# Spark运行架构

## 1.1 术语定义

l**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；

l**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；

l**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。

l**Cluster Manager**：指的是在集群上获取资源的外部服务，目前有：

Ø  Standalone：Spark原生的资源管理，由Master负责资源的分配；

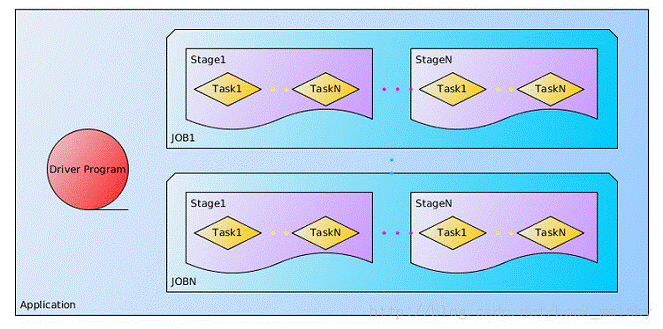
Ø  Hadoop Yarn：由YARN中的ResourceManager负责资源的分配；

l**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；

l**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；

l**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；

l**任务（Task）**： 被送到某个Executor上的工作任务；

[](http://images0.cnblogs.com/blog/107289/201508/111609093951090.gif)

## **1.2 Spark运行基本流程**

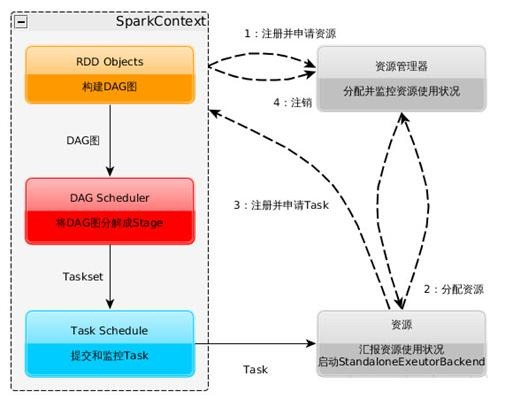
Spark运行基本流程参见下面示意图

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。

[](http://images0.cnblogs.com/blog/107289/201508/111609327851544.jpg)

Spark运行架构特点：

l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。

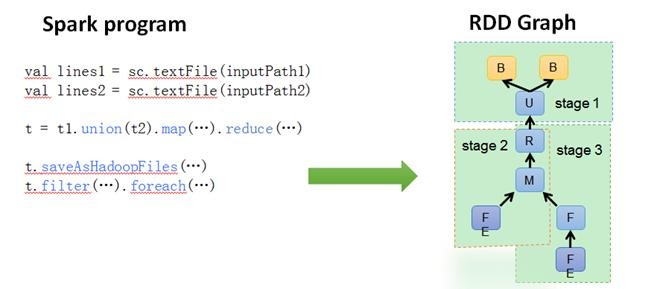
lSpark与资源管理器无关，只要能够获取executor进程，并能保持相互通信就可以了。

l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。

lTask采用了数据本地性和推测执行的优化机制。

### **1.2.1 DAGScheduler**

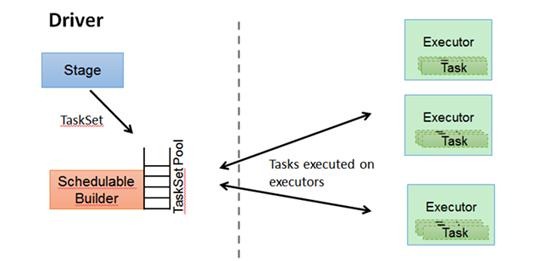
DAGScheduler把一个Spark作业转换成Stage的DAG（Directed Acyclic Graph有向无环图），根据RDD和Stage之间的关系找出开销最小的调度方法，然后把Stage以TaskSet的形式提交给TaskScheduler，下图展示了DAGScheduler的作用：

[](http://images0.cnblogs.com/blog/107289/201508/111609388178267.jpg)

### **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：

[](http://images0.cnblogs.com/blog/107289/201508/111609414109809.jpg)

在不同运行模式中任务调度器具体为：

l  Spark on Standalone模式为TaskScheduler；

l  YARN-Client模式为YarnClientClusterScheduler

l  YARN-Cluster模式为YarnClusterScheduler

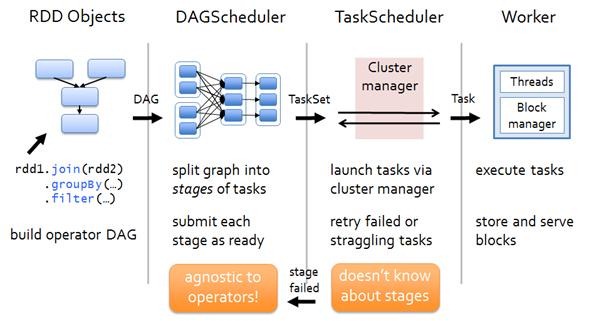
## **1.3 RDD运行原理**

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：

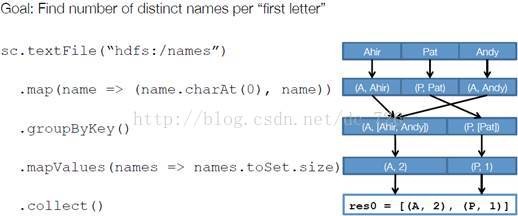
1.创建 RDD 对象

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。

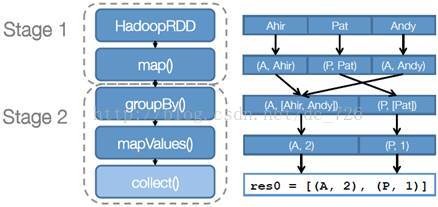
[](http://images0.cnblogs.com/blog/107289/201508/111609511453401.jpg)

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。

[](http://images0.cnblogs.com/blog/107289/201508/111610089731588.jpg)

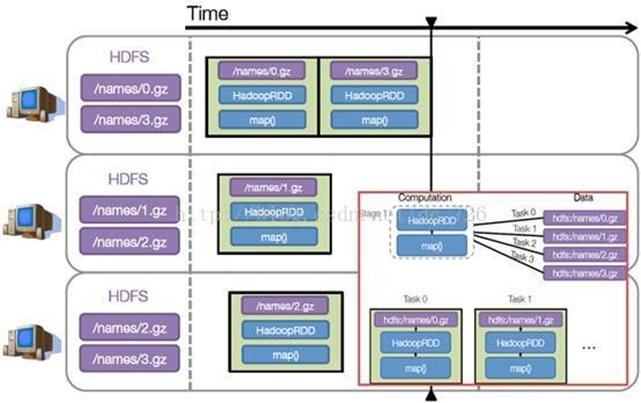
步骤 1 ：创建 RDD  上面的例子除去最后一个 collect 是个动作，不会创建 RDD 之外，前面四个转换都会创建出新的 RDD 。因此第一步就是创建好所有 RDD( 内部的五项信息 ) 。

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。

[](http://images0.cnblogs.com/blog/107289/201508/111610154426712.jpg)

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。

[](http://images0.cnblogs.com/blog/107289/201508/111610206146565.jpg)

# **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）；

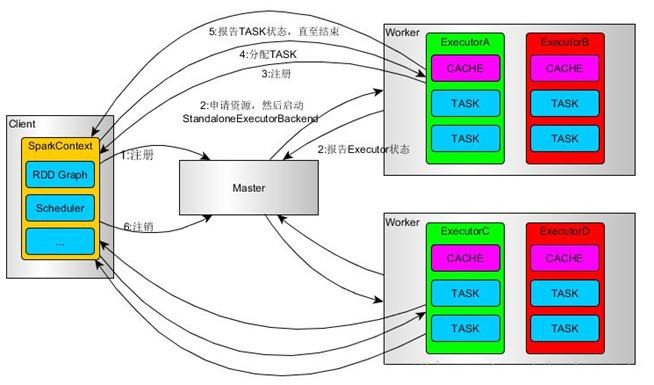
2.Master根据SparkContext的资源申请要求和Worker心跳周期内报告的信息决定在哪个Worker上分配资源，然后在该Worker上获取资源，然后启动StandaloneExecutorBackend；

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。

[](http://images0.cnblogs.com/blog/107289/201508/111610287072160.jpg)

## **2.2**  **Spark on YARN运行过程**

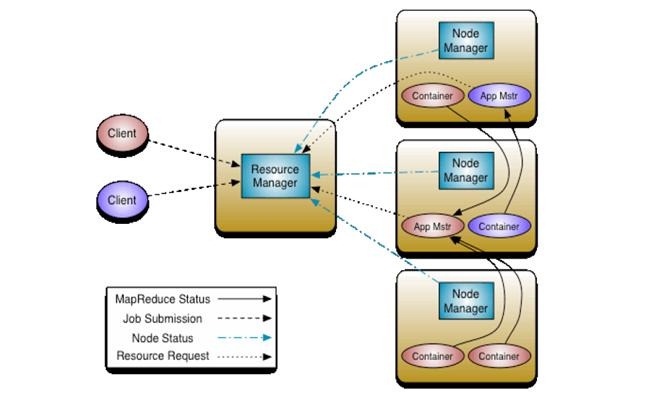
YARN是一种统一资源管理机制，在其上面可以运行多套计算框架。目前的大数据技术世界，大多数公司除了使用Spark来进行数据计算，由于历史原因或者单方面业务处理的性能考虑而使用着其他的计算框架，比如MapReduce、Storm等计算框架。Spark基于此种情况开发了Spark on YARN的运行模式，由于借助了YARN良好的弹性资源管理机制，不仅部署Application更加方便，而且用户在YARN集群中运行的服务和Application的资源也完全隔离，更具实践应用价值的是YARN可以通过队列的方式，管理同时运行在集群中的多个服务。

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：

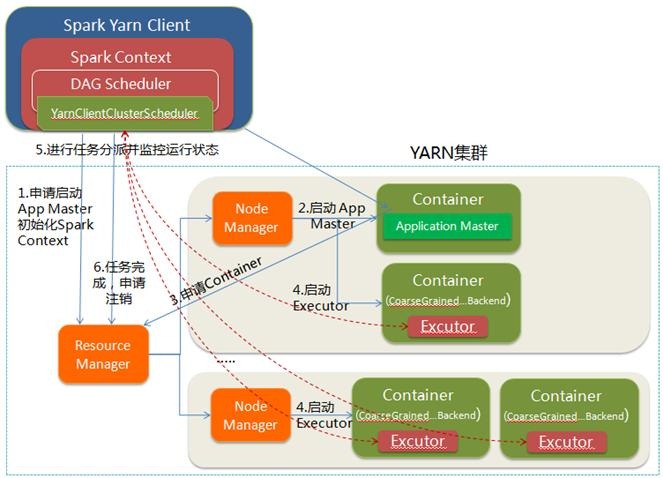
[](http://images0.cnblogs.com/blog/107289/201508/111610367546066.jpg)

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：

[](http://images0.cnblogs.com/blog/107289/201508/111610428794762.jpg)

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；

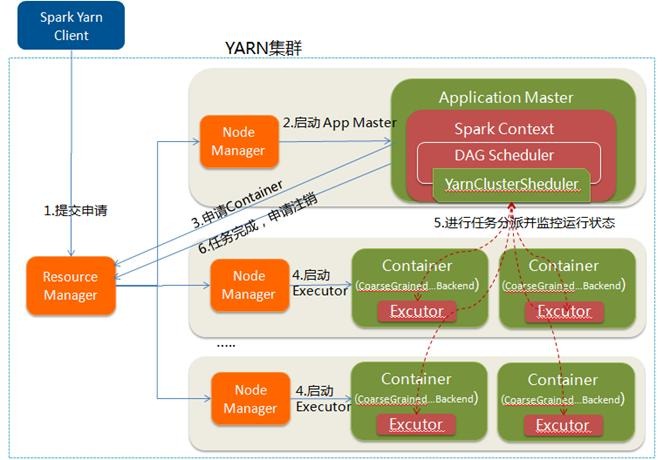
5.Client中的SparkContext分配Task给CoarseGrainedExecutorBackend执行，CoarseGrainedExecutorBackend运行Task并向Driver汇报运行的状态和进度，以让Client随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务；

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：

[](http://images0.cnblogs.com/blog/107289/201508/111610511924057.jpg)

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；

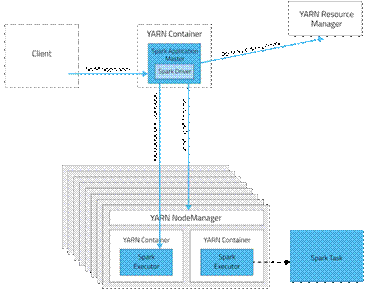
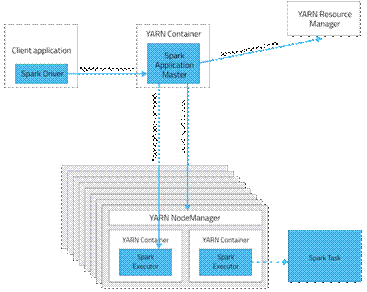
6.   应用程序运行完成后，ApplicationMaster向ResourceManager申请注销并关闭自己。

### **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。

l  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；

l  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。

[](http://www.iteblog.com/pic/spark-yarn-f31.png) [](http://www.iteblog.com/pic/spark-yarn-f22.png)