Spark 运行架构

目 录

1 SPARK运行架构 3

1.1 术语定义 3

1.2 SPARK运行基本流程 4

1.2.1 DAGScheduler 5

1.2.2 TaskScheduler 6

1.3 RDD运行原理 7

2 SPARK在不同集群中的运行架构 9

2.1 SPARK ON STANDALONE运行过程 9

2.2 SPARK ON YARN运行过程 10

2.2.1 YARN框架流程 10

2.2.2 YARN-Client 11

2.2.3 YARN-Cluster 13

2.2.4 YARN-Client 与 YARN-Cluster 区别 14

3 SPARK在不同集群中的运行演示 15

3.1 STANDALONE运行过程演示 15

3.1.1 查看测试文件存放位置 15

3.1.2 启动Spark-Shell 16

3.1.3 运行过程及结果分析 16

3.2 YARN-CLIENT运行过程演示 18

3.2.1 启动Spark-Shell 18

3.2.2 运行过程及结果分析 20

3.3 YARN-CLUSTER运行过程演示 21

3.3.1 运行程序 21

3.3.2 运行结果 22

4 问题解决 23

4.1 YARN-CLIENT启动报错 23

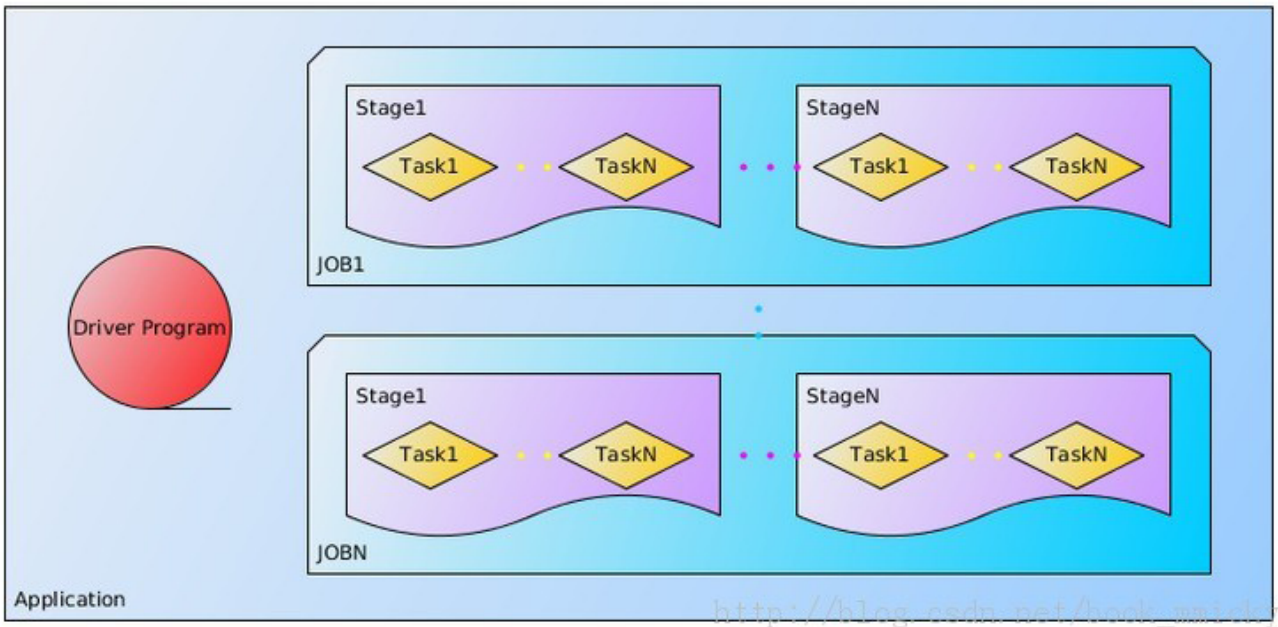
Spark 运行架构

1 Spark 运行架构

1.1 术语定义

- **Application**： Spark Application 的概念和 Hadoop MapReduce 中的类似，指的是用户编写的 Spark 应用程序，包含了一个 Driver 功能的代码和分布在集群中多个节点上运行的 Executor 代码；
- **Driver**： Spark 中的 Driver 即运行上述 Application 的 main()函数并且创建 SparkContext，其中创建 SparkContext 的目的是为了准备 Spark 应用程序的运行环境。在 Spark 中由 SparkContext 负责和 ClusterManager 通信，进行资源的申请、任务的分配和监控等；当 Executor 部分运行完毕后，Driver 负责将 SparkContext 关闭。通常用 SparkContext 代表 Drive；
- **Executor**： Application 运行在 Worker 节点上的一个进程，该进程负责运行 Task，并且负责将数据存在内存或者磁盘上，每个 Application 都有各自独立的一批 Executor。在 Spark on Yarn 模式下，其进程名称为 CoarseGrainedExecutorBackend，类似于 Hadoop MapReduce 中的 YarnChild。一个 CoarseGrainedExecutorBackend 进程有且仅有一个 executor 对象，它负责将 Task 包装成 taskRunner，并从线程池中抽取出一个空闲线程运行 Task。每个 CoarseGrainedExecutorBackend 能并行运行 Task 的数量就取决于分配给它的 CPU 的个数了；
- **Cluster Manager**：指的是在集群上获取资源的外部服务，目前有：
  - Standalone：Spark 原生的资源管理，由 Master 负责资源的分配；
  - Hadoop Yarn：由 YARN 中的 ResourceManager 负责资源的分配；
- **Worker**：集群中任何可以运行 Application 代码的节点，类似于 YARN 中的 NodeManager 节点。在 Standalone 模式中指的就是通过 Slave 文件配置的 Worker 节点，在 Spark on Yarn 模式中指的就是 NodeManager 节点；
- **作业（Job）**：包含多个 Task 组成的并行计算，往往由 Spark Action 催生，一个 JOB 包含多个 RDD 及作用于相应 RDD 上的各种 Operation；
- **阶段（Stage）**：每个 Job 会被拆分很多组 Task，每组任务被称为 Stage，也可称 TaskSet，一个作业分为多个阶段；

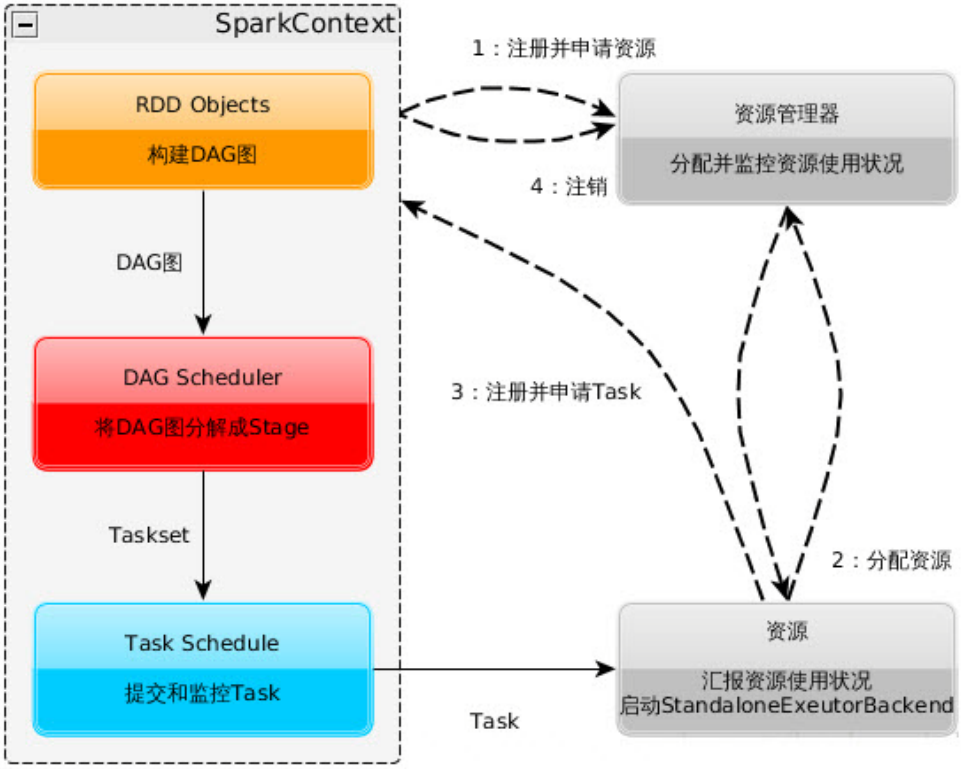
- **任务（Task）**： 被送到某个 Executor 上的工作任务；



1.2 Spark 运行基本流程

Spark 运行基本流程参见下面示意图

1. 构建 Spark Application 的运行环境（启动 SparkContext），SparkContext 向资源管理器（可以是 Standalone、Mesos 或 YARN）注册并申请运行 Executor 资源；
2. 资源管理器分配 Executor 资源并启动 StandaloneExecutorBackend，Executor 运行情况将随着心跳发送到资源管理器上；
3. SparkContext 构建成 DAG 图，将 DAG 图分解成 Stage，并把 Taskset 发送给 Task Scheduler。Executor 向 SparkContext 申请 Task，Task Scheduler 将 Task 发放给 Executor 运行同时 SparkContext 将应用程序代码发放给 Executor。
4. Task 在 Executor 上运行，运行完毕释放所有资源。

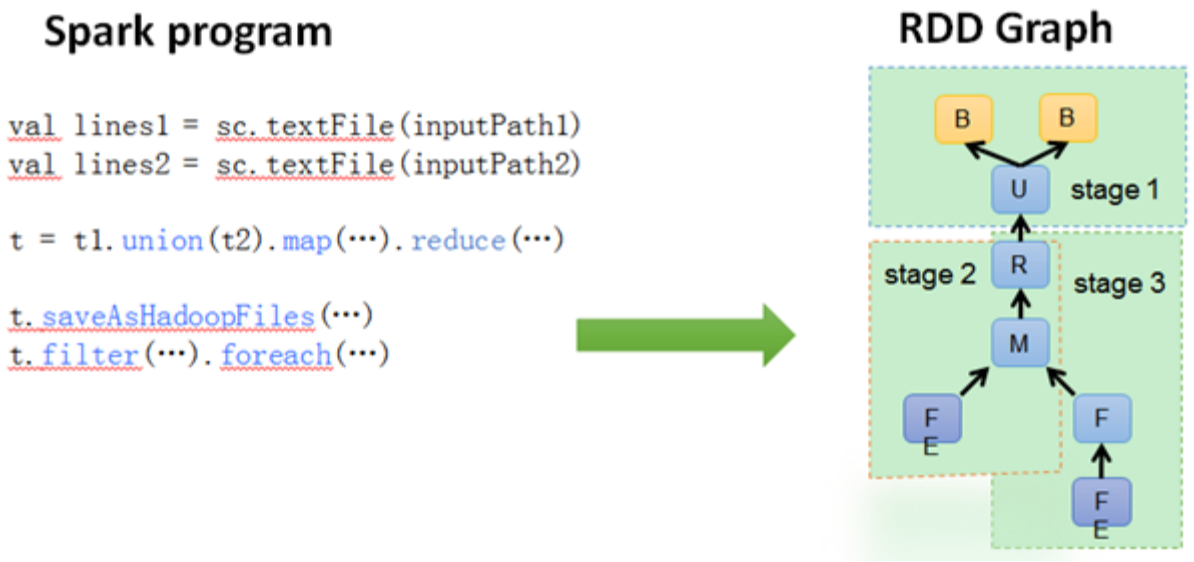


Spark 运行架构特点：

- 每个 Application 获取专属的 executor 进程，该进程在 Application 期间一直驻留，并以多线程方式运行 tasks。这种 Application 隔离机制有其优势的，无论是从调度角度看（每个 Driver 调度它自己的任务），还是从运行角度看（来自不同 Application 的 Task 运行在不同的 JVM 中）。当然，这也意味着 Spark Application 不能跨应用程序共享数据，除非将数据写入到外部存储系统。
- Spark 与资源管理器无关，只要能够获取 executor 进程，并能保持相互通信就可以了。
- 提交 SparkContext 的 Client 应该靠近 Worker 节点（运行 Executor 的节点），最好是在同一个 Rack 里，因为 Spark Application 运行过程中 SparkContext 和 Executor 之间有大量的信息交换；如果想在远程集群中运行，最好使用 RPC 将 SparkContext 提交给集群，不要远离 Worker 运行 SparkContext。
- Task 采用了数据本地性和推测执行的优化机制。

1.2.1 DAGScheduler

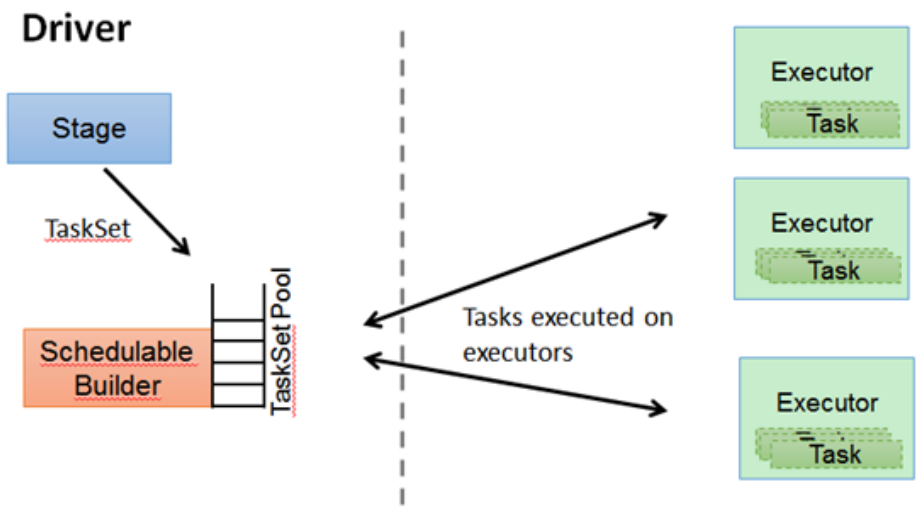
DAGScheduler 把一个 Spark 作业转换成 Stage 的 DAG（Directed Acyclic Graph 有向无环图），根据 RDD 和 Stage 之间的关系找出开销最小的调度方法，然后把 Stage 以 TaskSet 的形式提交给 TaskScheduler，下图展示了 DAGScheduler 的作用：



1.2.2 TaskScheduler

DAGScheduler 决定了运行 Task 的理想位置，并把这些信息传递给下层的 TaskScheduler。此外，DAGScheduler 还处理由于 Shuffle 数据丢失导致的失败，这有可能需要重新提交运行之前的 Stage（非 Shuffle 数据丢失导致的 Task 失败由 TaskScheduler 处理）。

TaskScheduler 维护所有 TaskSet，当 Executor 向 Driver 发送心跳时，TaskScheduler 会根据其资源剩余情况分配相应的 Task。另外 TaskScheduler 还维护着所有 Task 的运行状态，重试失败的 Task。下图展示了 TaskScheduler 的作用：



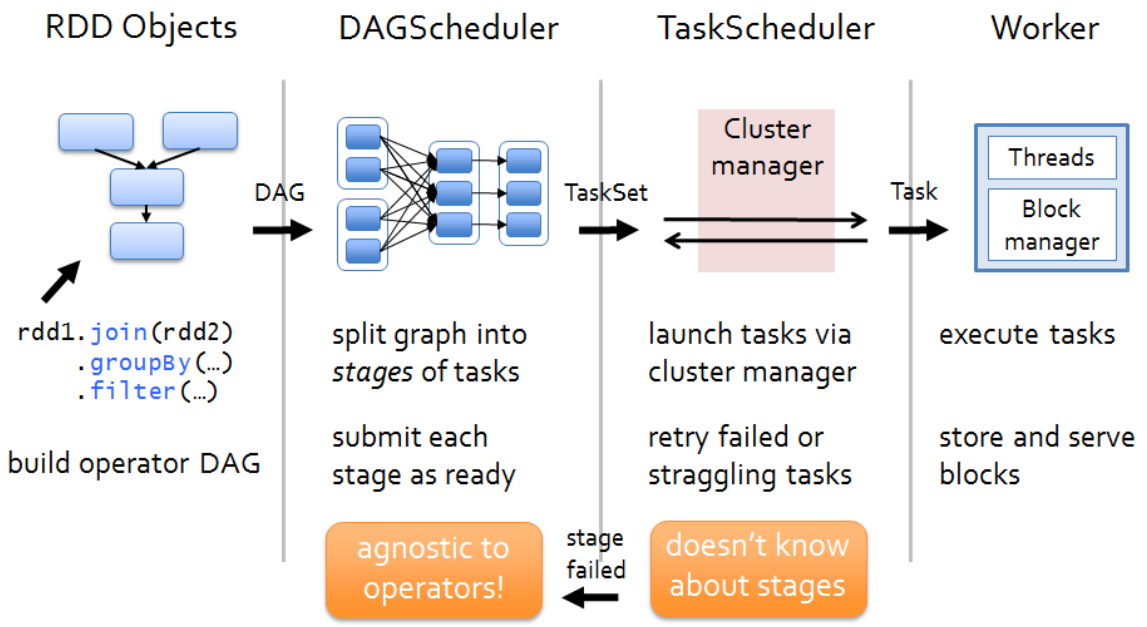
在不同运行模式中任务调度器具体为：

- Spark on Standalone 模式为 TaskScheduler；
- YARN-Client 模式为 YarnClientClusterScheduler
- YARN-Cluster 模式为 YarnClusterScheduler

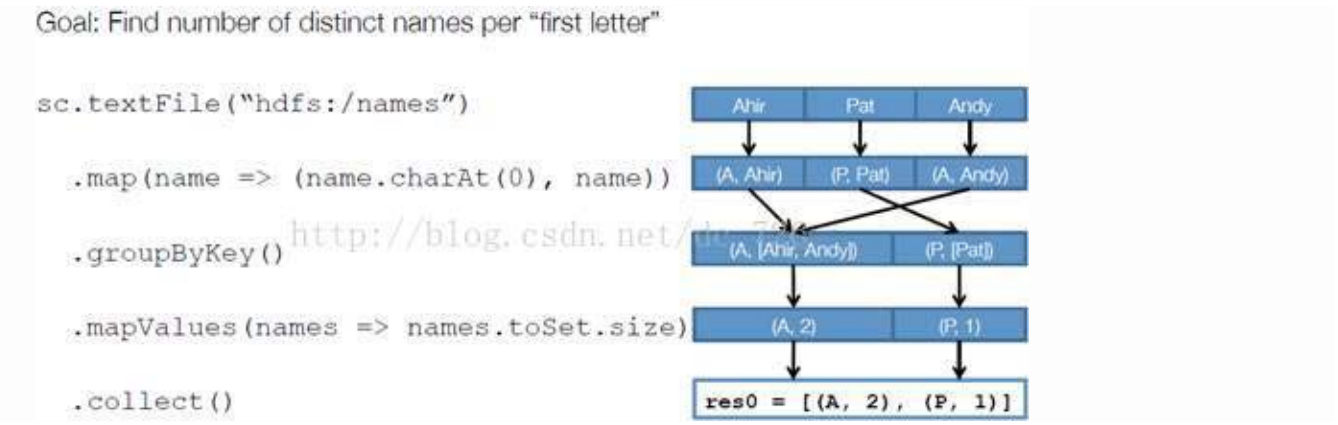
1.3 RDD 运行原理

那么 RDD 在 Spark 架构中是如何运行的呢？总高层次来看，主要分为三步：

1. 创建 RDD 对象
2. DAGScheduler 模块介入运算，计算 RDD 之间的依赖关系。RDD 之间的依赖关系就形成了 DAG
3. 每一个 JOB 被分为多个 Stage ,划分 Stage 的一个主要依据是当前计算因子的输入是否是确定的，如果是则将其分在同一个 Stage，避免多个 Stage 之间的消息传递开销。



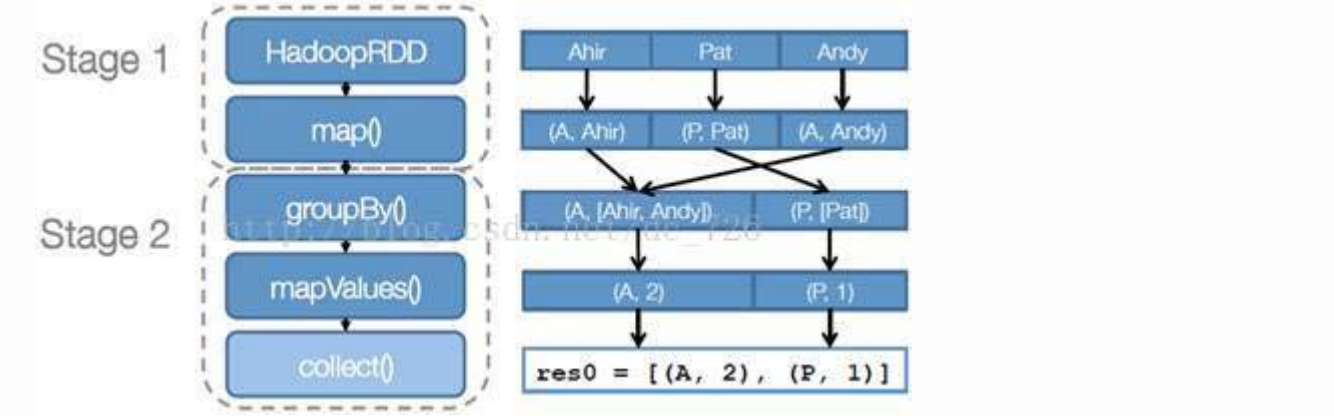
以下面一个按 A-Z 首字母分类 ,查找相同首字母下不同姓名总个数的例子来看一下 RDD 是如何运行起来的。



步骤 1 ：创建 RDD 上面的例子除去最后一个 collect 是个动作，不会创建 RDD 之外，前面四个转换都会创建出新的 RDD 。因此第一步就是创建好所有 RDD(内部的五项信息) 。

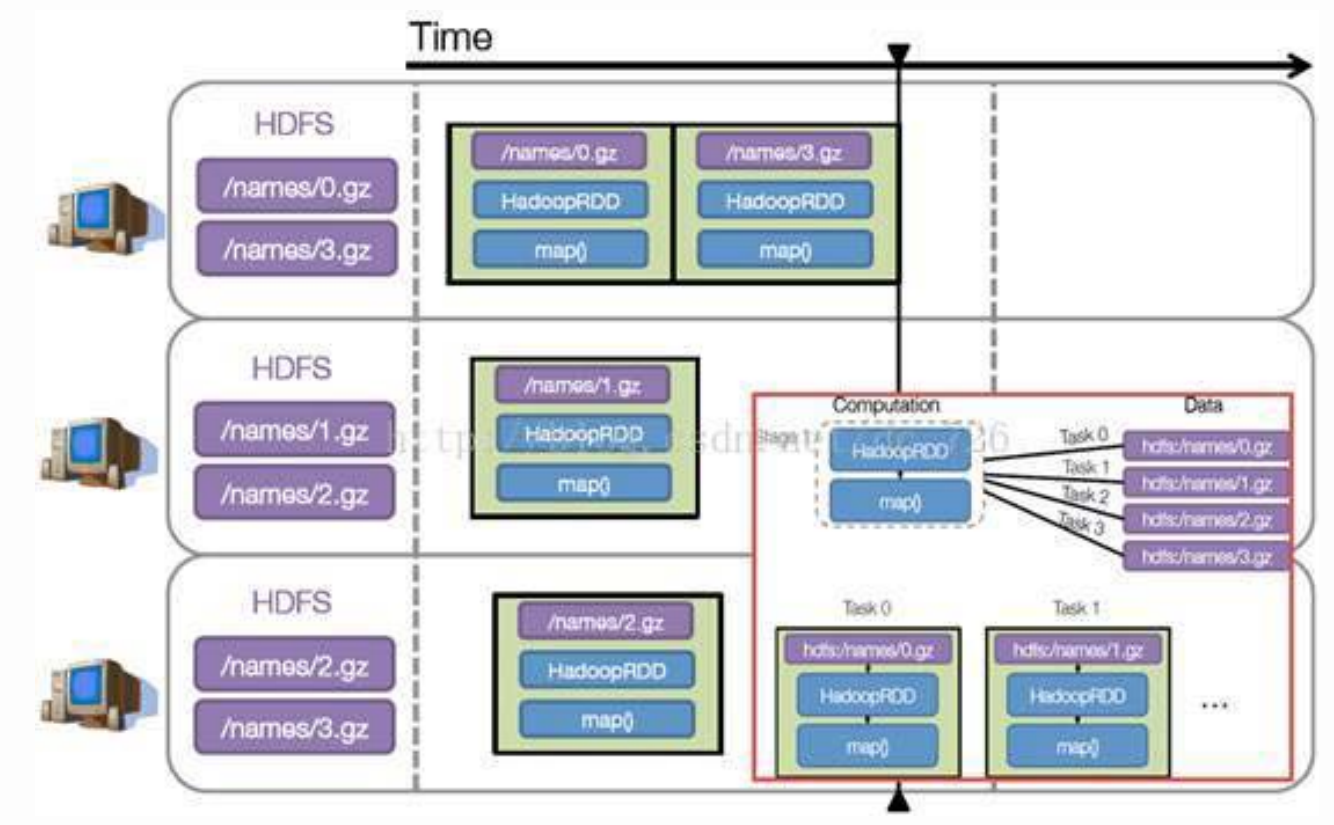
步骤 2 ：创建执行计划 Spark 会尽可能地管道化，并基于是否要重新组织数据来划分 **阶段 (stage)** ，例如本例中的 groupBy() 转换就会将整个执行计划划分成两阶段执行。最终会产

生一个 **DAG(directed acyclic graph ，有向无环图)** 作为逻辑执行计划。



步骤 3 ：调度任务　将各阶段划分成不同的 **任务 (task)** ，每个任务都是数据和计算的合体。在进行下一阶段前，当前阶段的所有任务都要执行完成。因为下一阶段的第一个转换一定是重新组织数据的，所以必须等当前阶段所有结果数据都计算出来了才能继续。

假设本例中的 hdfs://names 下有四个文件块，那么 HadoopRDD 中 partitions 就会有四个分区对应这四个块数据，同时 preferedLocations 会指明这四个块的最佳位置。现在，就可以创建出四个任务，并调度到合适的集群结点上。



2 Spark 在不同集群中的运行架构

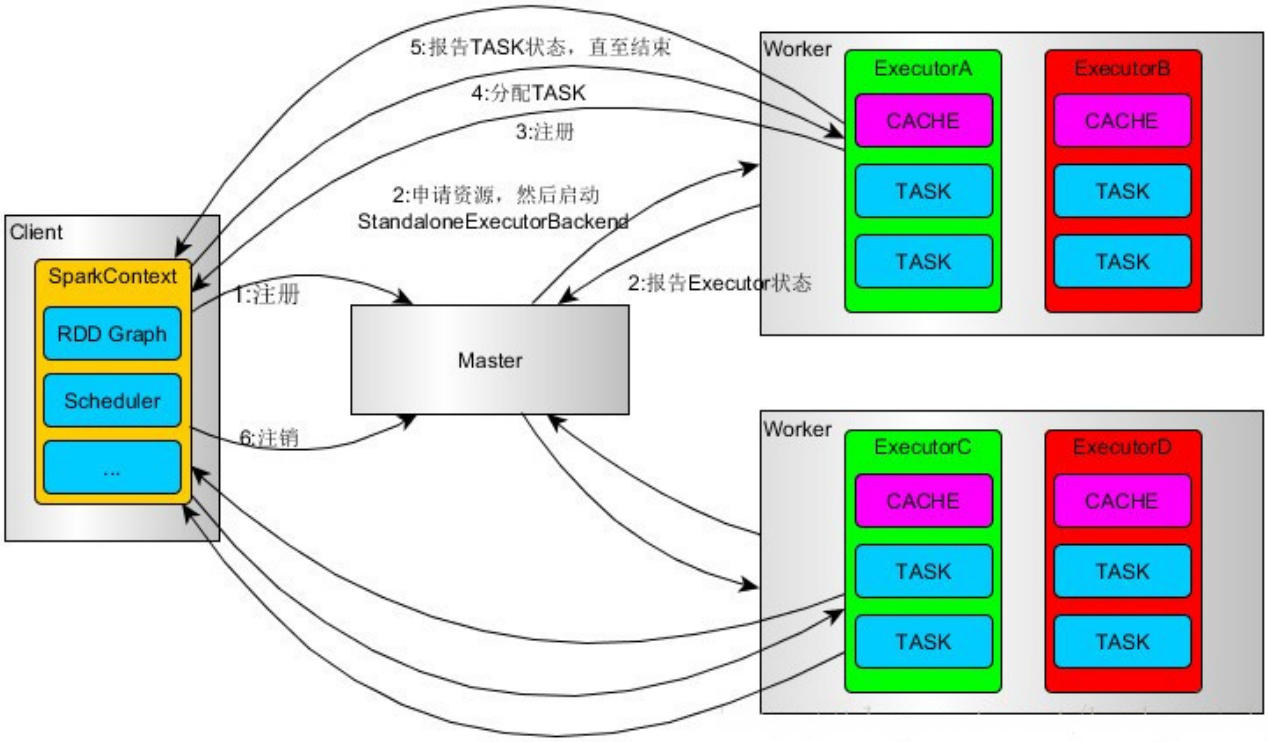
Spark 注重建立良好的生态系统，它不仅支持多种外部文件存储系统，提供了多种多样的集群运行模式。部署在单台机器上时，既可以用本地（Local）模式运行，也可以使用伪分布式模式来运行；当以分布式集群部署的时候，可以根据自己集群的实际情况选择 Standalone 模式（Spark 自带的模式）、YARN-Client 模式或者 YARN-Cluster 模式。Spark 的各种运行模式虽然在启动方式、运行位置、调度策略上各有不同，但它们的目的基本都是一致的，就是在合适的位置安全可靠的根据用户的配置和 Job 的需要运行和管理 Task。

2.1 Spark on Standalone 运行过程

Standalone 模式是 Spark 实现的资源调度框架，其主要的节点有 Client 节点、Master 节点和 Worker 节点。其中 Driver 既可以运行在 Master 节点上中，也可以运行在本地 Client 端。当用 spark-shell 交互式工具提交 Spark 的 Job 时，Driver 在 Master 节点上运行；当使用 spark-submit 工具提交 Job 或者在 Eclips、IDEA 等开发平台上使用" new SparkConf.setManager("spark://master:7077")"方式运行 Spark 任务时，Driver 是运行在本地 Client 端上的。

其运行过程如下：

1. SparkContext 连接到 Master，向 Master 注册并申请资源（CPU Core 和 Memory）；
2. Master 根据 SparkContext 的资源申请要求和 Worker 心跳周期内报告的信息决定在哪个 Worker 上分配资源，然后在该 Worker 上获取资源，然后启动 StandaloneExecutorBackend；
3. StandaloneExecutorBackend 向 SparkContext 注册；
4. SparkContext 将 Applicaiton 代码发送给 StandaloneExecutorBackend；并且 SparkContext 解析 Applicaiton 代码，构建 DAG 图，并提交给 DAG Scheduler 分解成 Stage（当碰到 Action 操作时，就会催生 Job；每个 Job 中含有 1 个或多个 Stage，Stage 一般在获取外部数据和 shuffle 之前产生），然后以 Stage（或者称为 TaskSet）提交给 Task Scheduler，Task Scheduler 负责将 Task 分配到相应的 Worker，最后提交给 StandaloneExecutorBackend 执行；
5. StandaloneExecutorBackend 会建立 Executor 线程池，开始执行 Task，并向 SparkContext 报告，直至 Task 完成。
6. 所有 Task 完成后，SparkContext 向 Master 注销，释放资源。



2.2 Spark on YARN 运行过程

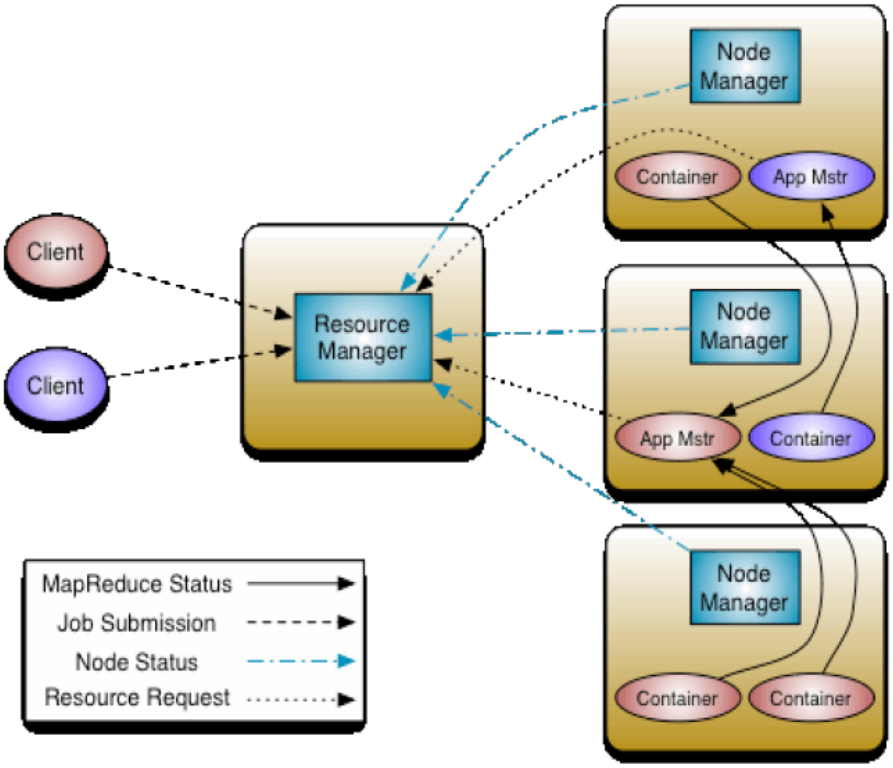
YARN 是一种统一资源管理机制，在其上面可以运行多套计算框架。目前的大数据技术世界，大多数公司除了使用 Spark 来进行数据计算，由于历史原因或者单方面业务处理的性能考虑而使用着其他的计算框架，比如 MapReduce、Storm 等计算框架。Spark 基于此种情况开发了 Spark on YARN 的运行模式，由于借助了 YARN 良好的弹性资源管理机制，不仅部署 Application 更加方便，而且用户在 YARN 集群中运行的服务和 Application 的资源也完全隔离，更具实践应用价值的是 YARN 可以通过队列的方式，管理同时运行在集群中的多个服务。

Spark on YARN 模式根据 Driver 在集群中的位置分为两种模式：一种是 YARN-Client 模式，另一种是 YARN-Cluster（或称为 YARN-Standalone 模式）。

2.2.1 YARN 框架流程

任何框架与 YARN 的结合，都必须遵循 YARN 的开发模式。在分析 Spark on YARN 的实现细节之前，有必要先分析一下 YARN 框架的一些基本原理。

Yarn 框架的基本运行流程图为：

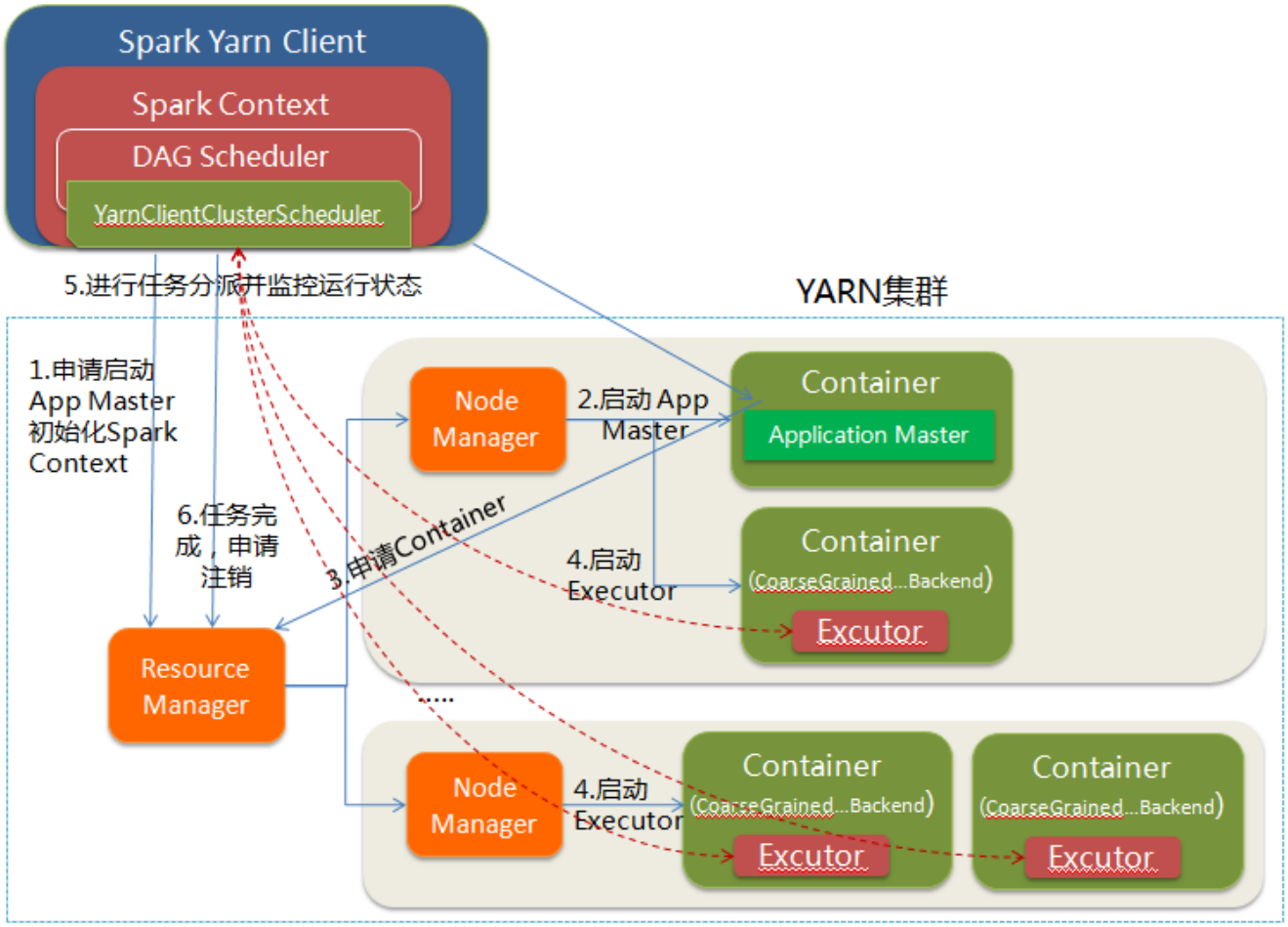


其中，ResourceManager 负责将集群的资源分配给各个应用使用，而资源分配和调度的基本单位是 Container，其中封装了机器资源，如内存、CPU、磁盘和网络等，每个任务会被分配一个 Container，该任务只能在该 Container 中执行，并使用该 Container 封装的资源。NodeManager 是一个个的计算节点，主要负责启动 Application 所需的 Container，监控资源（内存、CPU、磁盘和网络等）的使用情况并将之汇报给 ResourceManager。ResourceManager 与 NodeManagers 共同组成整个数据计算框架，ApplicationMaster 与具体的 Application 相关，主要负责同 ResourceManager 协商以获取合适的 Container，并跟踪这些 Container 的状态和监控其进度。

2.2.2 YARN-Client

Yarn-Client 模式中，Driver 在客户端本地运行，这种模式可以使得 Spark Application 和客户端进行交互，因为 Driver 在客户端，所以可以通过 webUI 访问 Driver 的状态，默认是 http://hadoop1:4040 访问，而 YARN 通过 http:// hadoop1:8088 访问。

YARN-client 的工作流程分为以下几个步骤：



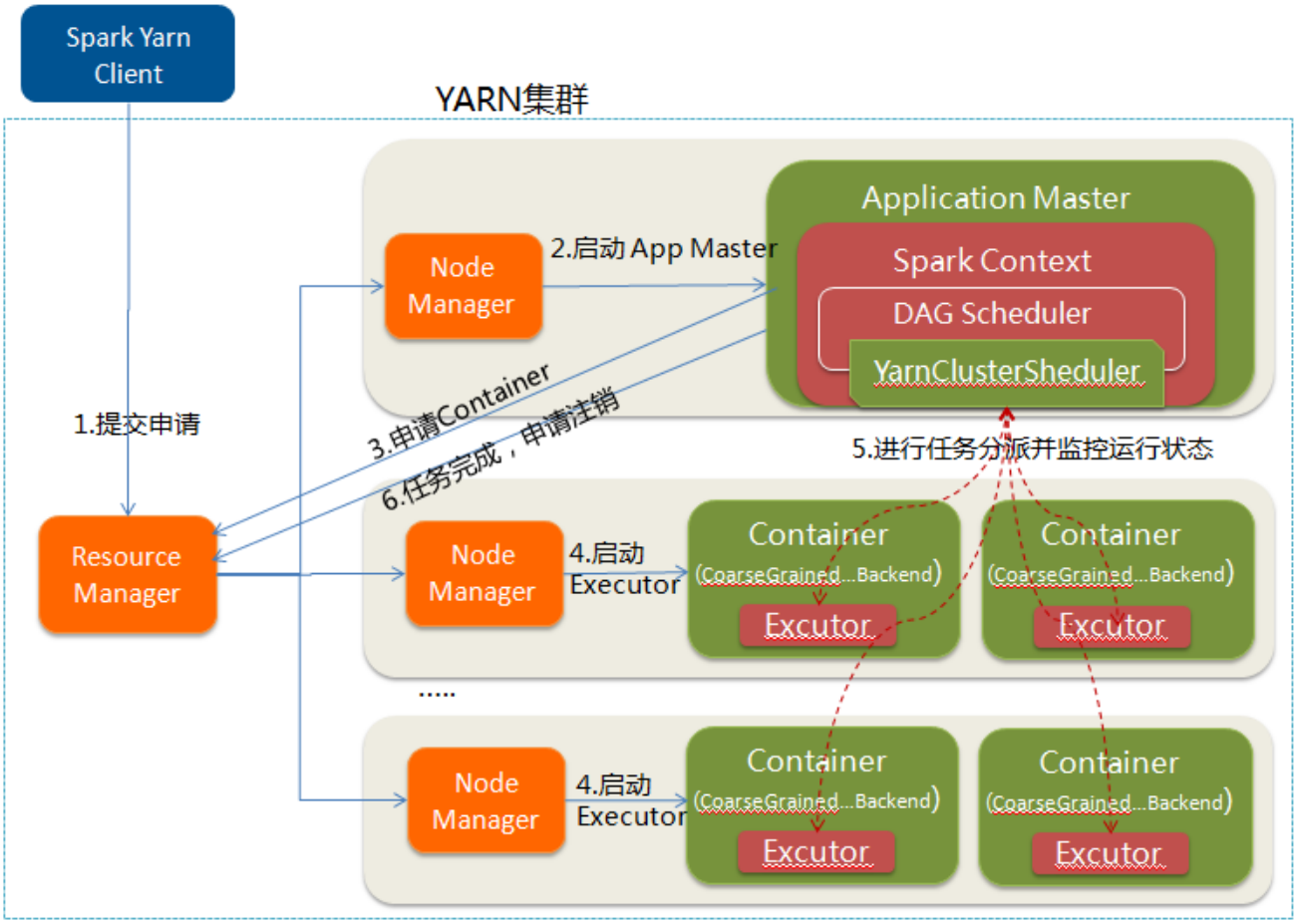
1. Spark Yarn Client 向 YARN 的 ResourceManager 申请启动 Application Master。同时在 SparkContent 初始化中将创建 DAGScheduler 和 TASKScheduler 等，由于我们选择的是 Yarn-Client 模式，程序会选择 YarnClientClusterScheduler 和 YarnClientSchedulerBackend；
2. ResourceManager 收到请求后，在集群中选择一个 NodeManager，为该应用程序分配第一个 Container，要求它在这个 Container 中启动应用程序的 ApplicationMaster，与 YARN-Cluster 区别的是在该 ApplicationMaster 不运行 SparkContext，只与 SparkContext 进行联系进行资源的分派；
3. Client 中的 SparkContext 初始化完毕后，与 ApplicationMaster 建立通讯，向 ResourceManager 注册，根据任务信息向 ResourceManager 申请资源（Container）；
4. 一旦 ApplicationMaster 申请到资源(也就是 Container)后 ,便与对应的 NodeManager 通信，要求它在获得的 Container 中启动启动 CoarseGrainedExecutorBackend，CoarseGrainedExecutorBackend 启动后会向 Client 中的 SparkContext 注册并申请 Task；
5. Client 中的 SparkContext 分配 Task 给 CoarseGrainedExecutorBackend 执行，CoarseGrainedExecutorBackend 运行 Task 并向 Driver 汇报运行的状态和进度，以让 Client 随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务；

6. 应用程序运行完成后，Client 的 SparkContext 向 ResourceManager 申请注销并关闭自己。

2.2.3 YARN-Cluster

在 YARN-Cluster 模式中，当用户向 YARN 中提交一个应用程序后，YARN 将分两个阶段运行该应用程序：第一个阶段是把 Spark 的 Driver 作为一个 ApplicationMaster 在 YARN 集群中先启动；第二个阶段是由 ApplicationMaster 创建应用程序，然后为它向 ResourceManager 申请资源，并启动 Executor 来运行 Task，同时监控它的整个运行过程，直到运行完成。

YARN-cluster 的工作流程分为以下几个步骤：



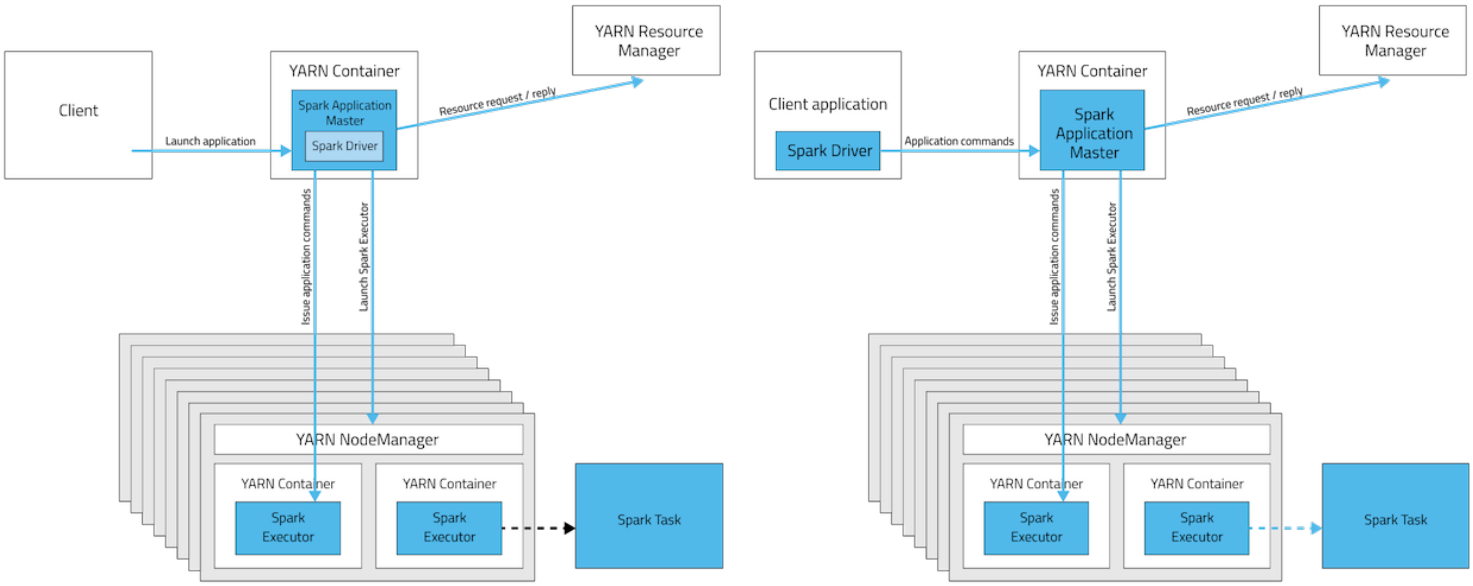
1. Spark Yarn Client 向 YARN 中提交应用程序，包括 ApplicationMaster 程序、启动 ApplicationMaster 的命令、需要在 Executor 中运行的程序等；
2. ResourceManager 收到请求后，在集群中选择一个 NodeManager，为该应用程序分配第一个 Container，要求它在这个 Container 中启动应用程序的 ApplicationMaster，其中 ApplicationMaster 进行 SparkContext 等的初始化；

3. ApplicationMaster 向 ResourceManager 注册，这样用户可以直接通过 ResourceManage 查看应用程序的运行状态，然后它将采用轮询的方式通过 RPC 协议为各个任务申请资源，并监控它们的运行状态直到运行结束；
4. 一旦 ApplicationMaster 申请到资源(也就是 Container)后 ,便与对应的 NodeManager 通信，要求它在获得的 Container 中启动启动 CoarseGrainedExecutorBackend，CoarseGrainedExecutorBackend 启动后会向 ApplicationMaster 中的 SparkContext 注册并申请 Task。这一点和 Standalone 模式一样，只不过 SparkContext 在 Spark Application 中初始化时，使用 CoarseGrainedSchedulerBackend 配合 YarnClusterScheduler 进行任务的调度，其中 YarnClusterScheduler 只是对 TaskSchedulerImpl 的一个简单包装，增加了对 Executor 的等待逻辑等；
5. ApplicationMaster 中的 SparkContext 分配 Task 给 CoarseGrainedExecutorBackend 执行，CoarseGrainedExecutorBackend 运行 Task 并向 ApplicationMaster 汇报运行的状态和进度，以让 ApplicationMaster 随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务；
6. 应用程序运行完成后，ApplicationMaster 向 ResourceManager 申请注销并关闭自己。

2.2.4 YARN-Client 与 YARN-Cluster 区别

理解 YARN-Client 和 YARN-Cluster 深层次的区别之前先清楚一个概念：Application Master。在 YARN 中，每个 Application 实例都有一个 ApplicationMaster 进程，它是 Application 启动的第一个容器。它负责和 ResourceManager 打交道并请求资源，获取资源之后告诉 NodeManager 为其启动 Container。从深层次的含义讲 YARN-Cluster 和 YARN-Client 模式的区别其实就是 ApplicationMaster 进程的区别。

- YARN-Cluster 模式下，Driver 运行在 AM(Application Master)中，它负责向 YARN 申请资源，并监督作业的运行状况。当用户提交了作业之后，就可以关掉 Client，作业会继续在 YARN 上运行，因而 YARN-Cluster 模式不适合运行交互类型的作业；
- YARN-Client 模式下，Application Master 仅仅向 YARN 请求 Executor，Client 会和请求的 Container 通信来调度他们工作，也就是说 Client 不能离开。



3 Spark 在不同集群中的运行演示

在以下运行演示过程中需要启动 Hadoop 和 Spark 集群，其中 Hadoop 需要启动 HDFS 和 YARN，启动过程可以参见第三节《Spark 编程模型（上）--概念及 Shell 试验》。

3.1 Standalone 运行过程演示

在 Spark 集群的节点中，40%的数据用于计算，60%的内存用于保存结果，为了能够直观感受数据在内存和非内存速度的区别，在该演示中将使用大小为 1G 的 Sogou3.txt 数据文件（参见第三节《Spark 编程模型（上）--概念及 Shell 试验》的 3.2 测试数据文件上传），通过对比得到差距。

3.1.1 查看测试文件存放位置

使用 HDFS 命令观察 Sogou3.txt 数据存放节点的位置

$cd /app/hadoop/hadoop-2.2.0/bin
$hdfs fsck /sogou/SogouQ3.txt -files -blocks -locations

通过可以看到该文件被分隔为 9 个块放在集群中

```
[hadoop@hadoop1 upload]$ hdfs fsck /sogou/SogouQ3.txt -files -blocks -locations
Connecting to namenode via http://hadoop1:50070
FSCK started by hadoop (auth:SIMPLE) from /192.168.10.111 for path /sogou/SogouQ3.txt at Tue Jul 14 16:12:12 CST 2015
/sogou/SogouQ3.txt 1086552775 bytes, 9 block(s):  OK
0. BP-335408096-10.88.147.221-1421303552573:blk_1073742213_1389 len=134217728 repl=2 [192.168.10.111:50010, 192.168.10.113:50010]
1. BP-335408096-10.88.147.221-1421303552573:blk_1073742214_1390 len=134217728 repl=2 [192.168.10.111:50010, 192.168.10.113:50010]
2. BP-335408096-10.88.147.221-1421303552573:blk_1073742215_1391 len=134217728 repl=2 [192.168.10.111:50010, 192.168.10.113:50010]
3. BP-335408096-10.88.147.221-1421303552573:blk_1073742216_1392 len=134217728 repl=2 [192.168.10.111:50010, 192.168.10.112:50010]
4. BP-335408096-10.88.147.221-1421303552573:blk_1073742217_1393 len=134217728 repl=2 [192.168.10.111:50010, 192.168.10.112:50010]
5. BP-335408096-10.88.147.221-1421303552573:blk_1073742218_1394 len=134217728 repl=2 [192.168.10.111:50010, 192.168.10.112:50010]
6. BP-335408096-10.88.147.221-1421303552573:blk_1073742219_1395 len=134217728 repl=2 [192.168.10.111:50010, 192.168.10.113:50010]
7. BP-335408096-10.88.147.221-1421303552573:blk_1073742220_1396 len=134217728 repl=2 [192.168.10.111:50010, 192.168.10.113:50010]
8. BP-335408096-10.88.147.221-1421303552573:blk_1073742221_1397 len=12810951 repl=2 [192.168.10.111:50010, 192.168.10.112:50010]

Status: HEALTHY
 Total size:    1086552775 B
 Total dirs:    0
 Total files:   1
 Total symlinks:                0
 Total blocks (validated):      9 (avg. block size 120728086 B)
 Minimally replicated blocks:   9 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    2
 Average block replication:     2.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          3
 Number of racks:               1
FSCK ended at Tue Jul 14 16:12:12 CST 2015 in 29 milliseconds


The filesystem under path '/sogou/SogouQ3.txt' is HEALTHY
```

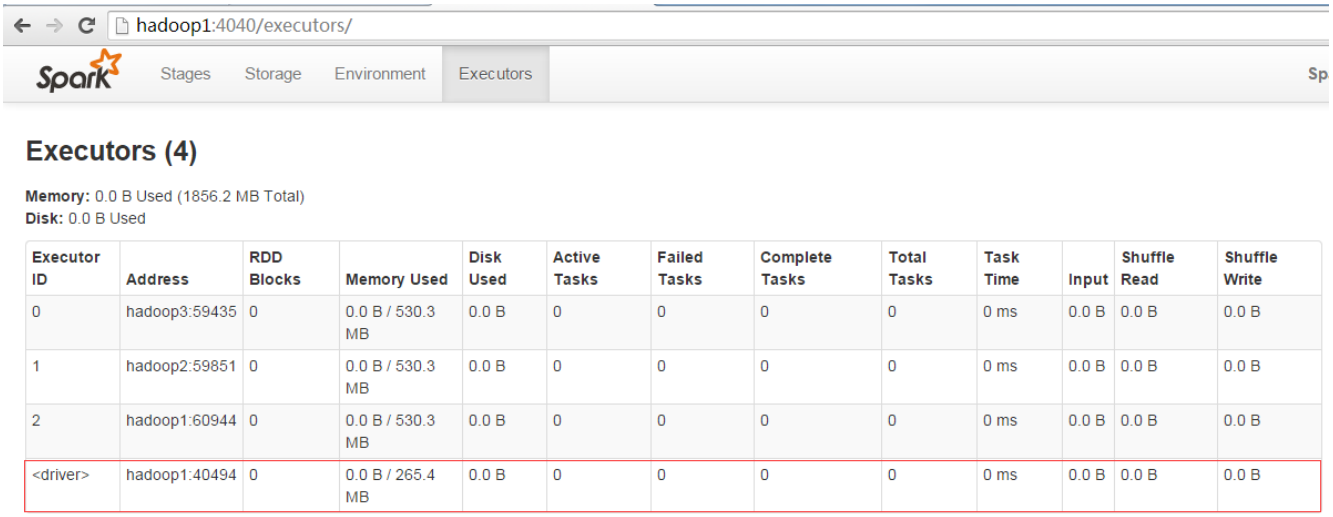
3.1.2 启动 Spark-Shell

通过如下命令启动 Spark-Shell，在演示当中每个 Executor 分配 1G 内存

$cd /app/hadoop/spark-1.1.0/bin

$./spark-shell --master spark://hadoop1:7077 --executor-memory 1g

通过 Spark 的监控界面查看 Executors 的情况，可以观察到有 1 个 Driver 和 3 个 Executor，其中 hadoop2 和 hadoop3 启动一个 Executor，而 hadoop1 启动一个 Executor 和 Driver。在该模式下 Driver 中运行 SparkContect，也就是 DAGSheduler 和 TaskSheduler 等进程是运行在节点上，进行 Stage 和 Task 的分配和管理。



Executor ID	Address	RDD Blocks	Memory Used	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write
0	hadoop3:59435	0	0.0 B / 530.3 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B
1	hadoop2:59851	0	0.0 B / 530.3 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B
2	hadoop1:60944	0	0.0 B / 530.3 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B
<driver>	hadoop1:40494	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B

3.1.3 运行过程及结果分析

第一步 读取文件后计算数据集条数，并计算过程中使用 cache()方法对数据集进行缓存

val sogou=sc.textFile("hdfs://hadoop1:9000/sogou/SogouQ3.txt")

sogou.cache()

sogou.count()

通过页面监控可以看到该作业分为 8 个任务，其中一个任务的数据来源于两个数据分片，其他的任务各对应一个数据分片，即显示 7 个任务获取数据的类型为（NODE_LOCAL），1 个任务获取数据的类型为任何位置（ANY）。

Tasks

Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Accumulators	Input	Errors
0	0	0	SUCCESS	NODE_LOCAL	hadoop1	2015/07/14 16:24:37	1.2 min	27 s		128.0 MB (hadoop)	
3	1	0	SUCCESS	NODE_LOCAL	hadoop2	2015/07/14 16:24:37	2.2 min	1.5 min		128.0 MB (hadoop)	
1	2	0	SUCCESS	NODE_LOCAL	hadoop3	2015/07/14 16:24:37	4.4 min	3.2 min		128.0 MB (hadoop)	
2	3	0	SUCCESS	NODE_LOCAL	hadoop1	2015/07/14 16:25:51	3.9 min	57 s		128.0 MB (hadoop)	
4	4	0	SUCCESS	NODE_LOCAL	hadoop2	2015/07/14 16:26:57	2.6 min	2.2 min		128.0 MB (hadoop)	
6	5	0	SUCCESS	NODE_LOCAL	hadoop3	2015/07/14 16:29:01	1.5 min	58 s		128.0 MB (hadoop)	
5	6	0	SUCCESS	NODE_LOCAL	hadoop2	2015/07/14 16:29:32	14 s	2 s		128.0 MB (hadoop)	
7	7	0	SUCCESS	ANY	hadoop2	2015/07/14 16:29:46	9 s	2 s		140.2 MB (hadoop)	

在存储监控界面中，我们可以看到缓存份数为 3，大小为 907.1M，缓存率为 38%

hadoop1:4040/storage/

Spark　Stages　Storage　Environment　Executors　Spark shell application UI

Storage

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size in Tachyon	Size on Disk
hdfs://hadoop1:9000/sogou/SogouQ3.txt	Memory Deserialized 1x Replicated	3	38%	907.1 MB	0.0 B	0.0 B

运行结果得到数据集的数量为 1000 万笔数据，总共花费了 352.17 秒

```
15/07/14 16:25:51 INFO TaskSetManager: Starting task 2.0 in stage 0.0 (TID 3, hadoop1, NODE_LOCAL, 1193 bytes)
15/07/14 16:25:51 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 74270 ms on hadoop1 (1/8)
15/07/14 16:26:33 WARN BlockManagerMasterActor: Removing BlockManager BlockManagerId(0, hadoop2, 39755, 0) with no recent heart beats: 49062ms exceeds 45000ms
15/07/14 16:26:33 WARN BlockManagerMasterActor: Removing BlockManager BlockManagerId(1, hadoop3, 39749, 0) with no recent heart beats: 69740ms exceeds 45000ms
15/07/14 16:26:51 INFO BlockManagerMasterActor: Registering block manager hadoop2:39755 with 530.3 MB RAM
15/07/14 16:26:51 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on hadoop2:39755 (size: 9.9 KB, free: 530.3 MB)
15/07/14 16:26:51 INFO BlockManagerInfo: Added rdd_1_3 in memory on hadoop2:39755 (size: 299.6 MB, free: 230.7 MB)
15/07/14 16:26:52 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on hadoop2:39755 (size: 1551.0 B, free: 230.7 MB)
15/07/14 16:26:57 INFO TaskSetManager: Starting task 4.0 in stage 0.0 (TID 4, hadoop2, NODE_LOCAL, 1193 bytes)
15/07/14 16:26:57 INFO TaskSetManager: Finished task 3.0 in stage 0.0 (TID 1) in 140043 ms on hadoop2 (2/8)
15/07/14 16:27:36 INFO BlockManagerMasterActor: Registering block manager hadoop3:39749 with 530.3 MB RAM
15/07/14 16:27:36 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on hadoop3:39749 (size: 9.9 KB, free: 530.3 MB)
15/07/14 16:28:33 WARN BlockManagerMasterActor: Removing BlockManager BlockManagerId(1, hadoop3, 39749, 0) with no recent heart beats: 56194ms exceeds 45000ms
15/07/14 16:29:01 INFO BlockManagerMasterActor: Registering block manager hadoop3:39749 with 530.3 MB RAM
15/07/14 16:29:01 INFO TaskSetManager: Starting task 6.0 in stage 0.0 (TID 5, hadoop3, NODE_LOCAL, 1193 bytes)
15/07/14 16:29:01 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 2) in 264050 ms on hadoop3 (3/8)
15/07/14 16:29:01 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on hadoop3:39749 (size: 9.9 KB, free: 530.3 MB)
15/07/14 16:29:01 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on hadoop3:39749 (size: 9.9 KB, free: 530.3 MB)
15/07/14 16:29:01 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on hadoop3:39749 (size: 1551.0 B, free: 530.3 MB)
15/07/14 16:29:01 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on hadoop3:39749 (size: 1551.0 B, free: 530.3 MB)
15/07/14 16:29:01 INFO BlockManagerInfo: Added rdd_1_1 in memory on hadoop3:39749 (size: 304.5 MB, free: 225.7 MB)
15/07/14 16:29:01 INFO BlockManagerInfo: Added rdd_1_1 in memory on hadoop3:39749 (size: 304.5 MB, free: 225.7 MB)
15/07/14 16:29:32 INFO TaskSetManager: Starting task 5.0 in stage 0.0 (TID 6, hadoop2, NODE_LOCAL, 1193 bytes)
15/07/14 16:29:32 INFO TaskSetManager: Finished task 4.0 in stage 0.0 (TID 4) in 155899 ms on hadoop2 (4/8)
15/07/14 16:29:46 INFO TaskSetManager: Starting task 7.0 in stage 0.0 (TID 7, hadoop2, ANY, 1193 bytes)
15/07/14 16:29:46 INFO TaskSetManager: Finished task 5.0 in stage 0.0 (TID 6) in 14205 ms on hadoop2 (5/8)
15/07/14 16:29:46 INFO TaskSetManager: Finished task 2.0 in stage 0.0 (TID 3) in 235719 ms on hadoop1 (6/8)
15/07/14 16:29:55 INFO TaskSetManager: Finished task 7.0 in stage 0.0 (TID 7) in 8871 ms on hadoop2 (7/8)
15/07/14 16:30:29 INFO TaskSetManager: Finished task 6.0 in stage 0.0 (TID 5) in 88109 ms on hadoop3 (8/8)
15/07/14 16:30:29 INFO DAGScheduler: Stage 0 (count at <console>:15) finished in 352.171 s
15/07/14 16:30:29 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
15/07/14 16:30:29 INFO SparkContext: Job finished: count at <console>:15, took 352.670791915 s
res1: Long = 10000000
```

第二步 再次读取文件后计算数据集条数，此次计算使用缓存的数据，对比前后

sogou.count()

通过页面监控可以看到该作业还是分为 8 个任务，其中 3 个任务数据来自内存（PROCESS_LOCAL），3 个任务数据来自本机（NODE_LOCAL），其他 2 个任务数据来自任何位置（ANY）。任务所耗费的时间多少排序为：ANY> NODE_LOCAL> PROCESS_LOCAL，对

比看出使用内存的数据比使用本机或任何位置的速度至少会快 2 个数量级。

Tasks

Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Accumulators	Input	Errors
0	8	0	SUCCESS	PROCESS_LOCAL	hadoop1	2015/07/14 16:38:52	47 ms			303.0 MB (memory)	
1	10	0	SUCCESS	PROCESS_LOCAL	hadoop3	2015/07/14 16:38:52	19 ms			304.5 MB (memory)	
3	9	0	SUCCESS	PROCESS_LOCAL	hadoop2	2015/07/14 16:38:52	68 ms			299.6 MB (memory)	
2	11	0	SUCCESS	NODE_LOCAL	hadoop3	2015/07/14 16:38:56	18 s	4 s		128.0 MB (hadoop)	
4	12	0	SUCCESS	NODE_LOCAL	hadoop2	2015/07/14 16:38:56	9 s	2 s		128.0 MB (hadoop)	
5	13	0	SUCCESS	NODE_LOCAL	hadoop1	2015/07/14 16:38:56	27 s	11 s		128.0 MB (hadoop)	
6	14	0	SUCCESS	ANY	hadoop2	2015/07/14 16:39:05	19 s	2 s		128.0 MB (hadoop)	
7	15	0	SUCCESS	ANY	hadoop3	2015/07/14 16:39:14	12 s	3 s		140.2 MB (hadoop)	

整个作业的运行速度为 34.14 秒，比没有缓存提高了一个数量级。由于刚才例子中数据只是部分缓存（缓存率 38%），如果完全缓存速度能够得到进一步提升，从这体验到 Spark 非常耗内存，不过也够快、够锋利！

```
15/07/14 16:38:52 INFO TaskSetManager: Starting task 0.0 in stage 1.0 (TID 8, hadoop1, PROCESS_LOCAL, 1193 bytes)
15/07/14 16:38:52 INFO TaskSetManager: Starting task 3.0 in stage 1.0 (TID 9, hadoop2, PROCESS_LOCAL, 1193 bytes)
15/07/14 16:38:52 INFO TaskSetManager: Starting task 1.0 in stage 1.0 (TID 10, hadoop3, PROCESS_LOCAL, 1193 bytes)
15/07/14 16:38:52 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on hadoop3:39749 (size: 1551.0 B, free: 225.7 MB)
15/07/14 16:38:52 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on hadoop2:39755 (size: 1551.0 B, free: 230.7 MB)
15/07/14 16:38:52 INFO TaskSetManager: Finished task 1.0 in stage 1.0 (TID 10) in 201 ms on hadoop3 (1/8)
15/07/14 16:38:52 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on hadoop1:46517 (size: 1551.0 B, free: 227.3 MB)
15/07/14 16:38:52 INFO TaskSetManager: Finished task 3.0 in stage 1.0 (TID 9) in 269 ms on hadoop2 (2/8)
15/07/14 16:38:52 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 8) in 297 ms on hadoop1 (3/8)
15/07/14 16:38:56 INFO TaskSetManager: Starting task 2.0 in stage 1.0 (TID 11, hadoop3, NODE_LOCAL, 1193 bytes)
15/07/14 16:38:56 INFO TaskSetManager: Starting task 4.0 in stage 1.0 (TID 12, hadoop2, NODE_LOCAL, 1193 bytes)
15/07/14 16:38:56 INFO TaskSetManager: Starting task 5.0 in stage 1.0 (TID 13, hadoop1, NODE_LOCAL, 1193 bytes)
15/07/14 16:39:05 INFO TaskSetManager: Starting task 6.0 in stage 1.0 (TID 14, hadoop2, ANY, 1193 bytes)
15/07/14 16:39:05 INFO TaskSetManager: Finished task 4.0 in stage 1.0 (TID 12) in 9760 ms on hadoop2 (4/8)
15/07/14 16:39:14 INFO TaskSetManager: Starting task 7.0 in stage 1.0 (TID 15, hadoop3, ANY, 1193 bytes)
15/07/14 16:39:14 INFO TaskSetManager: Finished task 2.0 in stage 1.0 (TID 11) in 18871 ms on hadoop3 (5/8)
15/07/14 16:39:22 INFO TaskSetManager: Finished task 5.0 in stage 1.0 (TID 13) in 26814 ms on hadoop1 (6/8)
15/07/14 16:39:25 INFO TaskSetManager: Finished task 6.0 in stage 1.0 (TID 14) in 19323 ms on hadoop2 (7/8)
15/07/14 16:39:26 INFO TaskSetManager: Finished task 7.0 in stage 1.0 (TID 15) in 12591 ms on hadoop3 (8/8)
15/07/14 16:39:26 INFO DAGScheduler: Stage 1 (count at <console>:15) finished in 34.146 s
15/07/14 16:39:26 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
15/07/14 16:39:26 INFO SparkContext: Job finished: count at <console>:15, took 34.218380494 s
res2: Long = 10000000
```

3.2 YARN-Client 运行过程演示

3.2.1 启动 Spark-Shell

通过如下命令启动 Spark-Shell，在演示当中分配 3 个 Executor、每个 Executor 为 1G 内存

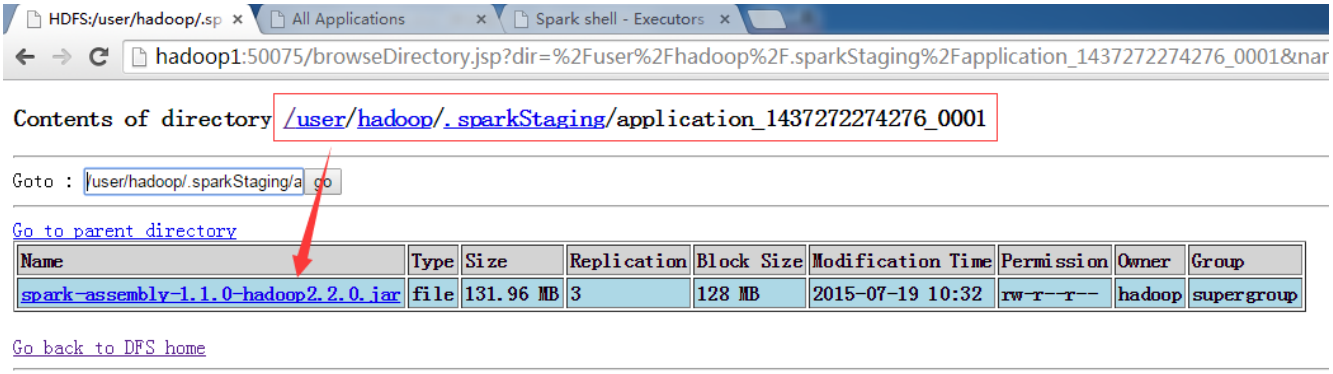
$cd /app/hadoop/spark-1.1.0/bin

$./spark-shell --master YARN-client --num-executors 3 --executor-memory 1g

第一步 把相关的运行 JAR 包上传到 HDFS 中

```
15/07/19 10:32:25 INFO client.RMProxy: Connecting to ResourceManager at hadoop1/192.168.10.111:8032
15/07/19 10:32:25 INFO yarn.Client: Got cluster metric info from ResourceManager, number of NodeManagers: 3
15/07/19 10:32:25 INFO yarn.Client: Max mem capabililty of a single resource in this cluster 8192
15/07/19 10:32:25 INFO yarn.Client: Preparing Local resources
15/07/19 10:32:26 INFO yarn.Client: Uploading file:/app/hadoop/spark-1.1.0/lib/spark-assembly-1.1.0-hadoop2.2.0.jar to hdfs://hadoop1:900
0/user/hadoop/.sparkStaging/application_1437272274276_0001/spark-assembly-1.1.0-hadoop2.2.0.jar
15/07/19 10:32:34 INFO yarn.Client: Prepared Local resources Map(__spark__.jar -> resource { scheme: "hdfs" host: "hadoop1" port: 9000 fi
le: "/user/hadoop/.sparkStaging/application_1437272274276_0001/spark-assembly-1.1.0-hadoop2.2.0.jar" } size: 138366090 timestamp: 1437273
153826 type: FILE visibility: PRIVATE)
15/07/19 10:32:34 INFO yarn.Client: Setting up the launch environment
15/07/19 10:32:34 INFO yarn.Client: Setting up container launch context
15/07/19 10:32:34 INFO yarn.Client: Yarn AM launch context:
```

通过 HDFS 查看界面可以看到在 /user/hadoop/.sparkStaging/应用编号，查看到这些文件：



第二步 启动 Application Master，注册 Executor

应用程序向 ResourceManager 申请启动 Application Master，在启动完成后会分配 Cotainer 并把这些信息反馈给 SparkContext，SparkContext 和相关的 NM 通讯，在获得的 Container 上启动 Executor，从下图可以看到在 hadoop1、hadoop2 和 hadoop3 分别启动了 Executor

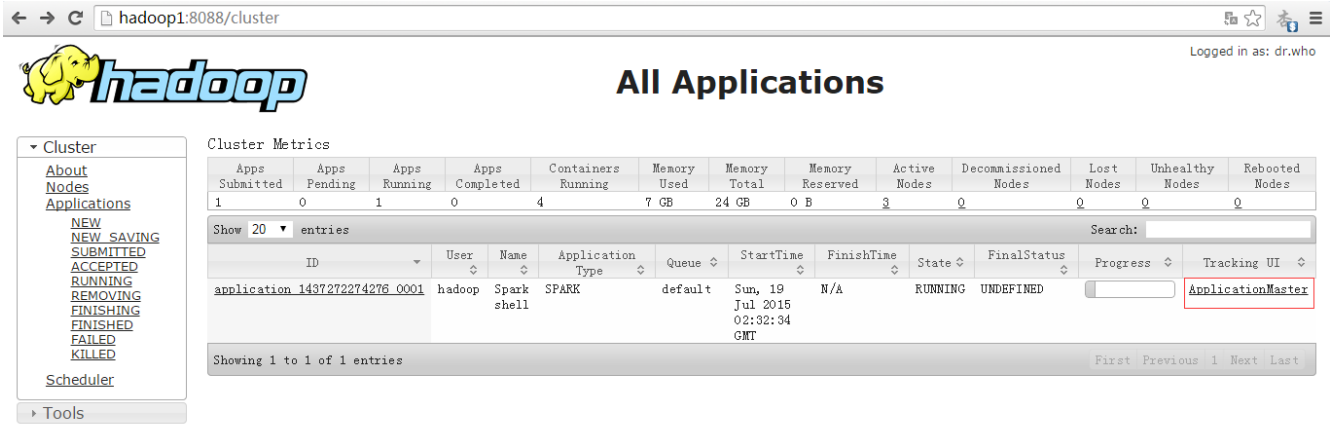
```
15/07/19 10:32:50 INFO cluster.YarnClientSchedulerBackend: Application report from ASM:
         appMasterRpcPort: 0
         appStartTime: 1437273154315
         yarnAppState: RUNNING

15/07/19 10:32:50 INFO cluster.YarnClientSchedulerBackend: Add WebUI Filter. org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter,
PROXY_HOST=hadoop1,PROXY_URI_BASE=http://hadoop1:8088/proxy/application_1437272274276_0001, /proxy/application_1437272274276_0001
15/07/19 10:32:50 INFO ui.JettyUtils: Adding filter: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
15/07/19 10:32:54 INFO cluster.YarnClientSchedulerBackend: SchedulerBackend is ready for scheduling beginning after waiting maxRegistered
ResourcesWaitingTime: 30000(ms)
15/07/19 10:32:54 INFO repl.SparkILoop: Created spark context..
Spark context available as sc.

scala> 15/07/19 10:32:59 INFO cluster.YarnClientSchedulerBackend: Registered executor: Actor[akka.tcp://sparkExecutor@hadoop1:51459/user/
Executor#-1127895451] with ID 3
15/07/19 10:33:00 INFO util.RackResolver: Resolved hadoop1 to /default-rack
15/07/19 10:33:08 INFO cluster.YarnClientSchedulerBackend: Registered executor: Actor[akka.tcp://sparkExecutor@hadoop2:39121/user/Executo
r#-1985657686] with ID 1
15/07/19 10:33:08 INFO util.RackResolver: Resolved hadoop2 to /default-rack
15/07/19 10:33:09 INFO storage.BlockManagerMasterActor: Registering block manager hadoop2:59817 with 530.3 MB RAM
15/07/19 10:33:09 INFO cluster.YarnClientSchedulerBackend: Registered executor: Actor[akka.tcp://sparkExecutor@hadoop3:37713/user/Executo
r#2039207811] with ID 2
15/07/19 10:33:09 INFO util.RackResolver: Resolved hadoop3 to /default-rack
```

第三步 查看启动结果

YARN-Client 模式中，Driver 在客户端本地运行，这种模式可以使得 Spark Application 和客户端进行交互，因为 Driver 在客户端所以可以通过 webUI 访问 Driver 的状态，默认是 http://hadoop1:4040 访问，而 YARN 通过 http:// hadoop1:8088 访问。



hadoop1:8088/proxy/application_1437272274276_0001/executors

Spark　Stages　Storage　Environment　Executors　Sp

Executors (4)

Memory: 0.0 B Used (1856.2 MB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Memory Used	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write
1	hadoop2:59817	0	0.0 B / 530.3 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B
2	hadoop3:57609	0	0.0 B / 530.3 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B
3	hadoop1:42487	0	0.0 B / 530.3 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B
<driver>	hadoop1:41016	0	0.0 B / 265.4 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B

3.2.2 运行过程及结果分析

第一步 读取文件后计算数据集条数，并计算过程中使用 cache()方法对数据集进行缓存

```
val sogou=sc.textFile("hdfs://hadoop1:9000/sogou/SogouQ3.txt")
sogou.cache()
sogou.count()
```

通过页面监控可以看到该作业分为 8 个任务，其中一个任务的数据来源于两个数据分片，其他的任务各对应一个数据分片，即显示 7 个任务获取数据的类型为（NODE_LOCAL），1 个任务获取数据的类型为任何位置（RACK_LOCAL）。

Tasks

Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Accumulators	Input	Errors
3	0	0	SUCCESS	NODE_LOCAL	hadoop2	2015/07/19 10:51:32	18 s	2 s		128.0 MB (hadoop)	
0	1	0	SUCCESS	NODE_LOCAL	hadoop1	2015/07/19 10:51:32	1.7 min	30 s		128.0 MB (hadoop)	
1	2	0	SUCCESS	NODE_LOCAL	hadoop3	2015/07/19 10:51:32	16 s	2 s		128.0 MB (hadoop)	
2	3	0	SUCCESS	NODE_LOCAL	hadoop3	2015/07/19 10:51:49	17 s	2 s		128.0 MB (hadoop)	
4	4	0	SUCCESS	NODE_LOCAL	hadoop2	2015/07/19 10:51:53	13 s	1 s		128.0 MB (hadoop)	
6	5	0	SUCCESS	NODE_LOCAL	hadoop3	2015/07/19 10:52:07	18 s	3 s		128.0 MB (hadoop)	
5	6	0	SUCCESS	NODE_LOCAL	hadoop2	2015/07/19 10:52:15	15 s	3 s		128.0 MB (hadoop)	
7	7	0	SUCCESS	RACK_LOCAL	hadoop3	2015/07/19 10:52:35	21 s	2 s		140.2 MB (hadoop)	

通过运行日志可以观察到在所有任务结束的时候，由 YARNClientScheduler 通知 YARN 集群任务运行完毕，回收资源，最终关闭 SparkContext，整个过程耗费 108.6 秒。

```
15/07/19 10:51:49 INFO scheduler.TaskSetManager: Starting task 2.0 in stage 0.0 (TID 3, hadoop3, NODE_LOCAL, 1193 bytes)
15/07/19 10:51:50 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 0.0 (TID 2) in 17318 ms on hadoop3 (1/8)
15/07/19 10:51:51 INFO storage.BlockManagerInfo: Added rdd_1_3 in memory on hadoop2:59817 (size: 299.6 MB, free: 230.7 MB)
15/07/19 10:51:53 INFO scheduler.TaskSetManager: Starting task 4.0 in stage 0.0 (TID 4, hadoop2, NODE_LOCAL, 1193 bytes)
15/07/19 10:51:53 INFO scheduler.TaskSetManager: Finished task 3.0 in stage 0.0 (TID 0) in 20461 ms on hadoop2 (2/8)
15/07/19 10:52:07 INFO scheduler.TaskSetManager: Starting task 6.0 in stage 0.0 (TID 5, hadoop3, NODE_LOCAL, 1193 bytes)
15/07/19 10:52:07 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 0.0 (TID 3) in 17555 ms on hadoop3 (3/8)
15/07/19 10:52:15 INFO scheduler.TaskSetManager: Starting task 5.0 in stage 0.0 (TID 6, hadoop2, NODE_LOCAL, 1193 bytes)
15/07/19 10:52:15 INFO scheduler.TaskSetManager: Finished task 4.0 in stage 0.0 (TID 4) in 22832 ms on hadoop2 (4/8)
15/07/19 10:52:36 INFO scheduler.TaskSetManager: Starting task 7.0 in stage 0.0 (TID 7, hadoop3, RACK_LOCAL, 1193 bytes)
15/07/19 10:52:36 INFO scheduler.TaskSetManager: Finished task 6.0 in stage 0.0 (TID 5) in 29202 ms on hadoop3 (5/8)
15/07/19 10:52:36 INFO scheduler.TaskSetManager: Finished task 5.0 in stage 0.0 (TID 6) in 20961 ms on hadoop2 (6/8)
15/07/19 10:52:57 INFO scheduler.TaskSetManager: Finished task 7.0 in stage 0.0 (TID 7) in 22097 ms on hadoop3 (7/8)
15/07/19 10:53:16 INFO storage.BlockManagerInfo: Added rdd_1_0 in memory on hadoop1:42487 (size: 303.0 MB, free: 227.3 MB)
15/07/19 10:53:18 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 1) in 106254 ms on hadoop1 (8/8)
15/07/19 10:53:19 INFO scheduler.DAGScheduler: Stage 0 (count at <console>:15) finished in 106.277 s
15/07/19 10:53:19 INFO cluster.YarnClientClusterScheduler: Removed TaskSet 0.0, whose tasks have all completed, from pool
15/07/19 10:53:21 INFO spark.SparkContext: Job finished: count at <console>:15, took 108.598685847 s
res1: Long = 10000000
```

第二步 查看数据缓存情况

通过监控界面可以看到，和 Standalone 一样 38%的数据已经缓存在内存中

Spark Stages Storage Environment Executors Spark shell application UI

Storage

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size in Tachyon	Size on Disk
hdfs://hadoop1:9000/sogou/SogouQ3.txt	Memory Deserialized 1x Replicated	3	38%	907.1 MB	0.0 B	0.0 B

第三步 再次读取文件后计算数据集条数，此次计算使用缓存的数据，对比前后

sogou.count()

通过页面监控可以看到该作业还是分为 8 个任务，其中 3 个任务数据来自内存（PROCESS_LOCAL），4 个任务数据来自本机（NODE_LOCAL），1 个任务数据来自机架（RACK_LOCAL）。对比在内存中的运行速度最快，速度比在本机要快至少 1 个数量级。

Tasks

Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Accumulators	Input	Errors
1	8	0	SUCCESS	PROCESS_LOCAL	hadoop3	2015/07/19 10:56:22	0.2 s			304.5 MB (memory)	
3	9	0	SUCCESS	PROCESS_LOCAL	hadoop2	2015/07/19 10:56:22	0.2 s			299.6 MB (memory)	
0	10	0	SUCCESS	PROCESS_LOCAL	hadoop1	2015/07/19 10:56:22	0.3 s			303.0 MB (memory)	
2	11	0	SUCCESS	NODE_LOCAL	hadoop3	2015/07/19 10:56:25	10 s	3 s		128.0 MB (hadoop)	
4	12	0	SUCCESS	NODE_LOCAL	hadoop1	2015/07/19 10:56:25	21 s	5 s		128.0 MB (hadoop)	
5	13	0	SUCCESS	NODE_LOCAL	hadoop2	2015/07/19 10:56:25	11 s	3 s		128.0 MB (hadoop)	
6	14	0	SUCCESS	NODE_LOCAL	hadoop3	2015/07/19 10:56:35	10 s	1 s		128.0 MB (hadoop)	
7	15	0	SUCCESS	RACK_LOCAL	hadoop2	2015/07/19 10:56:39	13 s	0.4 s		140.2 MB (hadoop)	

YARNClientClusterScheduler 替代了 Standalone 模式下得 TaskScheduler 进行任务管理，在任务结束后通知 YARN 集群进行资源的回收，最后关闭 SparkContect。部分缓存数据运行过程耗费了 29.77 秒，比没有缓存速度提升不少。

```
15/07/19 10:56:22 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 1.0 (TID 8) in 490 ms on hadoop3 (1/8)
15/07/19 10:56:22 INFO scheduler.TaskSetManager: Finished task 3.0 in stage 1.0 (TID 9) in 515 ms on hadoop2 (2/8)
15/07/19 10:56:22 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID 10) in 633 ms on hadoop1 (3/8)
15/07/19 10:56:25 INFO scheduler.TaskSetManager: Starting task 2.0 in stage 1.0 (TID 11, hadoop3, NODE_LOCAL, 1193 bytes)
15/07/19 10:56:25 INFO scheduler.TaskSetManager: Starting task 4.0 in stage 1.0 (TID 12, hadoop1, NODE_LOCAL, 1193 bytes)
15/07/19 10:56:25 INFO scheduler.TaskSetManager: Starting task 5.0 in stage 1.0 (TID 13, hadoop2, NODE_LOCAL, 1193 bytes)
15/07/19 10:56:35 INFO scheduler.TaskSetManager: Starting task 6.0 in stage 1.0 (TID 14, hadoop3, NODE_LOCAL, 1193 bytes)
15/07/19 10:56:35 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 1.0 (TID 11) in 10440 ms on hadoop3 (4/8)
15/07/19 10:56:36 INFO scheduler.TaskSetManager: Finished task 5.0 in stage 1.0 (TID 13) in 10885 ms on hadoop2 (5/8)
15/07/19 10:56:39 INFO scheduler.TaskSetManager: Starting task 7.0 in stage 1.0 (TID 15, hadoop2, RACK_LOCAL, 1193 bytes)
15/07/19 10:56:45 INFO scheduler.TaskSetManager: Finished task 6.0 in stage 1.0 (TID 14) in 9810 ms on hadoop3 (6/8)
15/07/19 10:56:47 INFO scheduler.TaskSetManager: Finished task 4.0 in stage 1.0 (TID 12) in 22025 ms on hadoop1 (7/8)
15/07/19 10:56:51 INFO scheduler.TaskSetManager: Finished task 7.0 in stage 1.0 (TID 15) in 12658 ms on hadoop2 (8/8)
15/07/19 10:56:51 INFO cluster.YarnClientClusterScheduler: Removed TaskSet 1.0, whose tasks have all completed, from pool
15/07/19 10:56:51 INFO scheduler.DAGScheduler: Stage 1 (count at <console>:15) finished in 29.690 s
15/07/19 10:56:51 INFO spark.SparkContext: Job finished: count at <console>:15, took 29.771373745 s
res2: Long = 10000000
```

3.3 YARN-Cluster 运行过程演示

3.3.1 运行程序

通过如下命令启动 Spark-Shell，在演示当中分配 3 个 Executor、每个 Executor 为 512M 内存

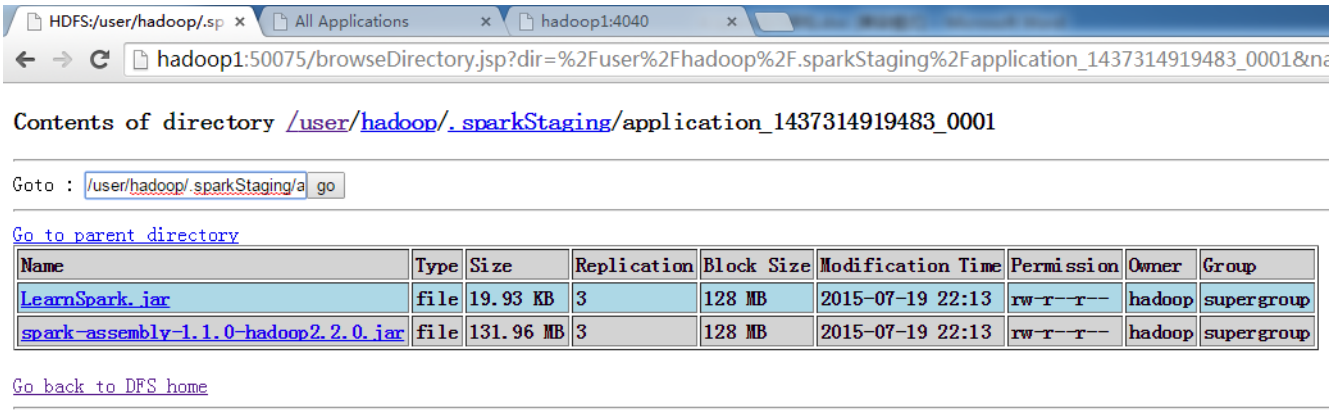
$cd /app/hadoop/spark-1.1.0

$ ./bin/spark-submit --master YARN-cluster --class class3.SogouResult --executor-memory 512m LearnSpark.jar hdfs://hadoop1:9000/sogou/SogouQ3.txt hdfs://hadoop1:9000/class3/output2

第一步 把相关的资源上传到 HDFS 中，相对于 YARN-Client 多了 LearnSpark.jar 文件

```
[hadoop@hadoop1 spark-1.1.0]$  ./bin/spark-submit --master yarn-cluster --class class3.SogouResult --executor-memory 512m LearnSpark.jar hdfs://hadoop1:9000/sogou/Sogou
Q3.txt  hdfs://hadoop1:9000/class3/output2
Spark assembly has been built with Hive, including Datanucleus jars on classpath
15/07/19 22:13:42 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
15/07/19 22:13:43 INFO client.RMProxy: Connecting to ResourceManager at hadoop1/192.168.10.111:8032
15/07/19 22:13:43 INFO yarn.Client: Got cluster metric info from ResourceManager, number of NodeManagers: 3
15/07/19 22:13:44 INFO yarn.Client: Max mem capability of a single resource in this cluster 8192
15/07/19 22:13:44 INFO yarn.Client: Preparing Local resources
15/07/19 22:13:45 INFO yarn.Client: Uploading file:/app/hadoop/spark-1.1.0/lib/spark-assembly-1.1.0-hadoop2.2.0.jar to hdfs://hadoop1:9000/user/hadoop/.sparkStaging/app
lication_1437314919483_0001/spark-assembly-1.1.0-hadoop2.2.0.jar
15/07/19 22:13:52 INFO yarn.Client: Uploading file:/app/hadoop/spark-1.1.0/LearnSpark.jar to hdfs://hadoop1:9000/user/hadoop/.sparkStaging/application_1437314919483_000
1/LearnSpark.jar
15/07/19 22:13:53 INFO yarn.Client: Prepared Local resources Map(__app__.jar -> resource { scheme: "hdfs" host: "hadoop1" port: 9000 file: "/user/hadoop/.sparkStaging/a
pplication_1437314919483_0001/LearnSpark.jar" } size: 20407 timestamp: 1437315232850 type: FILE visibility: PRIVATE, __spark__.jar -> resource { scheme: "hdfs" host: "h
adoop1" port: 9000 file: "/user/hadoop/.sparkStaging/application_1437314919483_0001/spark-assembly-1.1.0-hadoop2.2.0.jar" } size: 138366090 timestamp: 1437315232129 typ
e: FILE visibility: PRIVATE)
15/07/19 22:13:53 INFO yarn.Client: Setting up the launch environment
15/07/19 22:13:53 INFO yarn.Client: Setting up container launch context
15/07/19 22:13:53 INFO yarn.Client: Yarn AM launch context:
```

这些文件可以在 HDFS 中找到，具体路径为 http://hadoop1:9000/user/hadoop/.sparkStaging/应用编号 ：



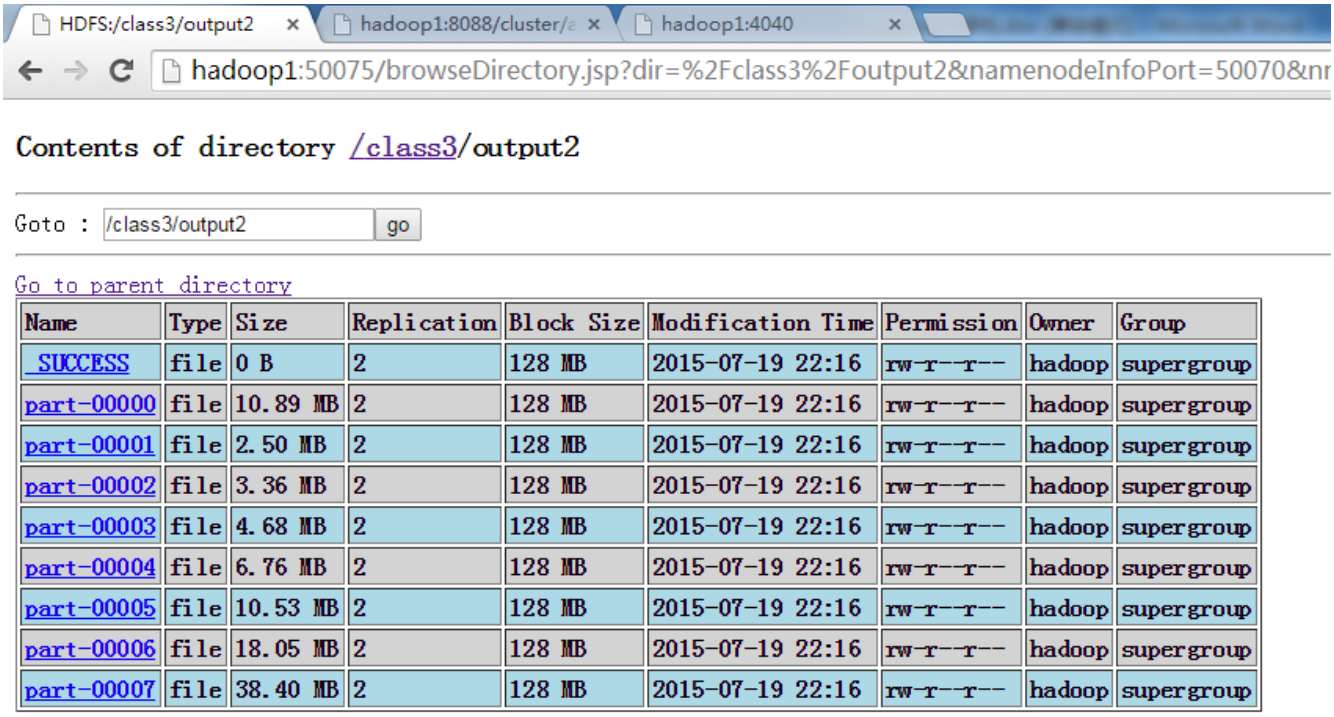
第二步 YARN 集群接管运行

首先 YARN 集群中由 ResourceManager 分配 Container 启动 SparkContext，并分配运行节点，由 SparkConext 和 NM 进行通讯，获取 Container 启动 Executor，然后由 SparkContext 的 YarnClusterScheduler 进行任务的分发和监控，最终在任务执行完毕时由 YarnClusterScheduler 通知 ResourceManager 进行资源的回收。

```
15/07/19 22:15:10 INFO yarn.Client: Application report from ResourceManager:
         application identifier: application_1437314919483_0001
         appId: 1
         clientToAMToken: null
         appDiagnostics:
         appMasterHost: N/A
         appQueue: default
         appMasterRpcPort: 0
         appStartTime: 1437315233658
         yarnAppState: ACCEPTED
         distributedFinalState: UNDEFINED
         appTrackingUrl: hadoop1:8088/proxy/application_1437314919483_0001/
         appUser: hadoop
```

3.3.2 运行结果

在 YARN-Cluster 模式中命令界面只负责应用的提交，SparkContext 和作业运行均在 YARN 集群中，可以从 http:// hadoop1:8088 查看到具体运行过程，运行结果输出到 HDFS 中，如下图所示：



Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	2	128 MB	2015-07-19 22:16	rw-r--r--	hadoop	supergroup
part-00000	file	10.89 MB	2	128 MB	2015-07-19 22:16	rw-r--r--	hadoop	supergroup
part-00001	file	2.50 MB	2	128 MB	2015-07-19 22:16	rw-r--r--	hadoop	supergroup
part-00002	file	3.36 MB	2	128 MB	2015-07-19 22:16	rw-r--r--	hadoop	supergroup
part-00003	file	4.68 MB	2	128 MB	2015-07-19 22:16	rw-r--r--	hadoop	supergroup
part-00004	file	6.76 MB	2	128 MB	2015-07-19 22:16	rw-r--r--	hadoop	supergroup
part-00005	file	10.53 MB	2	128 MB	2015-07-19 22:16	rw-r--r--	hadoop	supergroup
part-00006	file	18.05 MB	2	128 MB	2015-07-19 22:16	rw-r--r--	hadoop	supergroup
part-00007	file	38.40 MB	2	128 MB	2015-07-19 22:16	rw-r--r--	hadoop	supergroup

4 问题解决

4.1 YARN-Client 启动报错

在进行 Hadoop2.X 64bit 编译安装中由于使用到 64 位虚拟机，安装过程中出现下图错误：

[hadoop@hadoop1 spark-1.1.0]$ bin/spark-shell --master YARN-client --executor-memory 1g --num-executors 3
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Exception in thread "main" java.lang.Exception: When running with master 'YARN-client' either HADOOP_CONF_DIR or YARN_CONF_DIR must be set in the environment.
at org.apache.spark.deploy.SparkSubmitArguments.checkRequiredArguments(SparkSubmitArguments.scala:182)
at org.apache.spark.deploy.SparkSubmitArguments.<init>(SparkSubmitArguments.scala:62)
at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:70)
at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)

```
[hadoop@hadoop1 ~]$ cd /app/hadoop/spark-1.1.0/
[hadoop@hadoop1 spark-1.1.0]$ bin/spark-shell --master yarn-client --executor-memory 1g --num-executors 3
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Exception in thread "main" java.lang.Exception: When running with master 'yarn-client' either HADOOP_CONF_DIR or YARN_CONF_DIR must be set in the environment.
        at org.apache.spark.deploy.SparkSubmitArguments.checkRequiredArguments(SparkSubmitArguments.scala:182)
        at org.apache.spark.deploy.SparkSubmitArguments.<init>(SparkSubmitArguments.scala:62)
        at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:70)
        at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
[hadoop@hadoop1 spark-1.1.0]$
[hadoop@hadoop1 spark-1.1.0]$
```

参考资料：

（1）《Spark1.0.0 运行架构基本概念》 http://blog.csdn.net/book_mmicky/article/details/25714419

（2）《Spark 架构与作业执行流程简介》 http://www.cnblogs.com/shenh062326/p/3658543.html

（3）《Spark1.0.0 运行架构基本概念》 http://shiyanjun.cn/archives/744.html