# 第二章 Spark部署模式

本书在讲解各章的内容时，主要是以local模式为例。但是对大部分想学习Spark的人而言，如何构建稳定的Spark集群也是学习的重点。由于各个模式在部署上有很大的差异，所以本章需要对不同模式下的部署进行讲解。本书主要讲解以下部署方式：本地部署模式（Local模式，主要用于代码调试和跟踪，不具备容错能力，所以不适用于生产环境。）、伪分布部署模式（Local-Cluster模式，用于在一台机器上模拟集群运行。）Standalone部署模式、（具备容错能力且支持分布式部署，所以可以用于实际的生产。）、YARN部署模式（第三方部署模式）。

## 2.1 Local模式

### 2.1.1 Local模式的介绍

在本地运行模式中，Spark的所有进程都在一台机器上的JVM上运行。在本地运行模式下，在作业划分调度后，任务集会发送到本地终端点，本地终端接收到任务后，会在本地启动Executor，这一切工作都在本地执行。本地模式的标准写法是local[N]，其中N代表可以使用N个线程，每个线程拥有一个core。如果不指定N，则默认是1个线程（该线程有1个core）。

### 2.1.2 Local模式部署

Local（本地）模式下部署Spark应用程序比较简单，可用于检测Spark是否编译安装成功，需要配置Java环境变量和设置主节点，后面安装Spark章节会详细讲解，以下介绍Java环境变量和主节点的配置。

1. 进入Spark主程序的conf目录，执行：cd spark-2.3.0-bin-hadoop2.7/conf。
2. 以spark-env.sh.template 文件为模板创建spark-env.sh文件。
3. 修改spark-env.sh配置文件：

cp spark-env.sh.template spark-env.sh

vim spark-env.sh

export SPARK\_MASTER\_IP=$YOUR\_MASTER\_IP //配置Master节点绑定的IP

export JAVA\_HOME=$YOUR\_JAVA\_HOME //配置Java的环境变量

1. 验证版本，当安装完毕且配置好环境变量之后，在终端输入spark-shell命令即可进入Spark命令模式，此时会提示Spark的当前版本。

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_\_/ .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.0

/\_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0\_171)

### **2.1.3 Local模式Spark应用程序的运行**

Local模式下运行Spark程序方式十分简单首先进入Spark安装的主目录，可以看到主目录下面会由bin文件夹，然后执行命令：

./bin/spark-submit \ #主程序入口

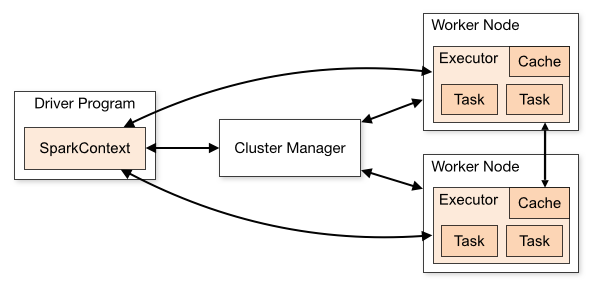
--class org.apache.spark.examples.SparkPi \ #主类选择

--master local[8] \ #设置主程序为本地，方括号内为线程数

/path/to/examples.jar \ #包含应用程序和所有依赖项的捆绑jar的路径

## 2.2 Standalone（独立）模式

在介绍独立模式之前，先介绍什么是集群模式（下面的Standalone模式、YARN模式、Mesos模式都属于集群模式）。Spark 的“集群”不是提供运算服务的，而是一种资源分配的调度器。执行任务的 Spark 进程作为客户端向“集群”申请资源(运算节点)，“集群”分配资源以后，这个 Spark 进程会分解一些计算工作，并把他们放到这些申请来的资源中运行。接下来通过一张Spark在分布式环境下的架构图对集群上运行Spark做一个初步的了解。



在分布式环境下，Spark集群是采用Master/Slave结构，在一个Spark集群中，有一个节点负责中央协调，调度各个分布式工作节点，这个中央协调节点就是驱动器节点（Driver），与之对应的工作节点被称为执行器节点（executor）节点。驱动器节点可以和大量执行器节点进行通信，它们作为独立Java进程运行。驱动器节点与所有 执行器节点一起被称为Spark应用（application）。Spark应用通过一个叫做集群管理器（Cluster Manager）的外部服务在集群中机器上启动。

### 2.2.1 Standalone模式的介绍

Standalone模式既独立模式，是Spark自带完整服务，可单独部署到一个集群中，无需依赖其他任何资源管理系统，只支持FIFO调度器。从一定程度上说，它是spark on yarn 和spark on mesos 的基础。在Standalone模式中，用户节点直接与master打交道，由driver负责向master申请资源，并由driver进行资源的分配和调度等等。

### 2.2.2 Standalone模式部署

安装Spark Standalone模式，需要在集群上的每个节点上下载Spark的编译版本即可部署，Standalone模式的Spark集群步骤如下：

1. 修改spark-env.sh配置文件（与Local模式相似）

cp spark-env.sh.template spark-env.sh

vim spark-env.sh

export SPARK\_MASTER\_IP=$YOUR\_MASTER\_IP //配置Master节点绑定的IP

export JAVA\_HOME=$YOUR\_JAVA\_HOME //配置Java的环境变量

1. 在Spark主目录下的conf配置目录下，创建一个名为slaves的文件，该文件内容为所有将要启动的Spark Worker的机器的hostname（主机名），每一行写一个。

cp slaves.template slaves

vim slaves

slave01

slave02

slave03

slave04

slave05

......

1. 发送配置文件spark-env.sh与slaves到所有的Worker节点（spark所在目录必须一致，因为master会登陆到worker上执行命令，master认为worker的spark路径与自己一样）以slave03为例：

scp -r $SPARK\_HOME/conf/spark-env.sh slave03:/$SPARK\_HOME/conf/

Scp -r $SPARK\_HOME/conf/slaves slave03:/$SPARK\_HOME/conf/

1. 主机通过ssh访问每个工作机器。默认情况下，ssh是并行运行的，需要设置无密码（使用私钥）访问权限。配置Master无密匙登陆slaves节点。
2. 在Master节点与所有节点安装openssh-server：

sudo apt-get install openssh-server

1. 建立SSH KEY：

ssh-keygen -t rsa -P “”

1. 在Master节点上启用SSH KEY：

cat  $HOME/.ssh/id\_rsa.pub >> $HOME/.ssh/authorized\_keys

sudo /etc/init.d/ssh reload

1. 验证SSH的配置：

ssh localhost

1. 将Master节点的authorized\_keys发送至所有的slaves节点，并且登陆验证。

部署完毕后，就可以启动集群中的Master节点和Worker节点。接下来介绍Standalone集群手动启动和脚本启动。

**1、手动启动集群介绍**

手动启动Master节点是用代码：./sbin/start-master.sh，启动后会打印出spark://HOST:POST，可以通过这个信息连接Master节点和Worker节点，或者通过Master的WebUI找到URL，默认的访问地址是http://localhost:8080，同时支持启动一个或更多的Worker节点，通过命令：./sbin/start-slave.sh <master-spark-URL>。如果启动了一个Worker节点，Master的WebUI中（默认http://localhost:8080）可以看到新增的Worker节点、CPU数目、内存大小，这些都会在网页列表中显示。

1. **脚本启动集群介绍**

之前介绍了spark主目录下的conf配置目录的slaves文件的创建，在该文件创建之后，就可以通过shell脚本在Master节点启动或终止集群。

sbin/start-master.sh  - 在脚本执行的机器上启动Master实例.

sbin/start-slaves.sh - 在conf / slaves文件中指定的每台机器上启动一个slaves实例.

sbin/start-all.sh - 如上所述启动一个Master和多个slaves.

sbin/stop-master.sh - 停止通过sbin / start-master.sh脚本启动的Master实例.

sbin/stop-slaves.sh - 停止conf / slaves文件中指定的机器上的所有slaves实例.

sbin/stop-all.sh - 停止启动的Master实例与slaves实例.

进入Spark主目录下的sbin文件夹中，可以执行./start-all.sh可以启动所有服务器上相关进程。执行jps命令可以查看当前服务器进程，所以当服务器是主节点时，可以看到Master进程，若是子节点，可看到Worker进程。

下面是一张spark-env.sh环境变量配置的表格，可以根据表格的环境变量模式进行配置，并复制到Worker节点上，可以使其配置生效。

|  |  |
| --- | --- |
| 环境变量 | 含义 |
| SPARK\_MASTER\_IP | 绑定一个IP给master. |
| SPARK\_MASTER\_PORT | 从另外一个端口启动master(默认: 7077) |
| SPARK\_MASTER\_WEBUI\_PORT | Master的web UI端口 (默认: 8080)，这个端口太常用，建议换一个 |
| SPARK\_LOCAL\_DIRS | 用于本地“暂存”空间目录，该目录应是系统中可快速读写的本地磁盘。 |
| SPARK\_WORKER\_CORES | 作业可用的CPU内核数量(默认: 所有CPU核都可用) |
| SPARK\_WORKER\_MEMORY | 允许Spark应用程序使用的内存大小（默认：所有内存减去1GB）；每个应用独立内存是spark.executor.memory配置的 |
| SPARK\_WORKER\_PORT | 启动Spark worker 的专用端口(默认：随机) |
| SPARK\_WORKER\_WEBUI\_PORT | worker 的web UI 启动端口(默认: 8081) |
| SPARK\_WORKER\_INSTANCES | 每台机器上运行worker数量 (默认: 1) |
| SPARK\_WORKER\_DIR | 运行应用程序的目录，该目录下包含日志和暂存空间（默认在SPARK\_HOME/work下） |
| SPARK\_DAEMON\_MEMORY | 分配给Spark master和 worker 守护进程的内存空间 (默认: 512MB) |
| SPARK\_DAEMON\_JAVA\_OPTS | Spark master 和 worker守护进程的JVM |

### 2.3.3 Standalone模式Spark程序的运行

1、spark-shell运行应用程序

Spark集群运行应用程序时，必须要把Master的Spark://IP:PORT URL传递给SparkContext构造函数。在集群上交互式Spark命令spark-shell，会以spark-env.sh中的SPARK\_MASTER\_IP与SPARK\_MASTRE\_PORT自动设置Master。

./bin/spark-shell --master spark://IP:PORT

2、spark-submit启动应用程序

spark-submit脚本提供了将编译的Spark应用程序提交到集群的最直接方式。 对于独立集群，Spark目前支持两种部署模式。 在客户端模式下，驱动程序在与提交应用程序的客户端相同的进程中启动。 然而，在集群模式下，驱动程序是从集群内的其中一个Worker进程启动的，客户端进程在完成其提交应用程序的责任时立即退出，而无需等待应用程序完成。

如果应用程序是通过spark-submit启动的，那么应用程序jar包将自动分发到所有Worker节点。对于应用程序依赖的任何其他jar包，应该使用逗号作为分隔符，通过--jars声明（例如--jars jar1，jar2），。

下面是提交应用程序的命令：

./bin/spark-submit \ #主程序入口

--class org.apache.spark.examples.SparkPi \ #主类选择

--master spark://$YOUR\_MASTER\_IP:7070 \ #设置主节点的端口地址

--executor-memory 2G \ #执行器内存为2G

--total-executor-cores 2 \ #启动执行器核心数为2

/path/to/examples.jar \ #包含应用程序和所有依赖项的捆绑jar包的路径

3、资源调度

独立群集模式当前仅支持跨应用程序的简单的FIFO调度程序。 但是，要允许多个并发执行，可以控制每个应用程序将使用的最大资源数量。 默认情况下，如果集群中一次只运行一个应用程序，它会获取所有CPU核。可以通过在SparkConf中设置spark.cores.max来限制核心数量。例子如下所示：

val conf = new SparkConf()

.setMaster(...)

.setAppName(...)

.set("spark.cores.max", "10") #设置最多内核数量为10

val sc = new SparkContext(conf)

此外，可以通过配置集群中spark.deploy.defaultCores来改变应用程序的默认值，而不必设置spark.cores.max，此时需要在配置文件spark-env.sh中添加下面内容：

export SPARK\_MASTER\_OPTS="-Dspark.deploy.defaultCores=<value>"

这对于用户可能没有单独配置最大内核数量的共享集群非常有用。

4、监控核日志

Spark的独立模式提供了一个基于Web的用户界面来监视群集。Master节点和每个Worker节点都有自己的Web UI，显示群集和工作统计信息。默认情况下，您可以在端口8080访问主节点的WebUI。另外，每个作业的详细日志输出也写入每个从节点的工作目录（SPARK\_HOME /work）。 每个作业都会有两个文件stdout和stderr，并将所有输出写入其控制台。

## 2.3 其他模式

如果只想在一堆机器上运行Spark，自带的独立模式是部署该集群的最简单的办法。但是，如果要一个与别的分布式应用共享的集群，Spark就需要运行在集群管理器上。本节将介绍Spark运行在第三方资源管理器上的部署方案，主要围绕Spark运行在Mesos与YARN上。

### 2.3.1 YARN模式

#### 1、 YARN模式介绍

资源管理器（Yet Another Resource Negotiator，YARN）是一个通用的资源管理系统能够为上层应用提供统一资源管理和资源调度。YARN的引入为集群在利用率、资源统一管理和数据共享等方面带来巨大的好处。YARN最初是为了修复MapReduce的不足，而且还对可伸缩性、可靠性和集群利用率进行提升。YARN 将资源管理和作业调度及监控分成了两个独立的服务程序：全局的资源管理RM（Resource Manager）和针对个人应用的AM（Application Master），此处应用指的是传统意义上的MapReduce任务或是任务的有向无环图（DAG）。

Spark on YARN模式，借助了YARN良好的弹性资源管理机制，不仅部署应用程序更加方便，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。

#### 2、 YARN模式部署

YARN模式的部署分为以下几个步骤：

1. 配置前准备

在配置Spark之前，首先要检查一下host文件，可以用root账户登陆服务器，然后执行

vim /etc/hosts命令，查看需要的Spark服务器的IP的机器是否都存在在内，没有的话需要添加进去。然后就是配置环境变量，终端中执行 vim /etc/profile，打开环境变量配置的文件,输入如下代码：

export SPARK\_HOME=”SPARK 安装目录”

export PATH=$PATH:$SPARK\_HOME/bin:$SPARK\_HOME/sbin

export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop

export YARN\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop

接着，为了使配置的环境变量生效，在终端执行source /etc/profile命令。

1. 主节点和Java环境变量设置

与其他部署模式修改spark-env.sh文件一样添加如下代码：

cp spark-env.sh.template spark-env.sh

vim spark-env.sh

export SPARK\_MASTER\_IP=$YOUR\_MASTER\_IP //配置Master节点绑定的IP

export JAVA\_HOME=$YOUR\_JAVA\_HOME //配置Java的环境变量

配置完成之后，就可以确定执行Spark的主节点了。接着，打开slaves文件，把子节点IP地址或名称添加进来。

cp slaves.template slaves

vim slaves

slave01

slave02

slave03

slave04

slave05

......

最后要将spark主目录文件夹、环境变量配置文件、hosts发送至子节点的各个服务器上，并且都要执行环境变量配置文件 source /etc/profile命令，然后先启动Hadoop，再启动Spark。

#### 3、 YARN模式Spark集群的运行

支持在YARN上运行的版本已在0.6.0版中添加到Spark中，并在后续版本中得到改进。运行之前，首先确保HADOOP\_CONF\_DIR或YARN\_CONF\_DIR指向包含Hadoop集群（客户端）配置文件的目录，该目录中包含的配置被分发到YARN集群，以便应用程序使用的所有容器使用相同的配置。（这一步在环境配置中讲了）部署完毕后，可以在YARN上运行Spark程序，其提交任务方式与独立模式类似，但是原理有些不同。

1、在YARN上启动Spark

有两种部署模式可用于在YARN上启动Spark应用程序。 在集群模式（yarn-cluster）下，Spark驱动程序在由集群上的YARN管理的应用程序主进程内运行，并且客户端可以在启动应用程序后离开。 在客户端模式（yarn-client）下，驱动程序在客户端进程中运行，而应用程序主服务器仅用于从YARN请求资源。

在yarn-client模式下调用Spark应用：

./bin/spark-submit \

--class path.your\_class \

--master yarn-client [options] <app jar> [app options]

提交应用程序代码如下：

./bin/spark-submit \ #主程序入口

--class org.apache.spark.examples.SparkPi \ #主类选择

--master yarn-client \ #设置主节点的类型

--num-executor 4 \ #执行器节点数目

--driver-memory 4g \ #驱动器内存为4G

--executor-memory 2g \ #执行器内存为2G

--executor-cores 1 \ #启动CPU核心数为2

examples/jars/spark-examples\*.jar \ #包含应用程序和所有依赖项的捆绑jar包的路径

在yarn-cluster模式下，启动应用的操作是一样的，只不过把--master yarn-client改为--master yarn-cluster。

2、添加其他的jar包

在cluster模式下，驱动程序运行在与客户端不同的机器上，因此SparkContext.addJar将无法使用客户端本地文件。要使客户端上的文件可用于SparkContext.addJar，在启动命令中使用--jars选项包含它们。

./bin/spark-submit

--class my.main.Class \

--master yarn-cluster \

--jars my-other-jar.jar,my-other-other-jar.jar \

my-main-jar.jar \

app\_arg1 app\_arg2

1. 监控和日志

程序运行完毕后，若Hadoop集群配置完毕后，Hadoop集群cluster页面上（http://$YOUR\_MASTER\_IP:8080/cluster/apps/FINISHED）可以看到执行的Spark程序及其输出日志，在http://$YOUR\_MASTER\_IP:8080/路径可看到执行的Spark任务。

### 2.3.2 Mesos模式

#### 1、 Mesos模式介绍

Mesos是Apache旗下的开源软件，采用Master/Slave结构，他作为Apache下的开源分布式资源管理框架，被称为是分布式系统的内核。Mesos是一个集群管理器，提供有效的、跨分布式应用或框架的资源隔离共享，可以支持Hadoop、MPI、Hypertable、Spark等。Apache Mesos由四个组件组成，分别为Mesos Master、Mesos Slaves、框架和执行容器。Mesos是整个系统的核心，负责管理接入到Mesos的各个计算框架和Slave，并将Slave上的资源按照指定的算法分配给计算框架。

#### 2、 Mesos模式的部署

（1）准备工作

检查host文件，查看Spark“服务器IP名”是否存在，不存在请添加，同时设置免ssh登陆（按照YARN模式的免ssh登陆步骤），再安装zookeeper。

（2）安装deb包

在Spark命令行中输入下面代码，安装所需的deb包

$ sudo dpkg -i \*\*\*.deb

（3）修改配置

修改Mesos文件的配置信息

#进入mesos目录

$ cd /home/mesos

#添加masters和slaves文件，文件中每行是master和slave的节点主机名或IP地址

$ sudo vim masters node #后面添加主节点主机名或者IP地址

$ sudo vim slaves node #后面添加从节点主机名或者IP地址

#修改master配置

$ sudo cp mesos-master-env.sh.template mesos-master-env.sh

$ sudo vim mesos-master-env.sh

#修改mesos-master-env.sh

export MESOS\_log\_dir=/home/mesos/master/log #日志文件存放目录

export MESOS\_work\_dir=/home/mesos/master/work #持久化文件存放目录

export MESOS\_ZK=zk://node:2181/mesos export MESOS\_quorum=1

#修改slave配置

$ sudo cp mesos-slave-env.sh.template mesos-slave-env.sh

$ sudo vim mesos-slave-env.sh

#修改mesos-slave-env.sh

export MESOS\_log\_dir=/home/mesos/slave/log

export MESOS\_work\_dir=/home/mesos/slave/work

export MESOS\_isolation=cgroups

#复制mesos-deploy-env.sh文件

$ sudo cp mesos-deploy-env.sh.template mesos-deploy-env.sh

（4）Spark本身的配置

与其他部署模式修改spark-env.sh文件一样添加如下代码：

cp spark-env.sh.template spark-env.sh

vim spark-env.sh

export SPARK\_MASTER\_IP=$YOUR\_MASTER\_IP #配置Master节点绑定的IP

export JAVA\_HOME=$YOUR\_JAVA\_HOME #配置Java的环境变量

最后把更新的环境变量发送至其他的子节点服务器上执行。

#### 3、 Mesos模式Spark集群的运行

（1）客户端模式下

客户端模式下，客户端机器上将会启动一个Spark Mesos框架，并且会等待驱动（driver）的输出。驱动需要spark-env.sh中的一些配置项，以便和Mesos交互操作：

在Spark-env.sh中设置一些环境变量：

export MESOS\_NATIVE\_JAVA\_LIBRARY=<path to libmesos.so>

export SPARK\_EXECUTOR\_URI=<上文所述的上传Spark包对应的URL>

同样，spark.executor.uri 也需要设成Spark包对应的URL，然后，你就可以向这个Mesos集群提交Spark应用了，当然，你需要把Mesos Master URL（mesos://）传给SparkContext，例如：

val conf = new SparkConf()

.setMaster("mesos://HOST:5050")

.setAppName("My app")

.set("spark.executor.uri", "<path to spark-1.6.0.tar.gz uploaded above>");

val sc =newSparkContext(conf);

如果是在Spark shell中，spark.executor.uri参数值是从SPARK\_EXECUTOR\_URI继承而来的，所以不需要额外再传一个系统属性：./bin/spark-shell --master mesos://host:5050。

1. 集群模式下

Mesos同样也支持Spark以集群模式提交作业，这种模式下，驱动器将在集群中某一台机器上启动，其运行结果可以在Mesos Web UI上看到。要使用集群模式，你首先需要利用 sbin/start-mesos-dispatcher.sh脚本启动 MesosClusterDispatcher，并且将Mesos Master URL（如：mesos://host:5050）传给该脚本。MesosClusterDispatcher启动后会以后台服务的形式运行在本机。客户机可以向Mesos集群提交任务，代码如下：

./bin/spark-submit \

--class org.apache.spark.examples.SparkPi \ #主类选择

--master mesos://207.184.161.138:7077 \ #设置主节点URL

--deploy-mode cluster \ #设置运行模式类型

--supervise \ #设置如果失败重新启动程序命令

--executor-memory 20G \ #设置执行器内存20G

--total-executor-cores 100\ #设置执行器所有CPU核为100个

http://path/to/examples.jar \ #包含应用程序和所有依赖项的捆绑jar包的路径

1. Mesos运行模式的类型

Spark可以在Mesos的两种模式下运行，“粗粒度”模式（默认）和“细粒度”模式。接下来介绍什么是粗粒度模式，什么是细粒度模式。

**粗粒度模式：**应用程序的各个任务正式运行之前，需要将运行环境中的资源全部申请好，且运行过程中要一直占用这些资源，即使不用，最后程序运行结束后，回收这些资源。举个例子，比如你提交应用程序时，指定使用5个Executor运行你的应用程序，每个Executor占用5GB内存和5个CPU，每个Executor内部设置了5个Task，则Mesos需要先为Executor分配资源并启动它们，之后开始调度任务。粗粒度是Mesos的默认模式。你也可以显示地在 [SparkConf](http://spark.apache.org/docs/latest/configuration.html" \l "spark-properties) 中设置spark.mesos.coarse属性为true来启用该模式：

conf.set("spark.mesos.coarse","true")

**细粒度模式：**鉴于粗粒度模式会造成大量资源浪费，Spark On Mesos还提供了另外一种调度模式，细粒度模式，这种模式类似于现在的云计算，思想是按需分配。与粗粒度模式一样，应用程序启动时，先会启动Executor，但每个Executor占用资源仅仅是自己运行所需的资源，不需要考虑将来要运行的任务，之后，Mesos会为每个Executor动态分配资源，每分配一些，便可以运行一个新任务，单个Task运行完之后可以马上释放对应的资源。每个Task会汇报状态给Mesos slave和Mesos Master，便于更加细粒度管理和容错，这种调度模式类似于MapReduce调度模式，每个Task完全独立，优点是便于资源控制和隔离，但缺点也很明显，短作业运行延迟大。要使用细粒度模式，可以在SparkConf 中将 spark.mesos.coarse 属性设为false：

conf.set("spark.mesos.coarse","false")

**YARN与Mesos的对比：**

|  |  |  |
| --- | --- | --- |
|  | Mesos | YARN |
| 框架担任的角色 | 各个计算框架需在Mesos中部署后才能使用，完全融入到Mesos中 | 计算框架只是作为客户端的库使用，不在YARN中部署即可使用 |
| 调度机制 | 双层调度；第一层由Mesos Master将空闲资源分配给框架，第二层由各个框架自带的调度器对资源的使用进行分配 | 双层调度，第一层由RM（Resource Manager）分配资源，第二层NM（Node Manager）再对框架的任务进行调度 |
| 资源分配 | 粗粒度和细粒度分配 | 细粒度分配 |
| 扩展性 | 支持6,000 ~ 50,000个节点 | 目标是支持6,000 ~ 100,000个节点 |

## 2.4 总结

如果是抱着先学习Spark基础的学习态度，安装本地模式就行。如果是想简单学习，接触集群模式的话，可以选择独立集群管理器，因为它安装简单，而且如果只是使用Spark的话，独立集群管理器提供与其他集群管理器一样的功能。如果在使用Spark的同时使用其他应用，或者是要用到更加丰富的资源调度功能（比如说队列），那么使用YARN和Mesos可以满足需求。但是目前YARN支持细粒度模式，如果想用粗粒度模式的话，还是需要用到Mesos模式。