前世今生

Spark是一个快速通用的用于大规模数据处理引擎

Spark使用**Spark RDD**、**Spark SQL**、**Spark Streaming**、**MLlib**、**GraphX**成功解决了大数据领域中，离线批处理、交互式查询、实时流计算、机器学习与图计算等最重要的任务和问题。

基于内存进行计算

Spark最重要的特点，就是基于内存进行计算，从而让它的速度可以达到**MapReduce**、**Hive**的数倍甚至数十倍！

前世今生

时间线

2009年，**Spark**诞生于伯克利大学的**AMPLab**实验室。最出**Spark**只是一个实验性的项目，代码量非常少，属于轻量级的框架。

2010年，伯克利大学正式开源了**Spark**项目。

2013年，**Spark**成为了**Apache**基金会下的项目，进入高速发展期。第三方开发者贡献了大量的代码，活跃度非常高。

2014年，**Spark**以飞快的速度称为了**Apache**的顶级项目。

2015年**~**，**Spark**在国内**IT**行业变得愈发火爆，大量的公司开始重点部署或者使用**Spark**来替代**MapReduce**、**Hive**、**Storm**等传统的大数据计算框架。

前世今生

四大特性

快速性：**spark**使用先进的**DAG**调度器、快速的优化器和物理执行引擎来实现高性能的批量和流式数据处理。

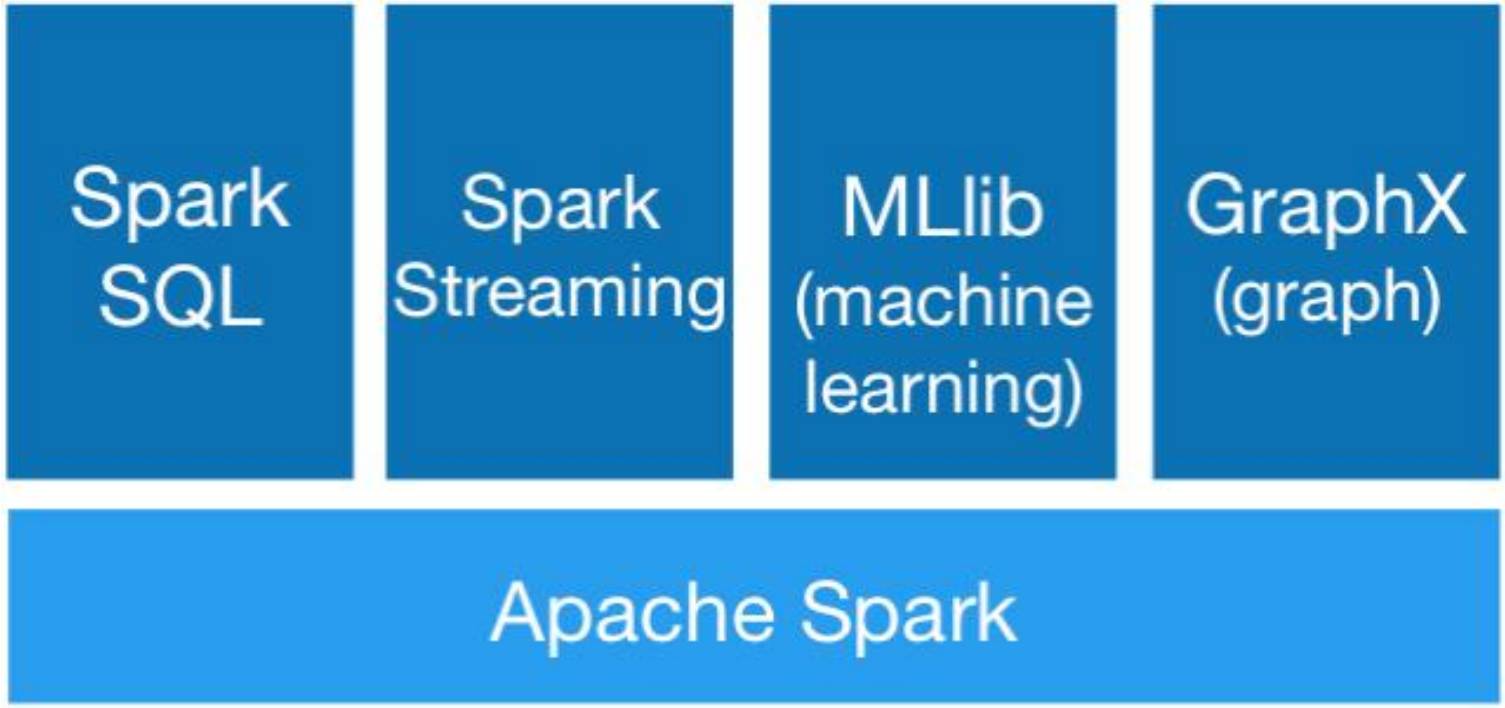
易用性：**spark**提供超过**80**个高级别的算子操作，使构建并行应用非常容易。另外，**spark**支持多种客户端编程语言（**Java**、**Scala**、**Python**、**R**、**SQL**），使用门槛低。

通用性：**spark**提供了一个生态栈，包括离线、实时、交互式、机器学习、图计算多种应用功能。

到处运行：**spark**支持多种运行方式（**standalone**、**YARN**、**Mesos**、**Kubernetes**、**EC2**）；**spark**支持多种数据源（**HDFS**、**Cassandra**、**HBase**、**Hive**及其他数据源）。

架构设计

架构图


Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark

架构设计

架构介绍

Spark Core：适用于离线计算。实现**Spark**的基本功能，其他**Spark**的库都是构建在**Spark Core**之上的。

Spark SQL：适用于交互式计算。用于结构化数据处理。

SparkStreaming：适用于实时流式计算。对实时数据流进行处理和控制。

MLlib：适用于机器学习场景。一个常用机器学习算法库，这个库包含可扩展的学习算法，比如分类、回归等需要对大量数据集进行迭代的操作。

GraphX：适用于图计算场景。一个控制图、并行图操作和计算的一组算法和工具的集合。

基本概念

基本概念

Application：用户编写的**Spark**应用程序。

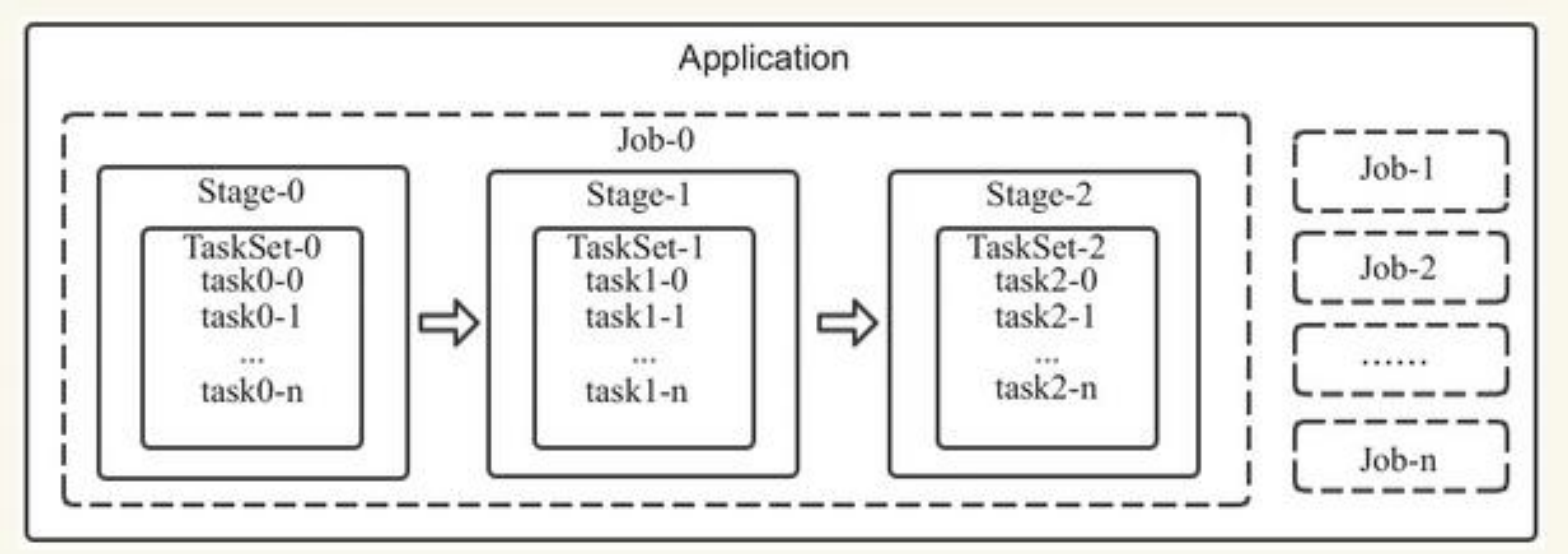
Job: 提供给**Spark**运行的作业，一个**Application**中往往会产生多个**Job**。

Stage: 每个**Job**会被拆分成多组**Task**， 作为一个**TaskSet**， 其名称为**Stage**。**Stage**有非最终的**Stage**（**Shuffle Map Stage**）和最终的**Stage**（**Result Stage**）两种。

Task: 真正执行的工作单元，多个**Task**组成一个**Stage**。

基本概念

基本概念


Application
Job-0
Stage-0
TaskSet-0
task0-0
task0-1
...
task0-n
Stage-1
TaskSet-1
task1-0
task1-1
...
task1-n
Stage-2
TaskSet-2
task2-0
task2-1
...
task2-n
Job-1
Job-2
.....
Job-n

基本概念

基本概念

Cluster Manager：在集群上获取资源的外部服务。

Worker: 集群中任何可以运行**Application**代码的节点。

Executor: 某个**Application**运行在**worker**节点上的一个进程， 该进程负责运行某些**Task**。

Driver：准备**Spark**应用程序的运行环境，负责*进行资源申请、任务的分配和监控*等。

DAGScheduler：根据**Job**构建**DAG**图，将**Job**拆分成多个**Stage**并提交给**TASkScheduler**。

TaskScheduler：将**Stage**拆分成多个**Task**并提交给**worker**运行，**Executor**运行什么**Task**就是在此处分配的。

运行模式

运行模式

Local
只有**Driver**，没有**Master**和**Worker**。
执行任务的**Executor**与**Driver**在同一个**JVM**进程中

Local-Cluster
Driver、**Master**和**Worker**运行在同一个**JVM**进程中
每个**Worker**可以有多个**Executor**，每个**Executor**都是一个**JVM**进程

Standalone
Driver在集群之外，可以是任意的客户端程序
Master部署于单独的进程，甚至在单独的机器上。可以有多个，但只能有一个处于激活状态
Worker部署于单独的进程，推荐在单独的机器上部署。

运行模式

运行模式

YARN
yarn-cluster：适用于生产环境
yarn-client：适用于交互、调试，希望立即看到**app**的输出

yarn-cluster和**yarn-client**模式的区别其实就是**Application Master**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。

yarn-cluster模式不适合运行交互类型的作业。而**yarn-client**模式下，**Application Master**仅仅向**YARN**请求**executor**，**client**会和请求的**container**通信来调度工作，**Client**不能离开。

运行模式

运行模式

Mesos
运行模式类似于**YARN**，分为**Client**和**Cluster**两种模式
调度器分为粗粒度（默认）和细粒度（不推荐）

安装配置

下载
http://spark.apache.org/downloads.html

解压
tar zxf /opt/software/spark-2.4.0-bin-hadoop2.7.tar.gz –C /opt/module/

配置**spark-env.sh**
export JAVA_HOME=/opt/module/jdk1.8.0_171
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop

启动
bin/start-all.sh

Shell操作

案例

```
scala> val textFile = spark.read.textFile("README.md")

scala> val wordCounts = textFile.flatMap(line =>
line.split(" ")).groupByKey(identity).count()

scala> wordCounts.collect()
```