1. Spark的产生背景

1.1. MapReduce的发展

1.1.1. MRv1的缺陷

早在 Hadoop-1.x 版本，当时采用的是 MRv1 版本的 MapReduce 编程模型。MRv1版本的实现都封装在 org.apache.hadoop.mapred 包中，MRv1 的 Map 和 Reduce 是通过接口实现的。MRv1包括三个部分：

```
运行时环境（JobTracker和TaskTracker）
编程模型（MapReduce）
数据处理引擎（MapTask和ReduceTask）
```

MRv1 存在以下不足：

可扩展性差

```
在运行时，JobTracker既负责资源管理又负责任务调度，当集群繁忙时，JobTracker很容易成为瓶颈，最终导致它的可扩展性问题。
```

可用性差

```
采用了单节点的Master，没有备用Master及选举操作，这导致一旦Master出现故障，整个集群将不可用。
```

资源利用率低

TaskTracker使用“slot”等量划分本节点上的资源量。“slot”代表计算资源（CPU、内存等）。一个Task获取到一个slot后才有机会运行，Hadoop调度器负责将各个TaskTracker上的空闲slot分配给Task使用。一些Task并不能充分利用slot，而其他Task也无法使用这些空闲的资源。slot分为Map slot和Reduce slot两种，分别供MapTask 和ReduceTask 使用。有时会因为作业刚刚启动等原因导致MapTask很多，而Reduce Task任务还没有调度的情况，这时Reduce slot也会被闲置。

不能支持多种MapReduce框架

无法通过可插拔方式将自身的MapReduce框架替换为其他实现，如Spark、Storm等。

1.1.2. MRv2的缺陷

Apache 为了解决 MRv1 中的缺陷，对 Hadoop 进行了升级改造。MRv2 就诞生了。

MRv2 中，重用了 MRv1 中的编程模型和数据处理引擎。但是运行时环境被重构了。JobTracker 被拆分成了通用的多个组件：

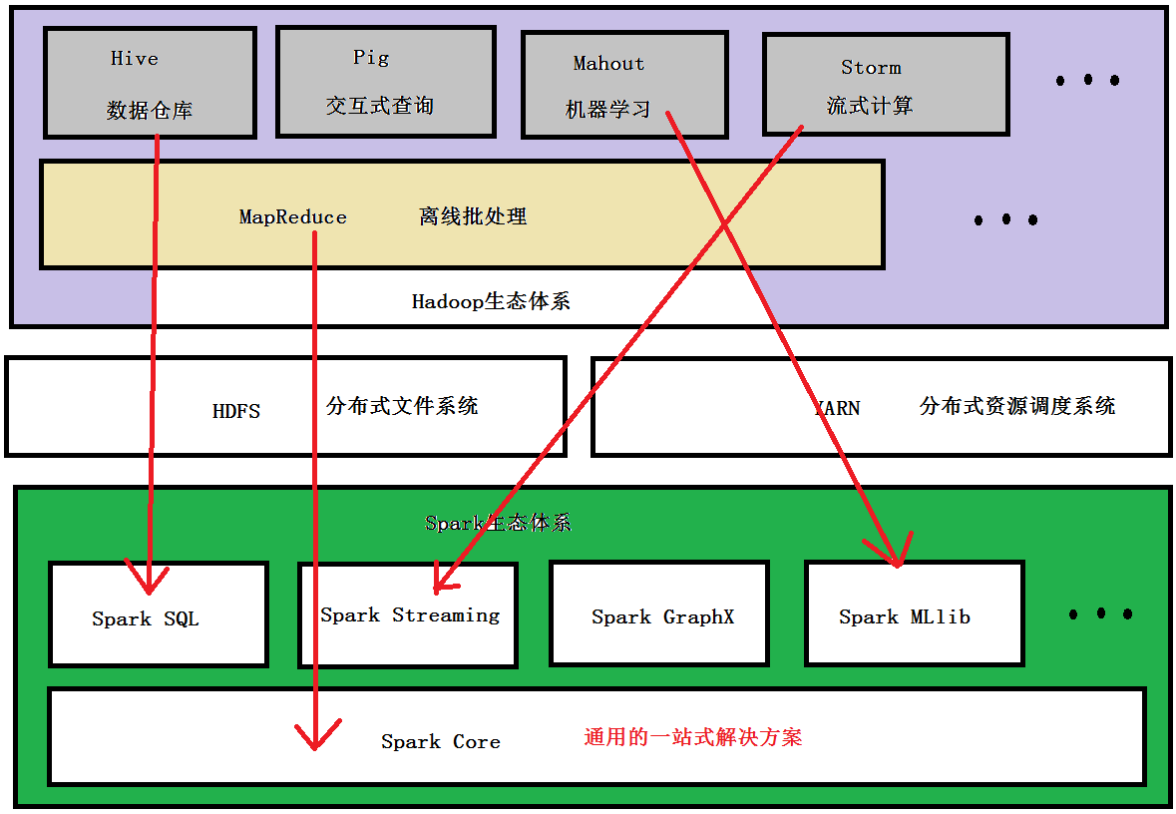
资源调度平台（ResourceManager，简称RM）、
节点管理器（NodeManager）、
负责各个计算框架的任务调度模型（ApplicationMaster，简称AM）。

ResourceManager 依然负责对整个集群的资源管理，但是在任务资源的调度方面只负责将资源封装为Container分配给 ApplicationMaster 的一级调度，二级调度的细节将交给 ApplicationMaster 去完成，这大大减轻了ResourceManager 的压力，使得 ResourceManager 更加轻量。NodeManager 负责对单个节点的资源管理，并将资源信息、Container 运行状态、健康状况等信息上报给ResourceManager。ResourceManager 为了保证Container 的利用率，会监控 Container，如果Container 未在有限的时间内使用，ResourceManager 将命令NodeManager 杀死 Container，以便于将资源分配给其他任务。MRv2 的核心不再是 MapReduce 框架，而是YARN。在以 YARN 为核心的MRv2 中，MapReduce 框架是可插拔的，完全可以替换为其他分布式计算模型实现，比如Spark、Storm等。

Hadoop MRv2 虽然解决了 MRv1 中的一些问题，但是由于对 HDFS 的频繁操作（包括计算结果持久化、数据备份、资源下载及 Shuffle 等）导致磁盘 I/O 成为系统性能的瓶颈，因此只适用于离线数据处理或批处理，而不能支持对迭代式、交互式、流式数据的处理。

重点概念：离线处理，批处理，实时处理，流式处理

1.2. Spark的产生



Spark 看到 MRv2 的问题，对 MapReduce 做了大量优化，总结如下：

1、减少磁盘I/O

随着实时大数据应用越来越多，Hadoop作为离线的高吞吐、低响应框架已不能满足这类需求。Hadoop MapReduce 的 map 端将中间输出和结果存储在磁盘中，reduce 端又需要从磁盘读写中间结果，势必造成磁盘IO成为瓶颈。Spark 允许将 map 端的中间输出和结果存储在内存中，reduce 端在拉取中间结果时避免了大量的磁盘I/O。Hadoop YARN 中的 ApplicationMaster 申请到 Container后，具体的任务需要利用NodeManager 从 HDFS 的不同节点下载任务所需的资源（如Jar包），这也增加了磁盘I/O。Spark 将应用程序上传的资源文件缓冲到 Driver 本地文件服务的内存中，当 Executor 执行任务时直接从 Driver 的内存中读取，也节省了大量的磁盘 I/O。

2、增加并行度

由于将中间结果写到磁盘与从磁盘读取中间结果属于不同的环节，Hadoop 将它们简单的通过串行执行衔接起来。Spark 把不同的环节抽象为 Stage，允许多个 Stage 既可以串行执行，又可以并行执行。

3、避免重新计算

当Stage中某个分区的Task执行失败后，会重新对此Stage调度，但在重新调度的时候会过滤已经执行成功的分区任务，所以不会造成重复计算和资源浪费。

4、可选的Shuffle和排序

Hadoop MapReduce在Shuffle之前有着固定的排序操作（只能对key排字典顺序），而Spark则可以根据不同场景选择在map端排序或者reduce端排序。

5、灵活的内存管理策略：

Spark将内存分为堆上的存储内存、堆外的存储内存、堆上的执行内存、堆外的执行内存4个部分。Spark既提供了执行内存和存储内存之间是固定边界的实现，又提供了执行内存和存储内存之间是“软”边界的实现。Spark默认使用“软”边界的实现，执行内存或存储内存中的任意一方在资源不足时都可以借用另一方的内存，最大限度的提高资源的利用率，减少对资源的浪费。Spark由于对内存使用的偏好，内存资源的多寡和使用率就显得尤为重要，为此Spark的内存管理器提供的Tungsten实现了一种与操作系统的内存Page非常相似的数据结构，用于直接操作操作系统内存，节省了创建的Java对象在堆中占用的内存，使得Spark对内存的使用效率更加接近硬件。Spark会给每个Task分配一个配套的任务内存管理器，对Task粒度的内存进行管理。Task的内存可以被多个内部的消费者消费，任务内存管理器对每个消费者进行Task内存的分配与管理，因此Spark对内存有着更细粒度的管理。

基于以上所列举的优化，Spark官网声称性能比Hadoop快100倍。即便是内存不足需要磁盘I/O时，其速度也是Hadoop的10倍以上。那么 Spark 会取代 Hadoop 么?

Spark是MapReduce的替代方案，而且兼容HDFS、Hive，可融入Hadoop的生态系统，以弥补MapReduce的不足。

2. Spark概念

官网：http://spark.apache.org/

```
Apache Spark is a fast and general-purpose cluster computing system. It provides
high-level APIs in Java, Scala, Python and R, and an optimized engine that
supports general execution graphs. It also supports a rich set of higher-level
tools including Spark SQL for SQL and structured data processing, MLlib for
machine learning, GraphX for graph processing, and Spark Streaming.
```

Spark是一种快速、通用、可扩展的大数据分析引擎
2009年诞生于加州大学伯克利分校AMP实验室
2010年开源
2013年6月成为Apache孵化项目
2014年2月成为Apache顶级项目
项目代码使用Scala语言进行编写

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 SparkCore 为核心，从 HDFS、Amazon S3 或者 HBase 等持久层读取数据，以 MESOS、YARN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 SparkShell/SparkSubmit 的批处理、SparkStreaming 的实时处理应用、SparkSQL 的结构化数据处理/即席查询、BlinkDB 的权衡查询、MLlib/MLbase 的机器学习、GraphX 的图处理和 PySpark 的数学/科学计算和SparkR的数据分析等等。

目前，Spark生态系统已经发展成为一个包含多个子项目的集合，其中包含SparkSQL、Spark Streaming、GraphX、MLib、SparkR等子项目，Spark是基于内存计算的大数据并行计算框架。除了扩展了广泛使用的MapReduce计算模型，而且高效地支持更多计算模式，包括交互式查询和流处理。Spark适用于各种各样原先需要多种不同的分布式平台的场景，包括批处理、迭代算法、交互式查询、流处理。通过在一个统一的框架下支持这些不同的计算，Spark使我们可以简单而低耗地把各种处理流程整合在一起。而这样的组合，在实际的数据分析过程中是很有意义的。不仅如此，Spark的这种特性还大大减轻了原先需要对各种平台分别管理的负担。Spark基于内存计算，提高了在大数据环境下数据处理的实时性，同时保证了高容错性和高可伸缩性，允许用户将Spark部署在大量廉价硬件之上，形成集群。Spark得到了众多大数据公司的支持，这些公司包括Hortonworks、IBM、Intel、Cloudera、MapR、Pivotal、百度、阿里、腾讯、京东、携程、优酷土豆。当前百度的Spark已应用于凤巢、大搜索、直达号、百度大数据等业务；阿里利用GraphX构建了大规模的图计算和图挖掘系统，实现了很多生产系统的推荐算法；腾讯 Spark 集群达到 8000 台的规模，是当前已知的世界上最大的Spark集群。

大一统的软件栈，各个组件关系密切并且可以相互调用，这种设计有几个好处：

> 1、软件栈中所有的程序库和高级组件都可以从下层的改进中获益。
>
> 2、运行整个软件栈的代价变小了。不需要运行5到10套独立的软件系统了，一个机构只需要运行一套软件系统即可。系统的部署、维护、测试、支持等大大缩减。
>
> 3、能够构建出无缝整合不同处理模型的应用。

3. Spark特点

Spark是一种快速、通用、可扩展的大数据分析引擎。

3.1. Speed：快速高效

随着实时大数据应用越来越多，Hadoop 作为离线的高吞吐、低响应框架已不能满足这类需求。Hadoop MapReduce 的 Job 将中间输出和结果存储在 HDFS 中，读写 HDFS 造成磁盘 IO 成为瓶颈。Spark 允许将中间输出和结果存储在内存中，节省了大量的磁盘IO。Apache Spark 使用最先进的DAG 调度程序，查询优化程序和物理执行引擎，实现批量和流式数据的高性能。同时 Spark 自身的 DAG 执行引擎也支持数据在内存中的计算。Spark官网声称性能比Hadoop快100倍。即便是内存不足需要磁盘IO，其速度也是Hadoop的10倍以上。

3.2. Ease of Use：简洁易用

Spark现在支持Java、Scala、Python和R等编程语言编写应用程序，大大降低了使用者的门槛。自带了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，可以非常方便的在这些Shell中使用Spark集群来验证解决问题的方法。

3.3. Generality：全栈式数据处理

Spark提供了统一的解决方案。Spark统一的解决方案非常具有吸引力，毕竟任何公司都想用统一的平台去处理遇到的问题，减少开发和维护的人力成本和部署平台的物力成本。

支持批处理（Spark Core）

Spark Core是Spark的核心功能实现，包括：SparkContext的初始化（DriverApplication通过SparkContext提交）、部署模式、存储体系、任务提交与执行、计算引擎等。Spark Core中还包含了对RDD (resilient distributed dataset，弹性分布式数据集)的 API定义。

支持交互式查询（Spark SQL）

Spark SQL 是Spark来操作结构化数据的程序包，可以让我们使用SQL语句的方式来查询数据，Spark支持多种数据源，包含Hive表，parquet以及JSON等内容。

支持流式计算（Spark Streaming）

与MapReduce只能处理离线数据相比，Spark还支持实时的流计算。Spark依赖Spark Streaming对数据进行实时的处理。

支持机器学习（Spark MLlib）

提供机器学习相关的统计、分类、回归等领域的多种算法实现。其一致的API接口大大降低了用户的学习成本。

支持图计算（Spark GraghX）

提供图计算处理能力，支持分布式， Pregel提供的API可以解决图计算中的常见问题。

支持Python操作--PySpark

Spark提供了一个Python_Shell，即pyspark，从而可以以交互的方式使用Python编写Spark程序。pyspark里最核心的模块是SparkContext（简称sc）,最重要的数据载体是RDD。RDD就像一个NumPy array或者一个Pandas Series，可以视作一个有序的item集合。只不过这些item并不存在driver端的内存里，而是被分割成很多个partitions，每个partition的数据存在集群的executor的内存中。

支持R语言--SparkR

从Spark1.4版本（2015.6）开始添加了SparkR API，它的出现能让数据分析师和数据科学家在Spark平台上使用R语言分析大型数据集以及交互式作业。

3.4. Runs Everywhere：兼容

可用性高。Spark 也可以不依赖于第三方的资源管理和调度器，它实现了 Standalone 作为其内置的资源管理和调度框架，这样进一步降低了 Spark 的使用门槛，使得所有人都可以非常容易地部署和使用 Spark，此模式下的Master 可以有多个，解决了单点故障问题。当然，此模式也完全可以使用其他集群管理器替换，比如 YARN、Mesos、Kubernetes、EC2等。

丰富的数据源支持。Spark除了可以访问操作系统自身的本地文件系统和HDFS之外，还可以访问Cassandra、HBase、Hive、Tachyon以及任何Hadoop的数据源。这极大地方便了已经使用HDFS、HBase的用户顺利迁移到Spark。

Spark支持的几种部署方案：

```
Standalone：Spark自己可以给自己分配资源（Master，Worker）
YARN：Spark可以运行在Hadoop的YARN上面
Mesos：Spark可以运行在Mesos里面（Mesos 类似于YARN的一个资源调度框架）
Kubernetes：Spark接收 Kubernetes的资源调度
```

4. Spark应用场景

目前大数据处理场景有以下几个类型：

```
1、复杂的批量处理（Batch Data Processing），偏重点在于处理海量数据的能力，至于处理速度可忍受，通常的时间可能是在数十分钟到数小时；
2、基于历史数据的交互式查询（Interactive Query），通常的时间在数十秒到数十分钟之间
3、基于实时数据流的数据处理（Streaming Data Processing），通常在数百毫秒到数秒之间
```

目前对以上三种场景需求都有比较成熟的处理框架：

```
第一种情况可以用Hadoop的MapReduce来进行批量海量数据处理
第二种情况可以Impala、Kylin进行交互式查询
第三中情况可以用Storm分布式处理框架处理实时流式数据
```

以上三者都是比较独立，各自一套维护成本比较高，而Spark的出现能够一站式平台满意以上需求。

```
第一种情况使用Spark Core解决
第二种情况使用Spark SQL解决
第三种情况使用Spark Streaming解决
```

通过以上分析，总结Spark场景有以下几个：

```
1、Spark是基于内存的迭代计算框架，适用于需要多次操作特定数据集的应用场合。需要反复操作的次数越多，所需读取的数据量越大，受益越大，数据量小但是计算密集度较大的场合，受益就相对较小
2、由于RDD的特性，Spark不适用那种异步细粒度更新状态的应用，例如web服务的存储或者是增量的web爬虫和索引。就是对于那种增量修改的应用模型不适合
3、数据量不是特别大，但是要求实时统计分析需求
```

典型行业应用场景：

```
1、Yahoo将Spark用在Audience Expansion中的应用，进行点击预测和即席查询等
2、淘宝技术团队使用了Spark来解决多次迭代的机器学习算法、高计算复杂度的算法等。应用于内容推荐、社区发现等
3、腾讯大数据精准推荐借助Spark快速迭代的优势，实现了在“数据实时采集、算法实时训练、系统实时预测”的全流程实时并行高维算法，最终成功应用于广点通PCTR投放系统上。
4、优酷土豆将Spark应用于视频推荐(图计算)、广告业务，主要实现机器学习、图计算等迭代计算。
5、……
```