#### 指定固定的executor数

spark-submit \ --master yarn-cluster \

--deploy-mode cluster \ #集群运行模式

--name wordcount\_${date} \ #作业名

--queue production.group.yanghao \ #指定队列

--conf spark.default.parallelism=1000 \ #并行度，shuffle后的默认partition数

--conf spark.network.timeout=1800s \

--conf spark.yarn.executor.memoryOverhead=1024 \ #堆外内存

--conf spark.scheduler.executorTaskBlacklistTime=30000 \

--conf spark.core.connection.ack.wait.timeout=300s \

--num-executors 200 \ #executor数目

--executor-memory 4G \ #executor中堆的内存

--executor-cores 2 \ #executor执行core的数目，设置大于1

--driver-memory 2G \ #driver内存，不用过大

--class ${main\_class} \ #主类

${jar\_path} \ #jar包位置 param\_list \ #mainClass接收的参数列表

#### 动态调整executor数目

spark-submit \ --master yarn-cluster \

--deploy-mode cluster \

--name wordcount\_${date} \

--queue production.group.yanghao \

--conf spark.dynamicAllocation.enabled=true \ #开启动态分配

--conf spark.shuffle.service.enabled=true \ #shuffle service，可以保证executor被删除时，shuffle file被保留

--conf spark.dynamicAllocation.minExecutors=200 \ #最小的executor数目

--conf spark.dynamicAllocation.maxExecutors=500 \ #最大的executor数目

--class ${main\_class} \

${jar\_path} \ param\_list

## yarn client模式

spark-shell \ --master yarn-client \

--queue production.group.yanghao \ #指定队列

--num-executors 200 \ #executor数目

--executor-memory 4G \ #executor中堆的内存

--executor-cores 2 \ #executor执行core的数目，设置大于1

--driver-memory 2G \ #driver内存，不用过大

--jars ${jar\_path} #jar包位置

---------------------

## yarn cluster模式 vs yarn client模式

yarn cluster模式：spark driver和application master在同一个节点上   
yarn client模式：spark driver和client在同一个节点上，支持shell

1.参数调优

1.1 资源类

1.1.1 num-executors

该参数主要用于设置该应用总共需要多少executors来执行，Driver在向集群资源管理器申请资源时需要根据此参数决定分配的Executor个数，并尽量满足所需。在不带的情况下只会分配少量Executor。

spark-submit时若带了–num-executors参数则取此值, 不带时取自spark.executor.instances配置，若没配置则取环境变量SPARK\_EXECUTOR\_INSTANCES的值，若其未设置，则取默认值DEFAULT\_NUMBER\_EXECUTORS=2。

### 1.1.2 executor-memory

设置每个executor的内存，对Spark作业运行的性能影响很大。一般4-8G就差不多了，当然还要看资源队列的情况。num-executor\*executor-memory的大小绝不能超过队列的内存总大小。

1.1.3 executor-cores

设置每个executor的cpu核数，其决定了每个executor并行执行task的能力。Executor的CPU core数量设置为2-4个即可。但要注意，num-executor\*executor-cores也不能超过分配队列中cpu核数的大小。具体的核数的设置需要根据分配队列中资源统筹考虑，取得Executor，核数，及任务数的平衡。对于多任务共享的队列，更要注意不能将资源占满。

### 1.1.4 driver-memory

运行sparkContext的Driver所在所占用的内存，通常不必设置，设置的话1G就足够了，除非是需要使用collect之类算子经常需要将数据提取到driver中的情况。

1.1.5 spark.default.parallelism

此参数用于设置每个stage经TaskScheduler进行调度时生成task的数量，此参数未设置时将会根据读到的RDD的分区生成task，即根据源数据在hdfs中的分区数确定，若此分区数较小，则处理时只有少量task在处理，前述分配的executor中的core大部分无任务可干。

通常可将此值设置为num-executors\*executor-cores的2-3倍为宜，如果与其相近的话，则对于先完成task的core则无任务可干。2-3倍数量关系的话即不至于太零散，又可是的任务执行更均衡。

# 参数调优6-Memory Management内存管理

spark.memory.fraction

执行内存和缓存内存（堆）占jvm总内存的比例，剩余的部分是spark留给用户存储内部源数据、数据结构、异常大的结果数据。

默认值0.6，调小会导致频繁gc，调大容易造成oom。

spark.memory.storageFraction

用于存储的内存在堆中的占比，默认0.5。调大会导致执行内存过小，执行数据落盘，影响效率；调小会导致缓存内存不够，缓存到磁盘上去，影响效率。

值得一提的是在spark中，执行内存和缓存内存公用java堆，当执行内存没有使用时，会动态分配给缓存内存使用，反之也是这样。如果执行内存不够用，可以将存储内存释放移动到磁盘上（最多释放不能超过本参数划分的比例），但存储内存不能把执行内存抢走。

spark.memory.offHeap.enabled

是否允许使用堆外内存来进行某些操作。默认false

spark.memory.offHeap.size

允许使用进行操作的堆外内存的大小，单位bytes 默认0

spark.memory.useLegacyModes

默认false，不开启，在spark1.5之后就被废弃了，下面三个参数也是。。就不做更多的介绍了：

spark.shuffle.memoryFraction

spark.storage.memoryFraction

spark.storage.unrollFraction

spark.storage.replication.proactive

针对失败的executor，主动去cache 有关的RDD中的数据。默认false

spark.cleaner.periodicGC.interval

控制触发gc的频率，默认30min

spark.cleaner.referenceTracking

是否进行context cleaning，默认true

spark.cleaner.referenceTracking.blocking

清理线程是否应该阻止清理任务，默认true

spark.cleaner.referenceTracking.blocking.shuffle

清理线程是否应该阻止shuffle的清理任务，默认false

spark.cleaner.referenceTracking.cleanCheckpoints

清理线程是否应该清理依赖超出范围的检查点文件（checkpoint files不知道怎么翻译。。）默认false

# 参数调优7-Executor behavior

spark.broadcast.blockSize

TorrentBroadcastFactory中的每一个block大小，默认4m

过大会减少广播时的并行度，过小会导致BlockManager 产生 performance hit.

（暂时没懂这是干啥用的）

spark.executor.cores

每个executor的核数，默认yarn下1核，standalone下为所有可用的核。

spark.default.parallelism

默认RDD的分区数、并行数。

像reduceByKey和join等这种需要分布式shuffle的操作中，最大父RDD的分区数；像parallelize之类没有父RDD的操作，则取决于运行环境下得cluster manager：

如果为单机模式，本机核数；集群模式为所有executor总核数与2中最大的一个。

spark.executor.heartbeatInterval

executor和driver心跳发送间隔，默认10s，必须远远小于spark.network.timeout

spark.files.fetchTimeout

从driver端执行SparkContext.addFile() 抓取添加的文件的超时时间，默认60s

spark.files.useFetchCache

默认true，如果设为true，拉取文件时会在同一个application中本地持久化，被若干个executors共享。这使得当同一个主机下有多个executors时，执行任务效率提高。

spark.files.overwrite

默认false，是否在执行SparkContext.addFile() 添加文件时，覆盖已有的内容有差异的文件。

spark.files.maxPartitionBytes

单partition中最多能容纳的文件大小，单位Bytes 默认134217728 (128 MB)

spark.files.openCostInBytes

小文件合并阈值，小于该参数就会被合并到一个partition内。

默认4194304 (4 MB) 。这个参数在将多个文件放入一个partition时被用到，宁可设置的小一些，因为在partition操作中，小文件肯定会比大文件快。

spark.storage.memoryMapThreshold

从磁盘上读文件时，最小单位不能少于该设定值，默认2m，小于或者接近操作系统的每个page的大小。

# Spark Shuffle FetchFailedException报错解决方案

下面， 主要从shuffle的数据量和处理shuffle数据的分区数两个角度入手。

1. 减少shuffle数据

思考是否可以使用map side join或是broadcast join来规避shuffle的产生。

将不必要的数据在shuffle前进行过滤，比如原始数据有20个字段，只要选取需要的字段进行处理即可，将会减少一定的shuffle数据。

2. SparkSQL和DataFrame的join,group by等操作（提供shuffle并发度）

通过spark.sql.shuffle.partitions控制分区数，默认为200，根据shuffle的量以及计算的复杂度提高这个值。

3. Rdd的join,groupBy,reduceByKey等操作

通过spark.default.parallelism控制shuffle read与reduce处理的分区数，默认为运行任务的core的总数（mesos细粒度模式为8个，local模式为本地的core总数），官方建议为设置成运行任务的core的2-3倍。

4. 提高executor的内存

通过spark.executor.memory适当提高executor的memory值

5. 是否存在数据倾斜的问题

空值是否已经过滤？某个key是否可以单独处理？考虑改变数据的分区规则。