark_shuffle</value>
</property>
<property>
    <name>yarn.nodemanager.aux-services.spark_shuffle.class</name>
    <value>org.apache.spark.network.yarn.YarnShuffleService</value>
</property>
<property>
    <name>spark.shuffle.service.port</name>
    <value>7337</value>
</property>
```

在Yarn上部署相应服务，参考。

Spark应用使用External shuffle Service。

```
在“spark-defaults.conf”中必须添加如下配置项：
spark.shuffle.service.enabled true
spark.shuffle.service.port 7337
```

说明

1. 如果“yarn.nodemanager.aux-services”配置项已存在，则在value中添加“spark_shuffle”，且用逗号和其他值分开。
2. “spark.shuffle.service.port”的值需要和上面“yarn-site.xml”中的值一样。

调优

- Driver的Core和Mem都可以通过Spark2-submit来配置
- Executor的Core和Mem都可以通过Spark2-submit来配置
- 如果开启了动态调整后Executor的数量会根据被阻塞的Task数目变换；
- 每个Stage创建一批task，task分配到各个Executor进程中执行。task是最小的计算单元，负责执行一模一样的计算逻辑，只是每个task处理的数据不同而已；
- Spark是根据shuffle类算子来进行stage的划分；
- Stage的所有task都执行完毕之后，会在各个节点本地的磁盘文件中写入计算中间结果，然后Driver就会调度运行下一个stage。下一个stage的task的输入数据就是上一个stage输出的中间结果；
- 在代码中执行了cache/persist等持久化操作时，根据我们选择的持久化级别的不同，每个task计算出来的数据也会保存到Executor进程的内存或者所在节点的磁盘文件中；
- Executor的内存分配：
  - 第一块是让task执行编写的代码时使用，默认是占Executor总内存的20%；
  - 第二块是让task通过shuffle过程拉取了上一个stage的task的输出后，进行聚合等操作时使用，默认也是占Executor总内存的20%；
  - 第三块是让RDD持久化时使用，默认占Executor总内存的60%；
- task的执行速度是跟每个Executor进程的CPU core数量有直接关系的。基本上一个CPU对应一个Task线程；

调优参数：

num-executors

参数说明：该参数用于设置Spark作业总共要用多少个Executor进程来执行。Driver在向YARN集群管理器申请资源时，YARN集群管理器会尽可能按照你的设置来在集群的各个工作节点上，启动相应数量的Executor进程。这个参数非常之重要，如果不设置的话，默认只会给你启动少量的Executor进程，此时你的Spark作业的运行速度是非常慢的。

参数调优建议：每个Spark作业的运行一般设置50~100个左右的Executor进程比较合适，设置太少或太多的Executor进程都不好。设置的太少，无法充分利用集群资源；设置的太多的话，大部分队列可能无法给予充分的资源。

executor-memory

参数说明：该参数用于设置每个Executor进程的内存。Executor内存的大小，很多时候直接决定了Spark作业的性能，而且跟常见的JVM OOM异常，也有直接的关联。

参数调优建议：每个Executor进程的**内存设置4G~8G**较为合适。但是这只是一个参考值，具体的设置还是得根据不同部门的资源队列来定。可以看看自己团队的资源队列的最大内存限制是多少，num-executors乘以executor-memory，就代表了你的Spark作业申请到的总内存量（也就是所有Executor进程的内存总和），这个量是不能超过队列的最大内存量的。此外，如果你是跟团队里其他人共享这个资源队列，那么申请的总内存量最好不要超过资源队列最大总内存的1/3~1/2，避免你自己的Spark作业占用了队列所有的资源，导致别的同学的作业无法运行。

executor-cores

参数说明：该参数用于设置每个Executor进程的CPU core数量。这个参数决定了每个Executor进程并行执行task线程的能力。因为每个CPU core同一时间只能执行一个task线程，因此每个Executor进程的CPU core数量越多，越能够快速地执行完分配给自己的所有task线程。

参数调优建议：Executor的CPU core数量设置为2~4个较为合适。同样得根据不同部门的资源队列来定，可以看看自己的资源队列的最大CPU core限制是多少，再依据设置的Executor数量，来决定每个Executor进程可以分配到几个CPU core。同样建议，如果是跟他人共享这个队列，那么num-executors * executor-cores不要超过队列总CPU core的1/3~1/2左右比较合适，也是避免影响其他同学的作业运行。

driver-memory

参数说明：该参数用于设置Driver进程的内存。

参数调优建议：Driver的内存通常来说不设置，或者设置1G左右应该就够了。唯一需要注意的一点是，**如果需要使用collect算子将RDD的数据全部拉取到Driver上进行处理，那么必须确保Driver的内存足够大，否则会出现OOM内存溢出的问题。**

spark.default.parallelism

参数说明：该参数用于设置每个stage的默认task数量。这个参数极为重要，如果不设置可能会直接影响你的Spark作业性能。

参数调优建议：Spark作业的默认task数量为500~1000个较为合适。**很多同学常犯的一个错误就是不去设置这个参数，那么此时就会导致Spark自己根据底层HDFS的block数量来设置task的数量，默认是一个HDFS block对应一个task。**通常来说，Spark默认设置的数量是偏少的（比如就几十个task），如果task数量偏少的话，就会导致你前面设置好的Executor的参数都前功尽弃。试想一下，无论你的Executor进程有多少个，内存和CPU有多大，但是task只有1个或者10个，那么90%的Executor进程可能根本就没有task执行，也就是白白浪费了资源！因此Spark官网建议的设置原则是，设置该参数为**num-executors * executor-cores的2~3倍**较为合适，比如Executor的总CPU core数量为300个，那么设置1000个task是可以的，此时可以充分地利用Spark集群的资源。

```
**在流式处理中中，以Kafka为数据源时并行度和Kafka的分区数目直接相关（使用createDirectStream时）。**
```

spark.storage.memoryFraction

参数说明：该参数用于设置RDD持久化数据在Executor内存中能占的比例，默认是0.6。也就是说，默认Executor 60%的内存，可以用来保存持久化的RDD数据。根据你选择的不同的持久化策略，如果内存不够时，可能数据就不会持久化，或者数据会写入磁盘。

参数调优建议：如果Spark作业中，**有较多的RDD持久化操作，该参数的值可以适当提高一些**，保证持久化的数据能够容纳在内存中。避免内存不够缓存所有的数据，导致数据只能写入磁盘中，降低了性能。但是**如果Spark作业中的shuffle类操作比较多，而持久化操作比较少，那么这个参数的值适当降低**一些比较合适。此外，**如果发现作业由于频繁的gc导致运行缓慢（通过spark web ui可以观察到作业的gc耗时），意味着task执行用户代码的内存不够用，那么同样建议调低这个参数的值。**

spark.shuffle.memoryFraction

参数说明：该参数用于设置shuffle过程中一个task拉取到上个stage的task的输出后，进行聚合操作时能够使用的Executor内存的比例，默认是0.2。也就是说，Executor默认只有20%的内存用来进行该操作。**shuffle操作在进行聚合时，如果发现使用的内存超出了这个20%的限制，那么多余的数据就会溢写到磁盘文件中去，此时就会极大地降低性能。**

参数调优建议：如果Spark作业中的RDD持久化操作较少，shuffle操作较多时，建议降低持久化操作的内存占比，提高shuffle操作的内存占比比例，避免shuffle过程

中数据过多时内存不够用，必须溢写到磁盘上，降低了性能。此外，如果发现作业由于频繁的gc导致运行缓慢，意味着task执行用户代码的内存不够用，那么同样建议调低这个参数的值。