# 第一章：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

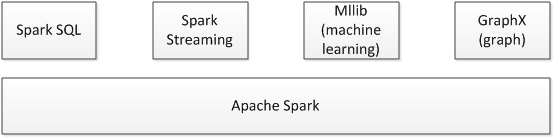


图 1-1 Spark技术堆栈

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

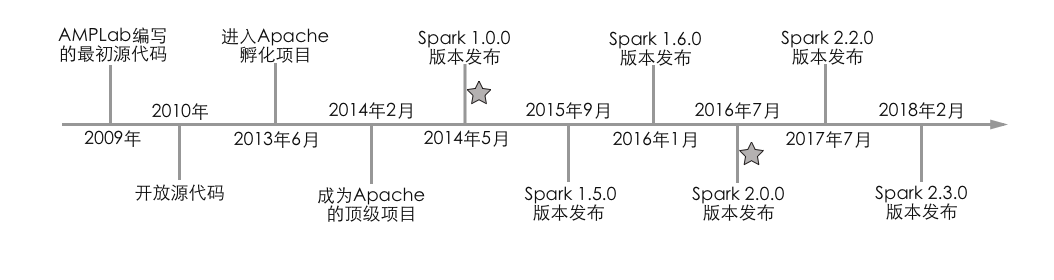


图 1-2 Spark演进时间轴

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。

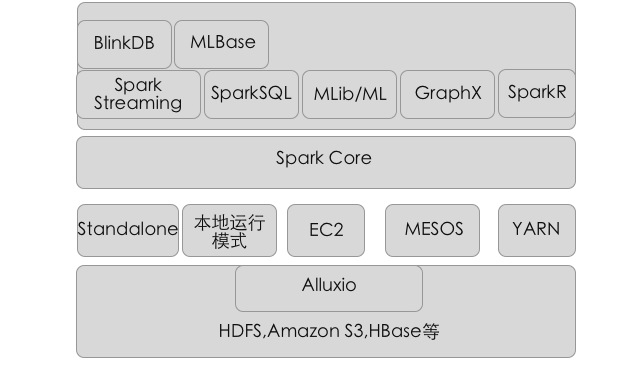


图 1-3 Spark生态系统

### 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。

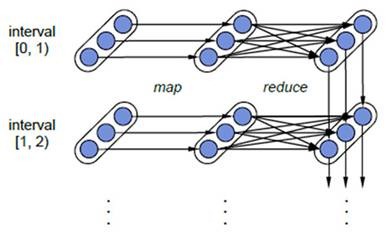


图 1-5 Spark Streaming中RDD的血统关系图

**·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。

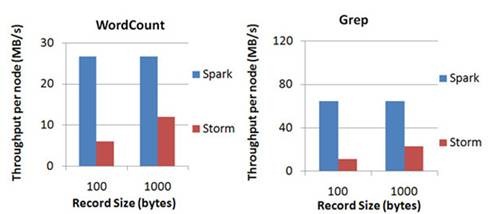


图 1-6 Spark Streaming与Storm吞吐量比较图

### 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。

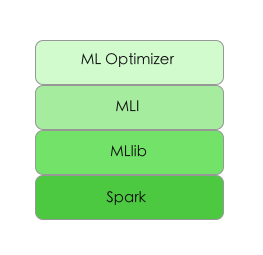


图 1-7 MLBase/MLilib结构图

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。

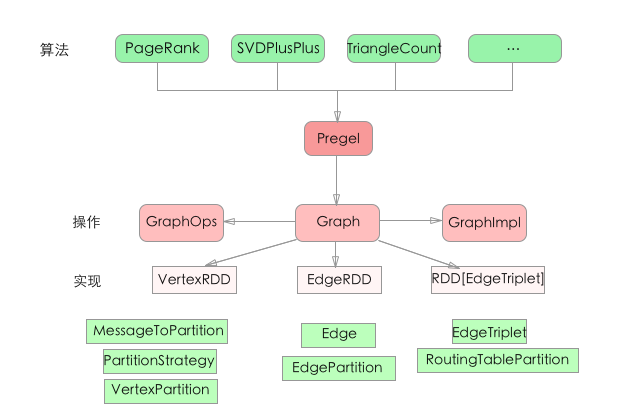


图 1-8 GraphX框架图

### 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。

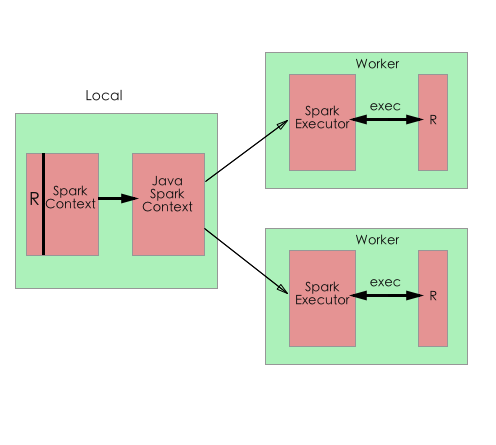


图 1-9 Spark工作原理图

## Spark的基本架构

### Spark架构中的组件介绍

1. Client：提交应用的客户端 。
2. Driver：客户端驱动程序，也可以理解为客户端的应用程序，执行Application中的main函数，并创建SparkContext。
3. ClusterManager：在YARN模式中为资源管理器，在Standalone模式中为Master（主节点），控制整个集群。
4. Worker：也称从节点。对于Spark应用程序来说，由集群管理器分配得到资源的Worker节点主要负责的工作是创建Executor，将资源和任务进一步分配给Executor，同步资源信息给Cluster Manager。
5. Executor：在计算节点执行任务的组件。
6. SparkContext：应用的上下文，控制应用的生命周期。
7. Job（作业）：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。
8. Stage（阶段）：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。
9. Task（任务）：被送到某个Executor上的工作任务。
10. DAG Scheduler：根据应用构建基于Stage的DAG，并将Stage提交给Task Scheduler。
11. Task Scheduler：将Task分发给Executor执行。
12. SparkEnv：线程级的上下文，存储运行时的重要组件的应用，具体内容包括：SparkConf（存储配置信息）、BroadcastManager（负责广播变量的控制及元信息的存储）、BlockManager（负责Block的管理、创建和查找）、MetricsSystem（监控运行时的性能指标）、MapOutputTracker（负责shuffle元信息的存储）。

### Spark架构介绍

Spark架构采用分布式计算中的Master-Slave模型。集群中的Master进程的节点称为Master，同样，集群中含有Worker进程的节点为Slave。Master负责控制整个集群的运行；Worker节点相当于分布式系统中的计算节点，它接收Master节点指令并返回计算进程到Master；Executor负责任务的执行；Client是用户提交应用的客户端；Driver负责协调提交后的分布式应用。根据下面的架构图，我们能看到，在Spark应用执行的过程中，Driver与Worker是相互对应的。Driver是应用逻辑的起点，负责Task任务的分发和调度；Worker负责管理计算节点，并创建Exector来并行处理Task任务。Task执行过程中所需的文件和包由Driver序列化后传输给对应的Worker节点，Executor对相应分区的任务进行处理。

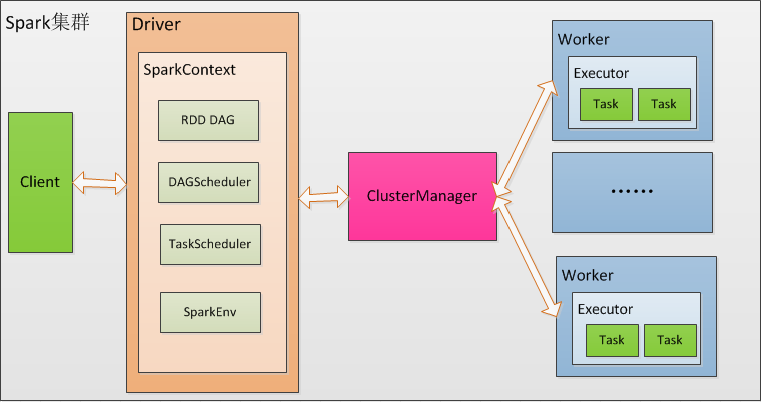


图 1-10 Spark架构图

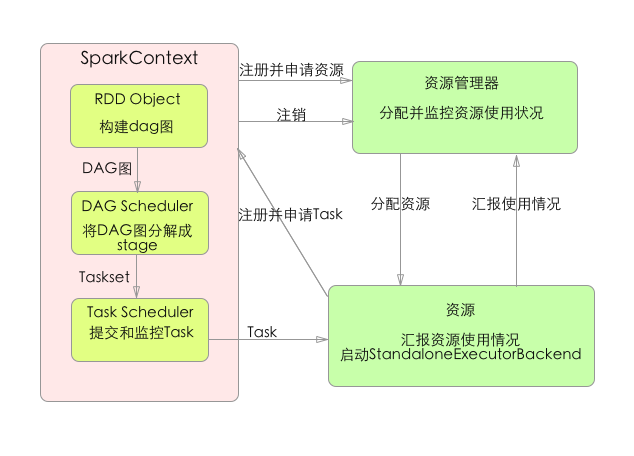


图 1-11 Spark流程图

从Spark流程图中，我们可以看出Spark的工作流程：首先，构建Spark Application的运行环境，启动SparkContext；然后SparkContext向资源管理器（可以是Standalone，Mesos，Yarn）申请运行Executor资源，并启动StandaloneExecutorbackend；接下来，Executor向SparkContext申请Task；接着SparkContext将应用程序分发给Executor，由SparkContext构建成DAG图，将DAG图分解成Stage、将Taskset发送给Task Scheduler，最后由Task Scheduler将Task发送给Executor运行；最后，Task在Executor上运行，运行完释放所有资源。

Spark运行有以下特点：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Task。无论是从调度角度看（每个Driver调度他自己的任务），还是从运行角度看（来自不同Application的Task运行在不同JVM中），这种Application隔离机制是有优势的，当然这样意味着Spark Application不能跨应用程序共享数据，除非将数据写入外部存储系统；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；Task采用了数据本地性和推测执行的优化机制。

### spark运行模式

Spark的运行模式多种多样，灵活多变，部署在单机上时，既可以用本地模式运行，也可以用伪分布模式运行，而当以分布式集群的方式部署时，也有众多的运行模式可供选择，这取决于集群的实际情况，底层的资源调度即可以依赖外部资源调度框架，也可以使用Spark内建的Standalone模式。对于外部资源调度框架的支持，目前的实现包括相对稳定的Mesos模式，以及hadoop YARN模式，本地模式常用于本地开发测试。

1. 本地（Local）运行模式

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。在下一小节的初始化SparkContext的例子中，是通过在SparkConf对象的setMaster(“local”)来指定本地运行模式，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。

1. 独立（Standalone）集群运行模式

Standalone模式使用Spark自带的资源调度框架，采用Master/Slaves的典型架构，其组成有Client节点、Master节点、Worker节点。Driver不仅可以在Master节点运行，也可以在本地客户端上运行。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。

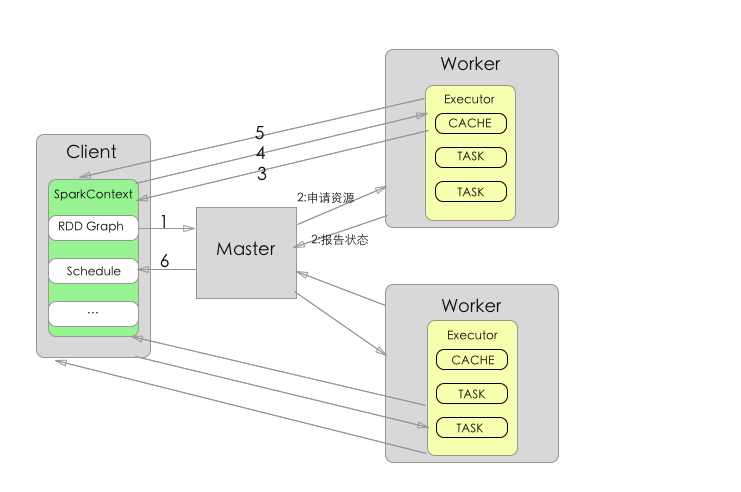


图 1-12 独立集群运行模式图

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. 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

1. 伪分布（Local-Cluster）模式

伪分布模式是指在一台机器上模拟集群运行。独立集群运行模式的Master、Worker和SparkContext是在不同的节点上，而伪分布运行模式这些都是 在一台机器上。所以伪分布运行模式运行流程与独立运行模式相同，区别在于伪分布模式运行在同一个节点中。

1. YARN运行模式

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）。

首先我们大致介绍一下Yarn-Client模式。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。

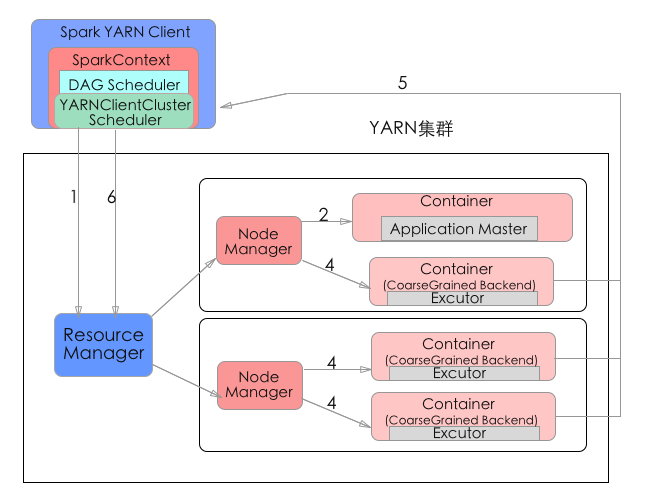


图 1-13 YARN-Client模式流程图

其工作步骤的流程如下：

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。

接下来我们介绍一下YARN-Cluster模式。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。

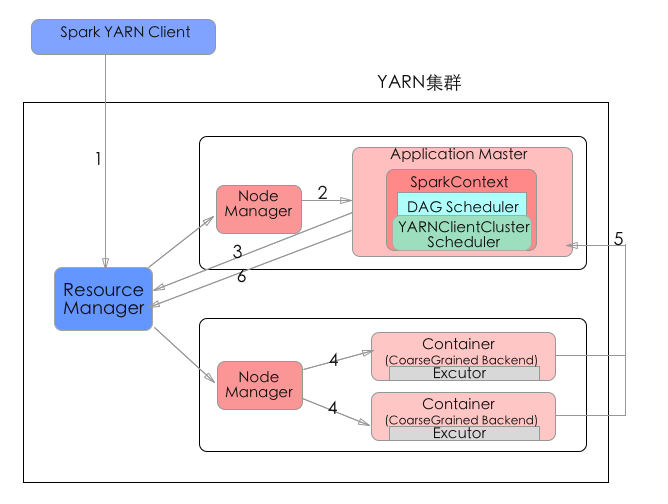


图 1-14 YARN-Cluster模式流程图

其工作步骤的流程如下：

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。

Spark Client 和 Spark Cluster的区别：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。

## 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 = input.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 scala.Tuple2;

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 java.util.Arrays;

import java.util.List;

import java.util.regex.Pattern;

public class WordCount {

public static void main(String[] args) {

String logFile = "README.md"; //设置路径

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

JavaSparkContext sc = new JavaSparkContext(conf);

JavaRDD<String> lines = sc.textFile(logFile);

//用空格拆分元素,使其切成一个个单词

JavaRDD<String> words = lines.flatMap(

new FlatMapFunction<String, String>() {

public Iterable<String> call(String s) throws Exception {

return Arrays.asList(s.split(" "));

}});

//转换成键值对，且相同键值组合计数

JavaPairRDD<String, Integer> counts = word.mapToPair(

new PairFunction<String, String, Integer>(){

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

return new Tuple2(s, 1);

}}).reduceByKey(new Function2<Integer, Integer, Integer>(){

public Integer call(Integer x, Integer y){ return x+y;}});

counts.saveAsTextFile(outputFile);

}

}

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。

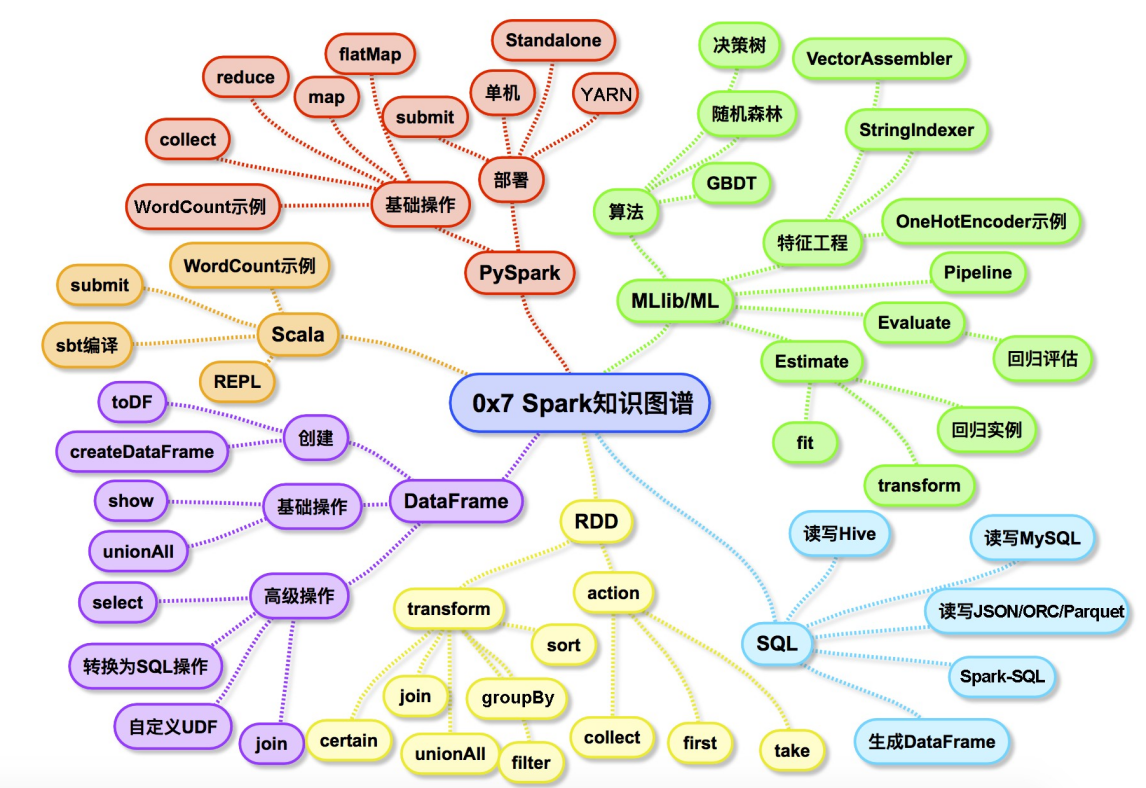


图 1-10 Spark知识图谱

## 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。