# Spark概述

## Spark简介

### 什么是Spark

Spark是加州大学伯克利分校AMP（Algorithms、Machines and People Lab）实验室开发的通用大数据处理集群计算框架，通过将大量的数据集计算任务分配到多台计算机上，提供高效内存计算。Spark和Hadoop的MapReduce计算框架类似，但它的运算是基于内存的，所以相对于其他的处理方法，Spark在性能上更具优势。

Spark拥有多种语言的函数式编程API，提供了除map和reduce之外更多的运算符，这些操作是通过一个称作弹性分布式数据集(Resilient Distributed Dataset，RDD)的分布式数据框架进行的。由于RDD可以被缓存在内存中，Spark对迭代应用特别有效，因为这些应用中，数据是在整个算法运算过程中都可以被重用。大多数机器学习和最优化算法都是迭代的，使得Spark对数据科学来说是个非常有效的工具。另外，由于Spark非常快，可以通过类似Python REPL的命令行提示符交互式访问。

Spark集批处理、实时流处理、交互式查询与图计算与一体，避免多种运算 场景下需要部署不同集群带来的资源浪费。Spark在过去的几年中获得了极大的关注，并得到广泛的应用，Spark社区也成为大数据领域和Apache软件基金会最活跃的项目之一，其活跃度甚至远超曾经只能望其项背的Hadoop。

**Spark有下面几个显著的优点：**

1、运行速度快

Spark的运行速度有多快呢，官方提供的数据曾表明如果数据是从磁盘读取的，Spark的速度是Hadoop MapReduce的十倍以上；如果数据是从内存中读取的话，它的速度可以达到Hadoop MapReduce的一百多倍。Spark为什么会有这么快的计算速度呢，其主要原因由以下三个方面：

1. Spark默认情况下的迭代的数据保存在内存中，后面的运行作业是用这些结果来计算的，相比较而言，Hadoop的计算结果是保存在磁盘中的，后续的计算需要从磁盘中读取之前的计算结果。因为从内存中读取数据要比从磁盘读取数据快的多，所以Spark运行速度会显得快很多。尤其是在做机器学习的时候，若用Hadoop，它每次计算都要从磁盘读数据，然后计算完把数据存在磁盘上，不断迭代数据，会消耗大量的I/O。
2. 由于复杂的计算任务要多步骤才能实现，且步骤之间由依赖性。对于这些步骤，Hadoop需要借助Oozie等工具来处理。而Spark在执行任务前，可以将这些步骤根据依赖关系来形成DAG图(有向无环图)，任务执行依照图，所以不需要人工干预，优化计算路径，减少I/O操作。
3. JVM的优化（Java Virtual Machine），Hadoop每次MapReduce操作，启动一个Task便会启动一次JVM，基于进程的操作。而Spark每次MapReduce操作是基于线程的，只在启动Executor启动一次JVM，内存的Task操作是在线程复用的。每次启动JVM的时间可能就需要几秒甚至十几秒，那么当Task多了，这个时间Hadoop不知道比Spark慢了多少。
4. 通用性强

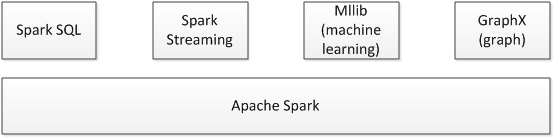


图 1-1 Spark技术堆栈

Spark有一个强大的堆库,包括可以用于交互式查询的Spark SQL、用于实时流处理的Spark Streaming、用于机器学习的Spark MLlib和用于图计算的Spark GraphX，而且可以在同一个应用中无缝的组合使用这些库。Spark这种统一的解决方案十分有吸引力，毕竟任何一家公司都想用统一的平台去处理遇到的问题，来减少开发和维护的人力成本和部署的物力成本。当然，作为统一的解决方案，Spark并没有以牺牲性能为代价，反而在性能方面，Spark具有很大的优势。

1. 易用性好

Spark支持Java、Scala、Python、和R等语言编写应用程序，大大降低使用者的门槛，而且它自带的80多个高等级操作符，允许在Scala、Python、R的shell中进行交互式查询，这表明我们可以很方便地在这些shell中使用Spark集群来验证解决问题的方法，而不是像以前一样，需要打包、上传集群、验证等。

1. 可融合性

Spark可以十分方便地与其他开源产品进行融合。Spark可以使用Hadoop的YARN与Apache Mesos作为它的资源管理和调度器，而且可以处理所有Hadoop支持的数据，包括HDFS、Cassandra、HBase等。这对部署了Hadoop集群的用户来说很重要，因为不需要数据迁移就可以使用Spark的强大处理能力。Spark也可以不依赖与第三方的资源管理和调度器，它实现Standalone作为其内置资源管理和调度框架，这样也会进一步降低Spark的使用门槛，使得所有人都可以十分容易地部署和使用Spark。

### Spark与MapReduce比较

Spark是借鉴Hadoop MapReduce发展而来的，继承分布式并行计算的优点，且改进MapReduce的明显缺点，具体表现如下：

1. Spark运算效率高：MapReduce在数据Shuffle之前，要花费大量时间排序，而Spark不需要对所有情景进行排序，由于采用有向无环图（DAG）执行计划，每次输出结果可以缓存在内存中，所以迭代运算效率高。对比而言，MapReduce的计算结果保存在磁盘上，上述的比较表明Spark减少迭代过程中数据的落地，提高处理效率。
2. Spark容错率高：Spark引进弹性分布式数据集（Resilient Distributed Dataset，RDD）的概念，它是分布在一组节点中的只读对象的集合。这些对象是弹性的，如果一部分数据集丢失了，可以根据“血统”对它们进行重建，一旦某个RDD失败了，就能通过父RDD自动重建，保证容错性。（弹性的含义：“弹性”是指在任何时候都能进行重算。这样当集群中的一台机器挂掉而导致存储在其上的RDD丢失后，Spark还可以重新计算出这部分的分区的数据。但用户感觉不到这部分的内容丢失过。这样RDD数据集就像块带有弹性的海绵一样，不管怎样挤压（分区遭到破坏）都是完整的。）
3. Spark更加通用：Hadoop只提供Map和Reduce两种操作，而Spark提供的数据集操作的类型很多。转换操作内容有Map、Filter，FlatMap、Sample、GroupByKey、ReduceByKey、Union、Join、Cogroup、MapValues、Sort和PartionBy等操作类型；行动操作包括Collect、Reduce、Lookup、Save等操作。另外，处理节点间的通信模型也不止有Shuffle一种模式，用户可以命名、物化、控制中间的结果的存储、分区。
4. Spark任务调度开销低：MapReduce系统是为了处理很长的批量作业设计的，在一些极端情况下，提交任务延迟很高，而Spark采用了事件驱动的类库AKKA来启动任务，通过线程池复用线程避免线程启动及切换产生的开销。
5. Spark提供丰富的API：Spark支持多种语言编程，比如Scala、Python及Java，便于开发者在自己熟悉的环境下工作。Spark自带80多个算子，且允许Spark Shell环境下交互式计算，开发者可以像写单机程序一样开发分布式程序，轻松用Spark搭建大数据的平台。
6. 代码量少：相同的应用程序Spark的代码量比Hadoop MapReduce少50%~80%。

### Spark的演进路线图

Spark在2009年算法比赛中诞生，在随后的四年时间内，Spark在Berkeley’s AMPLab逐渐形成现在的Spark雏形。并且在2013年6月成为了Apache的一个孵化项目，并在八个月后成为它的顶级项目。2014年底发布第一个正式版Spark 1.0.0，在随后时间里大致以三个月为周期发布一个小版本，且在2016年的七月推出Spark 2.0正式版，具体的严禁时间如下：

·2009年加州大学伯克利分校AMP实验室开始编写最初的源代码

·2010年开放源代码

·2012年2月发布0.60版本

·2013进入Apache

·2014年2月成为Apache的顶级项目

·2014年5月Spark 1.0.0版本发布

·2014年9月Spark 1.1.0版本发布

·2014年12月Spark 1.2.0版本发布

·2015年3月Spark 1.3.0版本发布

·2015年6月Spark 1.4.0版本发布

·2015年9月Spark 1.5.0版本发布

·2016年1月Spark 1.6.0版本发布

·2016年5月Spark 2.0.0 Preview版本发布

·2016年7月Spark 2.0.0正式版本发布

·2016年12月Spark 2.1.0版本发布

·2017年7月Spark 2.2.0版本发布

·2018年2月Spark 2.3.0版本发布

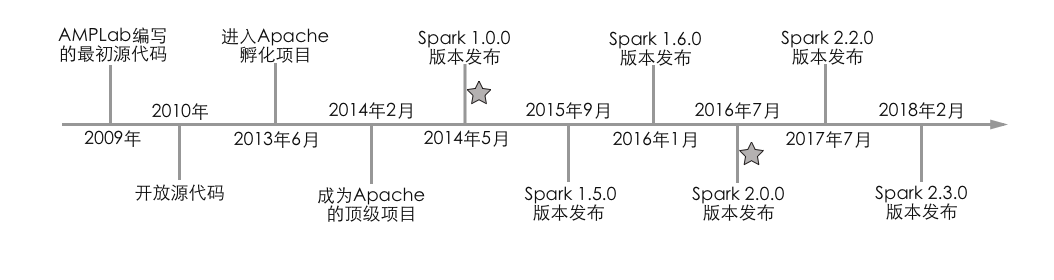


图 1-2 Spark演进时间轴

Spark进入了Apache之后就因为它的代码开源、内存计算和一站式解决方案风靡大数据生态圈，成为了Apache基金会最活跃的项目。

## Spark生态系统

Spark生态圈也称为BDAS（伯克利数据分析栈），是伯克利APMLab实验室打造的，力图在算法（Algorithms）、机器（Machines）、人（People）之间通过大规模集成来展现大数据应用的一个平台。伯克利AMPLab运用大数据、云计算、通信等各种资源以及各种灵活的技术方案，对海量不透明的数据进行甄别并转化为有用的信息，以供人们更好的理解世界。该生态圈已经涉及到机器学习、数据挖掘、数据库、信息检索、自然语言处理和语音识别等多个领域。

Spark生态圈以Spark Core为核心，从HDFS、Amazon S3、Allusio、NoSQL、传统文件和HBase等读取数据，以MESS、YARN、mesos和自身携带的Standalone为资源管理器调度Job完成Spark应用程序的计算。 这些应用程序可以来自于不同的组件，如Spark Shell/Spark Submit的批处理、Spark Streaming的实时处理应用、Spark SQL的即席查询、BlinkDB的权衡查询、MLlib/MLbase的机器学习、GraphX的图处理和SparkR的数学计算等等。

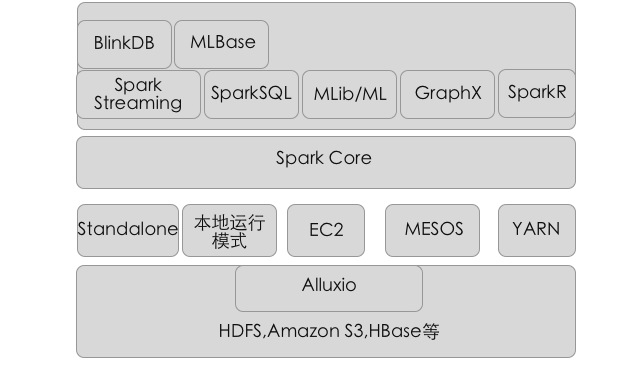


图 1-3 Spark生态系统

### Spark Core

Spark Core是整个BDAS的核心组件，是一种大数据分布式处理框架，它不仅实现了MapReduce的算子map函数和reduce函数及计算模型，还提供filter、join、groupByKey等丰富的算子。Spark Core有下面几个主要的特征：

（1）Spark Core提供多种运行模式，不仅可以使用自身的运行模式处理任务，如本地模式、Standalone，还可以使用第三方资源调度框架来处理任务，比如YARN、MESOS等。

（2）Spark Core提供了有向无环图分布式并行计算框架，而且提供内存机制来支持多次迭代计算或数据共享，大大减少迭代计算间读数据的开销，对数据挖掘这种需要多次迭代数据的处理来说，这种机制将会大大提高它的性能。

（3）Spark把分布式数据抽象成弹性分布式数据集（RDD），实现了应用任务的调度、RPC、序列化和压缩，并为运行在其上层的组件提供API，另外这些对象集合是弹性的，若有部分数据集丢失，它可根据“血统”对丢失部分进行重建，提高数据的容错性。

### Spark Streaming

Spark Streaming 是Spark核心API的一个扩展，可以实现高吞吐量的、具备容错机制的实时流数据的处理。支持从多种数据源获取数据，包括Kafk、Flume、Twitter、ZeroMQ、Kinesis 以及TCP sockets，从数据源获取数据之后，可以使用诸如map、reduce、join和window等高级函数进行复杂算法的处理。最后还可以将处理结果存储到文件系统，数据库和现场仪表盘。在“One Stack rule them all”的基础上，还可以使用Spark的其他子框架，Spark Streaming最大的优势是提供的处理引擎和RDD编程模型可以同时进行批处理和流处理。



图 1-4 Spark Streaming处理的数据流图

**·计算流程：**Spark Streaming是将流式计算分解成一系列短小的批处理作业。这里的批处理引擎是Spark Core，也就是把Spark Streaming的输入数据按照batch size（如1秒）分成一段一段的数据（Discretized Stream），每一段数据都转换成Spark中的RDD（Resilient Distributed Dataset），然后将Spark Streaming中对DStream的Transformation操作变为针对Spark中对RDD的Transformation操作，将RDD经过操作变成中间结果保存在内存中。整个流式计算根据业务的需求可以对中间的结果进行叠加或者存储到外部设备。

**·容错性：**对于流式计算来说，容错性至关重要。首先我们要明确一下Spark中RDD的容错机制。每一个RDD都是一个不可变的分布式可重算的数据集，其记录着确定性的操作继承关系（lineage），所以只要输入数据是可容错的，那么任意一个RDD的分区（Partition）出错或不可用，都是可以利用原始输入数据通过转换操作而重新算出的。对于Spark Streaming来说，其RDD的传承关系如下图所示，图中的每一个椭圆形表示一个RDD，椭圆形中的每个圆形代表一个RDD中的一个Partition，图中的每一列的多个RDD表示一个DStream（图中有三个DStream），而每一行最后一个RDD则表示每一个Batch Size所产生的中间结果RDD。我们可以看到图中的每一个RDD都是通过lineage相连接的，由于Spark Streaming输入数据可以来自于磁盘，例如HDFS（多份拷贝）或是来自于网络的数据流（Spark Streaming会将网络输入数据的每一个数据流拷贝两份到其他的机器）都能保证容错性，所以RDD中任意的Partition出错，都可以并行地在其他机器上将缺失的Partition计算出来。这个容错恢复方式比连续计算模型（如Storm）的效率更高。

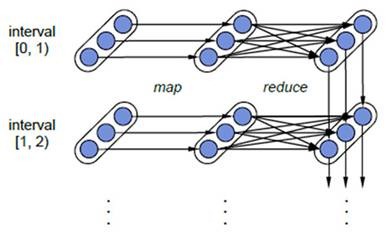


图 1-5 Spark Streaming中RDD的血统关系图

**·实时性：**对于实时性的讨论，会牵涉到流式处理框架的应用场景。Spark Streaming将流式计算分解成多个Spark Job，对于每一段数据的处理都会经过Spark DAG图分解以及Spark的任务集的调度过程。对于目前版本的Spark Streaming而言，其最小的Batch Size的选取在0.5~2秒钟之间（Storm目前最小的延迟是100ms左右），所以Spark Streaming能够满足除对实时性要求非常高（如高频实时交易）之外的所有流式准实时计算场景。

**·扩展性与吞吐量：**Spark目前在EC2上已能够线性扩展到100个节点（每个节点4Core），可以以数秒的延迟处理6GB/s的数据量（60M records/s），其吞吐量也比流行的Storm高2～5倍，图4是Berkeley利用WordCount和Grep两个用例所做的测试，在Grep这个测试中，Spark Streaming中的每个节点的吞吐量是670k records/s，而Storm是115k records/s。

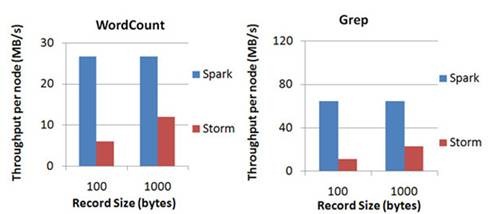


图 1-6 Spark Streaming与Storm吞吐量比较图

### Spark SQL

Shark是Spark SQL的前身，它发布的时候Hive可以说是SQL的唯一选择。Shark是构建在Spark和Hive基础上的数据仓库，它是通过把HQL翻译成Spark上对应的RDD操作，之后通过Hive的Metadata获取数据库中表的信息，最后信息被Shark获取，放入Spark计算。由于底层计算用了Spark，其性能相比于MapReduce的Hive普遍快了2倍以上，并且当数据全部存储与内存时要快10倍以上。2014年7月1日，Spark社区推出Spark SQL，重新实现SQL解析等之前Hive没有完成的工作，因此，Spark SQL在功能上覆盖原有的Shark，且性能更优。Spark SQL有如下特点：

·支持多种数据源：Hive、RDD、Parquet、JSON、JDBC等。

·多种性能优化技术：in-memory columnar storage、byte-code generation、cost model动态评估等。

·组件扩展性：对于SQL的语法解析器、分析器以及优化器，用户都可以自己重新开发，并且动态扩展。

Spark SQL的性能优化技术简介：

·内存列存储（in-memory columnar storage）：内存列存储意味着，Spark SQL的数据，不是使用Java对象的方式来进行存储，而是使用面向列的内存存储的方式来进行存储。也就是说，每一列作为一个数据存储的单位。从而大大优化了内存使用的效率。采用了内存列存储之后，减少了对内存的消耗，也就避免了回收大量数据的性能开销。

·字节码生成技术（byte-code generation）：Spark SQL在其catalyst模块的expressions中增加了codegen模块，对于SQL语句中的计算表达式，比如select num + num from t这种的sql，就可以使用动态字节码生成技术来优化其性能。

·Scala代码编写的优化：对于Scala代码编写中，可能会造成较大性能开销的地方，需要重写，使用更加复杂的方式，来获取更好的性能。比如Option样例类、for循环、map/filter/foreach等高阶函数，以及不可变对象，都改成了用null、while循环等来实现，并且重用可变的对象。

### MLBase/MLlib

MLBase是Spark生态圈里的一部分，专门负责机器学习这块，它的目标是降低机器学习的门槛。MLBase由四部分组成：MLRuntime、MLlib、MLI、ML Optimizer。

（1）MLRuntime：是Spark Core提供的分布式内存计算框架，运行由Optimizer优化过的算法进行数据计算，并且输出结果。

（2）MLlib：是Spark实现一些机器学习算法和实用程序，比如说分类、回归、聚类、降维、协同过滤还有底层的优化方面。

（3）ML Optimizer：它会把用户的数据用它认为最适合的内部已经实现好的机器学习算法以及相关参数处理，并且返回模型或者其他帮助分析的结果。

（4）MLI：它是一个特征抽取和高级ML编程抽象算法实现的API的平台。

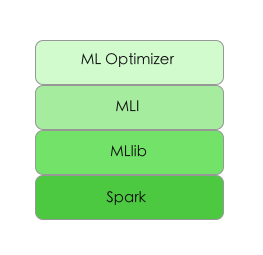


图 1-7 MLBase/MLilib结构图

MLBase核心是优化器（ML Optimizer），它会把复杂的学习任务转化成复杂的学习计划，且最终得出最优解。

### GraphX

GraphX是Spark中的一个新组件，用于图形和图形并行计算。 在高层次上，GraphX通过引入一个新的图抽象来扩展Spark RDD，这个抽象是有向图，且每个图的属性都附加在每个顶点和边上。 为了支持图计算，GraphX公开了一组基本运算符（例如subgraph，joinVertices和aggregateMessages）以及Pregel API的优化变体。 此外，GraphX还包含越来越多的图形算法和构建器，以简化图形分析任务。GraphX最大的优势是它在Spark的基础上提供了一栈式数据解决方案，便于高效完成图计算的完整的流水作业。

接下来介绍GraphX的核心抽象，Resilient Distributed Property Graph，它是点和边都带有属性的有向多重图，且有Table与Graph两种视图，但只需要一份物理存储，两种视图都有自己独有的操作符，从而获得了灵活操作和执行效率。

GraphX设计的关键：

·对Graph视图操作会转化成关联的Table视图的RDD操作，因此，Graph具有RDD的三个特性：不变性、分布性、容错性。

·两种视图底层共用物理数据由RDD[Vertex-Partition]和RDD[EdgePatition]两个RDD组成，点和边是VertexPartition/EdgePaatition在内部存储一个带索引结构的分片数据块形式存储 。不变的索引结构在RDD转换过程是共用的，降低计算和存储的开销。

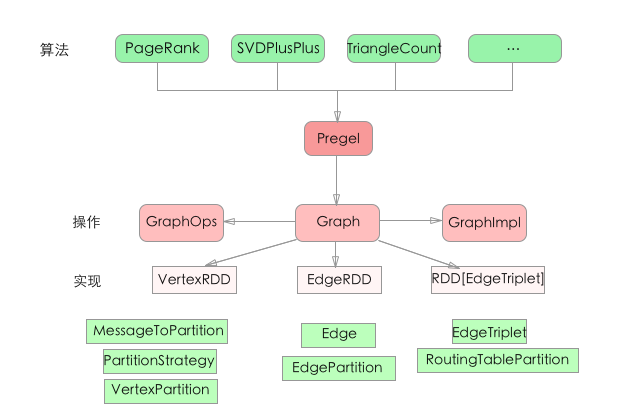


图 1-8 GraphX框架图

### SparkR

SparkR是一个R语言包，为Apache提供了轻量的前端。SparkR提供了Spark弹性分布式数据集（RDD）的API，用户可以在集群通过R shell交互性地运行Job。例如，我们可以在HDFS上读取或者写入文件，也可以使用lapply函数进行方法调用，定义对应每一个RDD元素的运算。

Spark具有快速、可扩展、交互的特点，R具有统计、绘图的优势，R和Spark的有效结合，解决了R语言中无法级联扩展的难题，同时，也在很大程度上丰富了Spark在机器学习方面能使用的Lib库。除了常见的RDD函数式算子Reduce、reduceByKey、GroupByKey等，SparkR也支持利用lapplyWithPartition对每个RDD的分区进行操作。SparkR也支持常见的闭包功能（用户定义的函数中所引用到的变量会自动被发送到集群中的其他机器上）。SparkR的工作原理如下图所示，首先加载R方法包和rJava包，然后用SparkR初始化SparkContext。

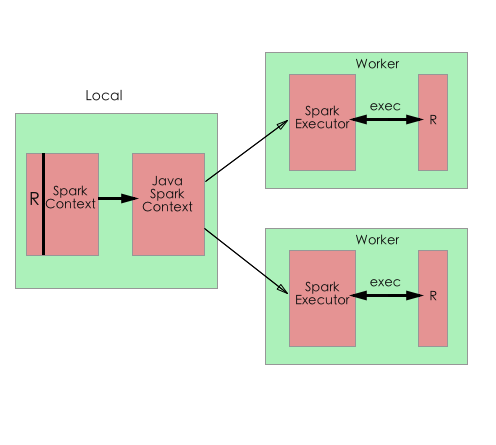


图 1-9 Spark工作原理图

## 开发语言的选择

Spark的开发语言有多种，Scala、Java和Python，所以，对于不同语言习惯的开发者，可以使用不同的语言。下面有不同语言的Spark编程示例，可以让大家对Spark有一个初步的认识。

1. 示例筛选文件中包含特定单词的行，以“Scala”这个单词为例，代码如下所示。

**Python版本例子：**

lines = sc.textFile(“README.md”)

#此处括号内为要获取的文件所在目录，README文件一般在下载的Spark文件的目录下。

ScalaLines = lines.filter(lambda line: “Scala” in line) #从每行中筛选含有Scala的行

ScalaLines.first() #输出第一个含有”Scala”的行

**Scala版本例子：**

val lines = sc.textFile(“README.md”)

val ScalaLines = lines.filter(line => line.contains(“Scala”))

ScalaLines.first()

**Java版本例子：**

JavaRDD<String> lines = sc.textFile(“README.md”);

JavaRDD<String> ScalaLines = lines.filter(line -> line.contains("Scala"));

ScalaLines.first();

它们得到的结果统一为：high-level APIs in Scala, Java, Python, and R, and an optimized engine that。

1. 初始化SparkContext的代码示例

**Python版本例子：**

from pyspark import SparkConf,SparkContext

Conf = SparkConf().setMaster(“local”).setAppName(“My First App”)

#设置url为本地单线程，程序名为“My First App”

sc = SparkContext(conf = conf)

**Scala版本例子：**

import org.apache.spark.SparkConf

import org.apache.spark.SaprkContext

Import org.apache.spark.SparkContext.\_

val conf = new SparkConf().setMaster(“local”).setAppName(“My First App”)

val sc = new SparkContext(conf)

**Java版本例子：**

import org.apache.spark.SparkConf;

import org.apache.spark.api.java.JavaSparkContext;

SparkConf conf = new SparkConf().setMaster(“local”).setAppName(“My First App”)

JavaSparkContext sc = new JavaSparkContext(conf);

1. 单词统计应用示例，该应用是以“README.md”为例，将这个文件内所有单词以空格为切分，然后统计每个单词的出现次数，保存为文本形式。

**Python版本例子：**

from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("local").setAppName("wordcount")

sc = SparkContext(conf=conf) #SparkContext初始化

lines = sc.textFile("README.md") #括号中为文件路径

words = lines.flatMap(lambda line: line.split(“ “)) #用空格拆分元素,使其切成一个个单词

counts = words.map(lambda x:(x,1)).reduceByKey(lambda x,y: x+y) #转换成键值对，且相同键值组合计数

counts.saveAsTextFile(outputFile) #保存到括号中的路径

**Scala版本的例子：**

import org.apache.spark.{SparkConf,SparkContext}

//建立的WordCount对象，以及定义main函数

object WordCount {

def main(args: Array[String]): Unit = {

val conf = new SparkConf().setMaster("local").setAppName("wordcount")

val sc = new SparkContext(conf)

val lines = sc.textFile("README.md")

val words = lines.flatMap(line => line.split(" "))

val count = words.map(word => (word,1)).reduceByKey{case(x,y) => x+y}

val output = count.saveAsTextFile(outputFile)

}

}

**Java版本的例子：**

import java.util.Arrays;

import java.util.Iterator;

import org.apache.spark.SparkConf;

import org.apache.spark.api.java.JavaPairRDD;

import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;

import org.apache.spark.api.java.function.FlatMapFunction;

import org.apache.spark.api.java.function.Function2;

import org.apache.spark.api.java.function.PairFunction;

import org.apache.spark.api.java.function.VoidFunction;

import scala.Tuple2;

public class SparkWordCount {

public static void main(String[] args) {

SparkConf conf = new SparkConf();

conf.setAppName("wordcount");

conf.setMaster("local");

// 第二步：创建JavaSparkContext对象，SparkContext是Spark的所有功能的入口

JavaSparkContext sc = new JavaSparkContext(conf);

// 第三步：创建一个初始的RDD

// SparkContext中，用于根据文件类型的输入源创建RDD的方法，叫做textFile()方法

JavaRDD<String> lines = sc.textFile("/home/xiongfan123/spark-2.3.0-bin-hadoop2.7/README.md");

// 第四步：对初始的RDD进行transformation操作，也就是一些计算操作

// 首先把单词用空格拆开

JavaRDD<String> words = lines.flatMap(new FlatMapFunction<String, String>() {

private static final long serialVersionUID = 1L;

@Override

public Iterator<String> call(String line) throws Exception {

return Arrays.asList(line.split(" ")).iterator();

}

});

// 将每一个单词，映射为（单词，1）的这种格式

JavaPairRDD<String, Integer> pairs = words.mapToPair(new PairFunction<String, String, Integer>() {

private static final long serialVersionUID = 1L;

@Override

public Tuple2<String, Integer> call(String word) throws Exception {

return new Tuple2<String, Integer>(word, 1);

}

});

// 以单词作为key，统计每个单词出现的次数

JavaPairRDD<String, Integer> wordCounts = pairs.reduceByKey(new Function2<Integer, Integer, Integer>() {

private static final long serialVersionUID = 1L;

@Override

public Integer call(Integer v1, Integer v2) throws Exception {

return v1 + v2;

}

});

// 用action操作foreach来执行

wordCounts.foreach(new VoidFunction<Tuple2<String,Integer>>() {

private static final long serialVersionUID = 1L;

@Override

public void call(Tuple2<String, Integer> wordCount) throws Exception {

System.out.println(wordCount.\_1 + "------" + wordCount.\_2+"times.");

}

});

sc.close();

}

}

以上是用三种语言开发Spark的示例，大家可以观察上面的程序语法，从自己熟悉的语言中开始接触Spark。

## 小结

众所周知，Spark在大数据中扮演着十分重要的角色，它继承了分布式并行计算的优点，并改进MapReduce的缺陷。本书在第一章初步介绍了Spark的发展历程以及它的重要扩展，以及它的基本架构，接下来，本书还会针对这些内容进行详细的解读。

## 习题

1. 结合你对本章的理解，说说Spark有哪些优点。
2. 初略地介绍一下Spark的组件。
3. 详细介绍独立集群模式。
4. 解释1.4中Scala和Java语言写的Spark示例。

# 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）。

## Local模式

### 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）。

### 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)

### 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

## 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。

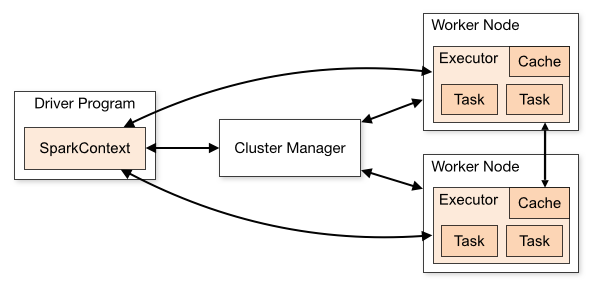


图 2-1 集群模式图

在分布式环境下，Spark集群是采用主/从结构，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。

### 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，没有AM和NM的概念，也没有RM的概念，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。

### 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

Yourslave1

Yourslave2

Yourslave3

Yourslave4

Yourslave5

......

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 |

### 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

1. 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

1. 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。

1. 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。

## 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。

### YARN模式

#### 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

（Resource Manager，RM）和针对个人应用的Master（Application Master，AM），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。

#### 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。

#### 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。

1. 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。

### Mesos模式

#### 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。

#### 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。

1. 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

1. 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

1. 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。

#### 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分配资源，第二层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 |

## 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。