哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

大数据技术与实践

李春山
哈尔滨工业大学（威海）计算学部

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Spark概述

提纲

- **Spark概述**
- **Spark生态系统**
- **Spark运行架构**
- **Spark SQL**
- **Spark的部署和应用方式**
- **Spark编程实践**

Spark

Spark概述

Spark简介

Scala简介

Spark与Hadoop的比较

Spark简介

- Spark最初由美国加州伯克利大学（UCBerkeley）的AMP实验室于2009年开发，是基于内存计算的大数据并行计算框架，可用于构建大型的、低延迟的数据分析应用程序

- 2013年Spark加入Apache孵化器项目后发展迅猛，如今已成为Apache软件基金会最重要的三大分布式计算系统开源项目之一（Hadoop、Spark、Storm）

- Spark在2014年打破了Hadoop保持的基准排序纪录
  - Spark/206个节点/23分钟/100TB数据
  - Hadoop/2000个节点/72分钟/100TB数据
  - Spark用十分之一的计算资源，获得了比Hadoop快3倍的速度

Spark简介

Spark具有如下几个主要特点：

•运行速度快：使用DAG执行引擎以支持循环数据流与内存计算

•容易使用：支持使用Scala、Java、Python和R语言进行编程，可以通过Spark Shell进行交互式编程

•通用性：Spark提供了完整而强大的技术栈，包括SQL查询、流式计算、机器学习和图算法组件

•运行模式多样：可运行于独立的集群模式中，可运行于Hadoop中，也可运行于Amazon EC2等云环境中，并且可以访问HDFS、Cassandra、HBase、Hive等多种数据源

Spark简介

Spark如今已吸引了国内外各大公司的注意，如腾讯、淘宝、百度、亚马逊等公司均不同程度地使用了Spark来构建大数据分析应用，并应用到实际的生产环境中

谷歌趋势：Spark与Hadoop对比

Scala简介

Scala是一门现代的多范式编程语言，运行于Java平台（JVM，Java 虚拟机），并兼容现有的Java程序

Scala的特性：

•Scala具备强大的并发性，支持函数式编程，可以更好地支持分布式系统

•Scala语法简洁，能提供优雅的API

Scala兼容Java，运行速度快，且能融合到Hadoop生态圈中

Scala是Spark的主要编程语言，但Spark还支持Java、Python、R作为编程语言

Scala的优势是提供了REPL（Read-Eval-Print Loop，交互式解释器），提高程序开发效率

Spark与Hadoop的对比

Hadoop存在如下一些缺点：

- 表达能力有限
- 磁盘IO开销大
- 延迟高
  - 任务之间的衔接涉及IO开销
  - 在前一个任务执行完成之前，其他任务就无法开始，难以胜任复杂、多阶段的计算任务

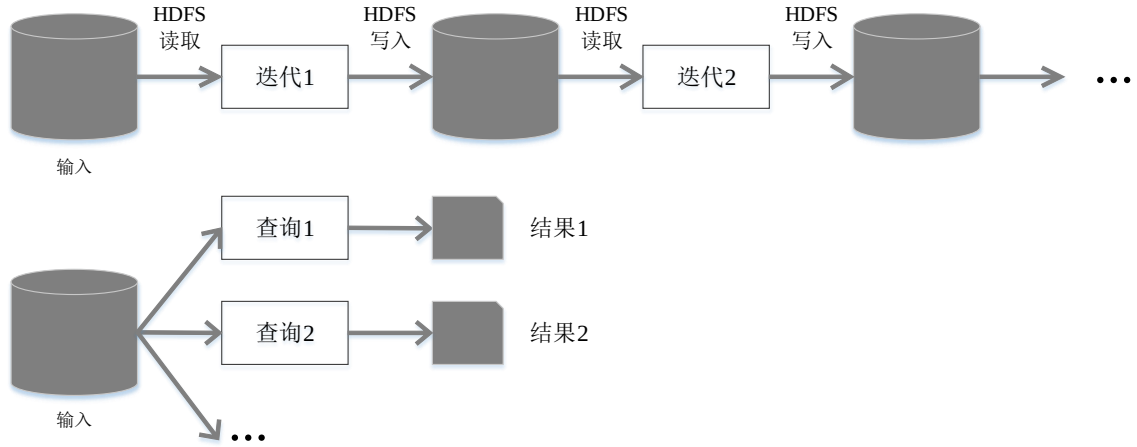
Spark与Hadoop的对比

Spark在借鉴Hadoop MapReduce优点的同时，很好地解决了MapReduce所面临的问题

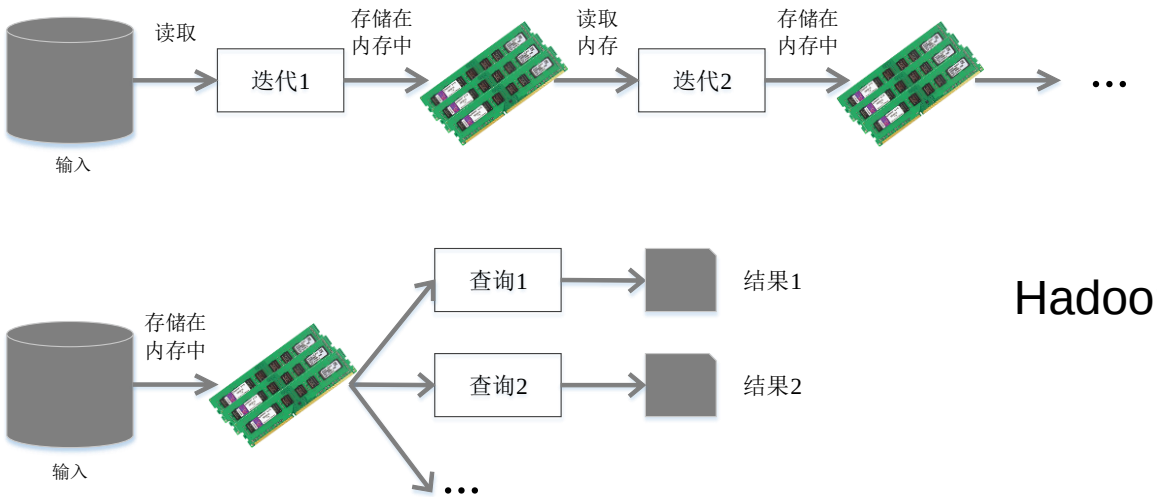
相比于Hadoop MapReduce，Spark主要具有如下优点：

- Spark的计算模式也属于MapReduce，但不局限于Map和Reduce操作，还提供了多种数据集操作类型，编程模型比Hadoop MapReduce更灵活
- Spark提供了内存计算，可将中间结果放到内存中，对于迭代运算效率更高
- Spark基于DAG的任务调度执行机制，要优于Hadoop MapReduce的迭代执行机制

Spark与Hadoop的对比



(a) Hadoop MapReduce执行流程

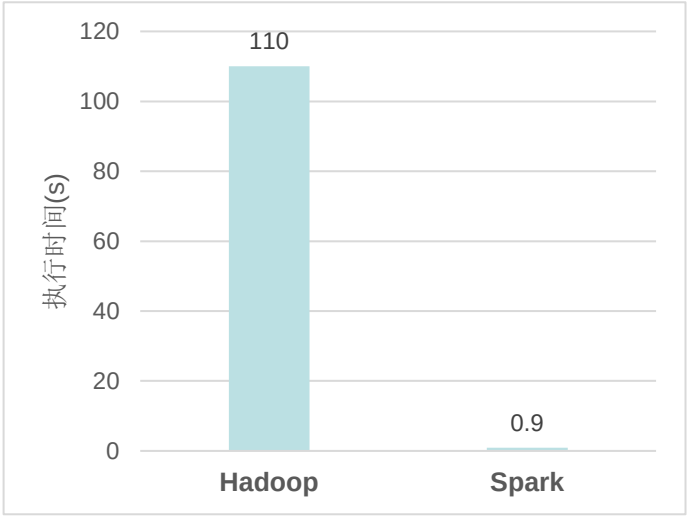


(b) Spark执行流程

Hadoop与Spark的执行流程对比

Spark与Hadoop的对比

- 使用Hadoop进行迭代计算非常耗资源
- Spark将数据载入内存后，之后的迭代计算都可以直接使用内存中的中间结果作运算，避免了从磁盘中频繁读取数据



Hadoop与Spark执行逻辑回归的时间对比

Spark生态系统

Spark生态系统

在实际应用中，大数据处理主要包括以下三个类型：

- 复杂的批量数据处理：通常时间跨度在数十分钟到数小时之间
- 基于历史数据的交互式查询：通常时间跨度在数十秒到数分钟之间
- 基于实时数据流的数据处理：通常时间跨度在数百毫秒到数秒之间

Spark生态系统

当同时存在以上三种场景时，就需要同时部署三种不同的软件

这样做难免会带来一些问题：

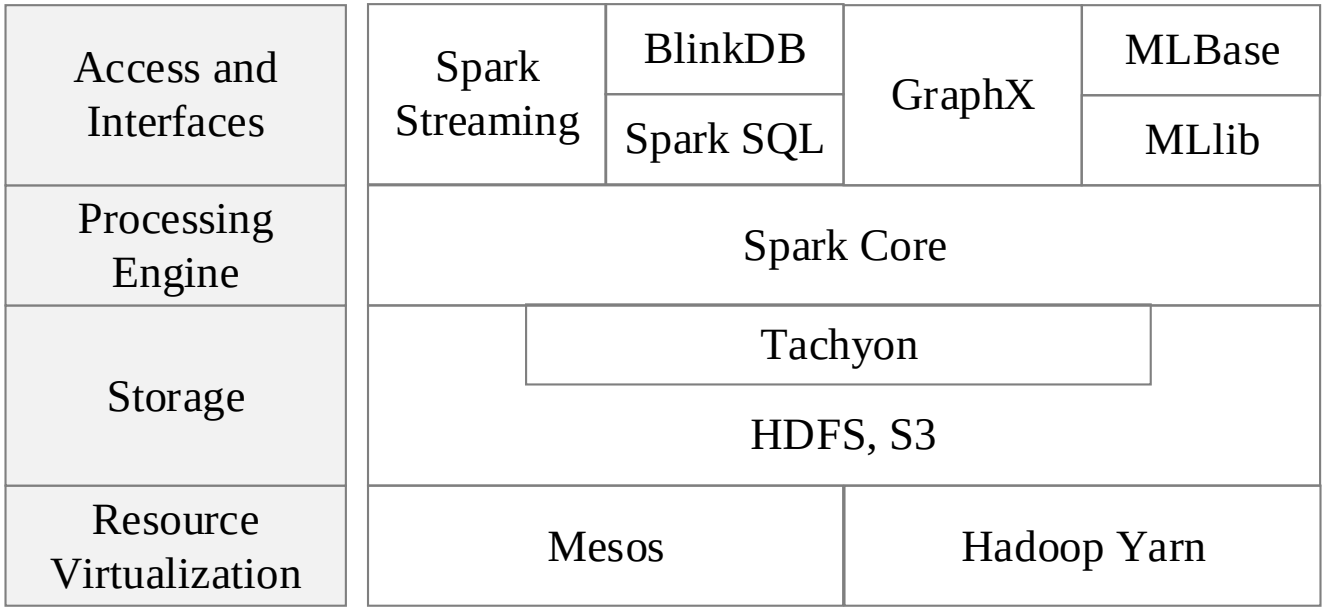
- 不同场景之间输入输出数据无法做到无缝共享，通常需要进行数据格式的转换
- 不同的软件需要不同的开发和维护团队，带来了较高的使用成本
- 比较难以对同一个集群中的各个系统进行统一的资源协调和分配

Spark生态系统

- Spark的设计遵循“一个软件栈满足不同应用场景”的理念，逐渐形成了一套完整的生态系统
- 既能够提供内存计算框架，也可以支持SQL即席查询、实时流式计算、机器学习和图计算等
- Spark可以部署在资源管理器YARN之上，提供一站式的大数据解决方案
- 因此，Spark所提供的生态系统足以应对上述三种场景，即同时支持批处理、交互式查询和流数据处理

Spark生态系统

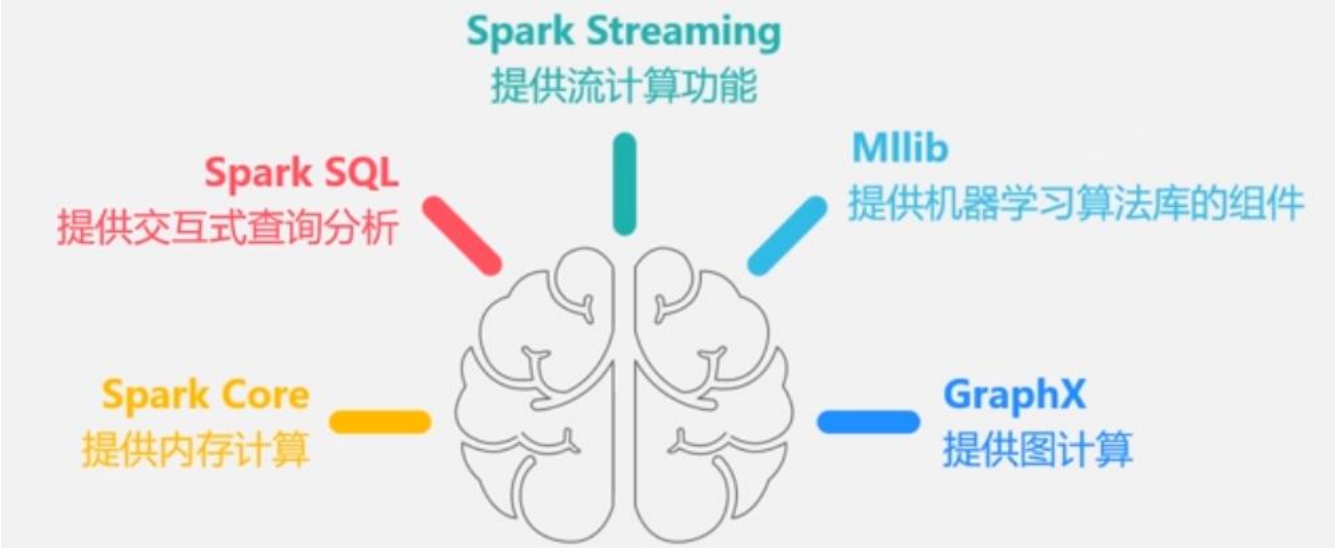
Spark生态系统已经成为伯克利数据分析软件栈BDAS（Berkeley Data Analytics Stack）的重要组成部分



BDAS架构

Spark生态系统

Spark的生态系统主要包含了Spark Core、Spark SQL、Spark Streaming、MLLib和GraphX 等组件



Spark生态系统

Spark生态系统组件的应用场景

应用场景	时间跨度	其他框架	Spark生态系统中的组件
复杂的批量数据处理	小时级	MapReduce、Hive	Spark
基于历史数据的交互式查询	分钟级、秒级	Impala、Dremel、Drill	Spark SQL
基于实时数据流的数据处理	毫秒、秒级	Storm、S4	Spark Streaming
基于历史数据的数据挖掘	-	Mahout	MLlib
图结构数据的处理	-	Pregel、Hama	GraphX

Spark运行架构

Spark运行架构

基本概念

架构设计

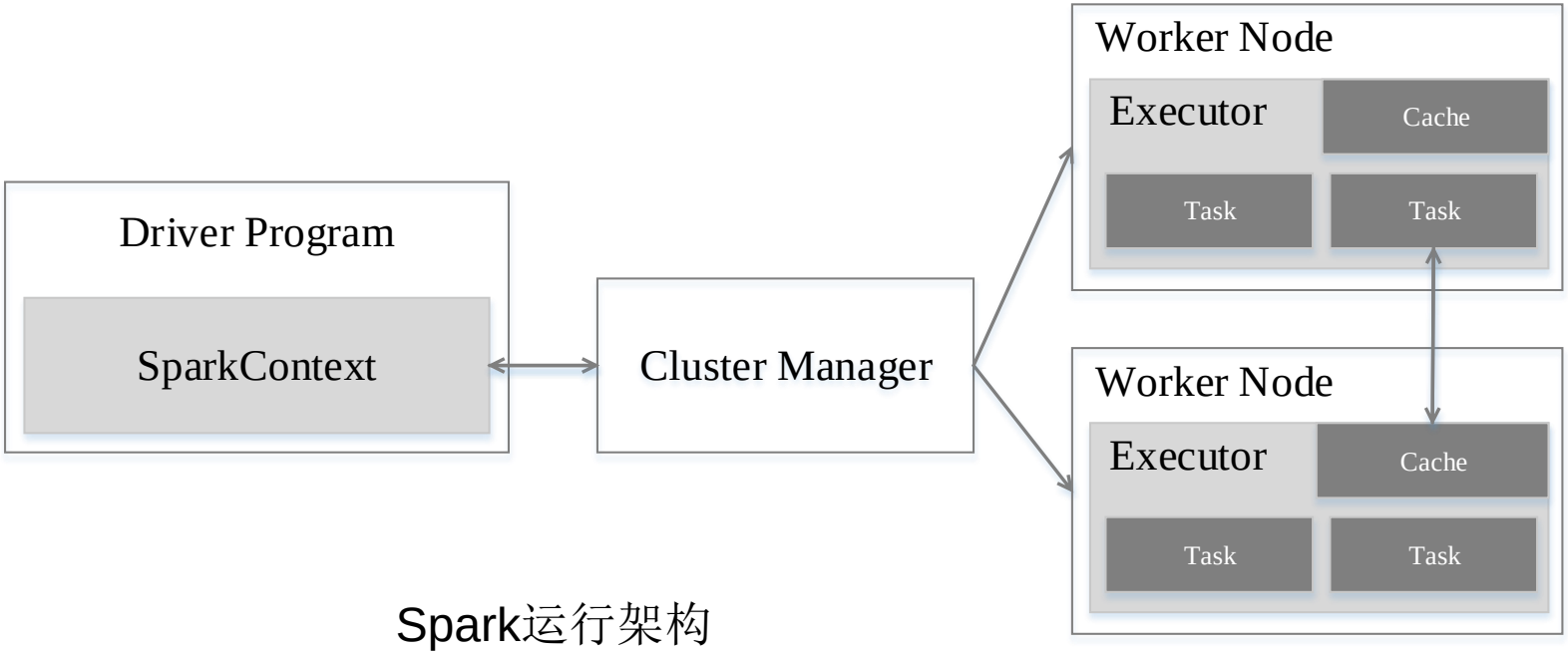
Spark运行基本流程

Spark运行原理

基本概念

- RDD：是Resillient Distributed Dataset（弹性分布式数据集）的简称，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型
- DAG：是Directed Acyclic Graph（有向无环图）的简称，反映RDD之间的依赖关系
- Executor：是运行在工作节点（WorkerNode）的一个进程，负责运行Task
- Application：用户编写的Spark应用程序
- Task：运行在Executor上的工作单元
- Job：一个Job包含多个RDD及作用于相应RDD上的各种操作
- Stage：是Job的基本调度单位，一个Job会分为多组Task，每组Task被称为Stage，或者也被称为TaskSet，代表了一组关联的、相互之间没有Shuffle依赖关系的任务组成的任务集

架构设计



Spark运行架构

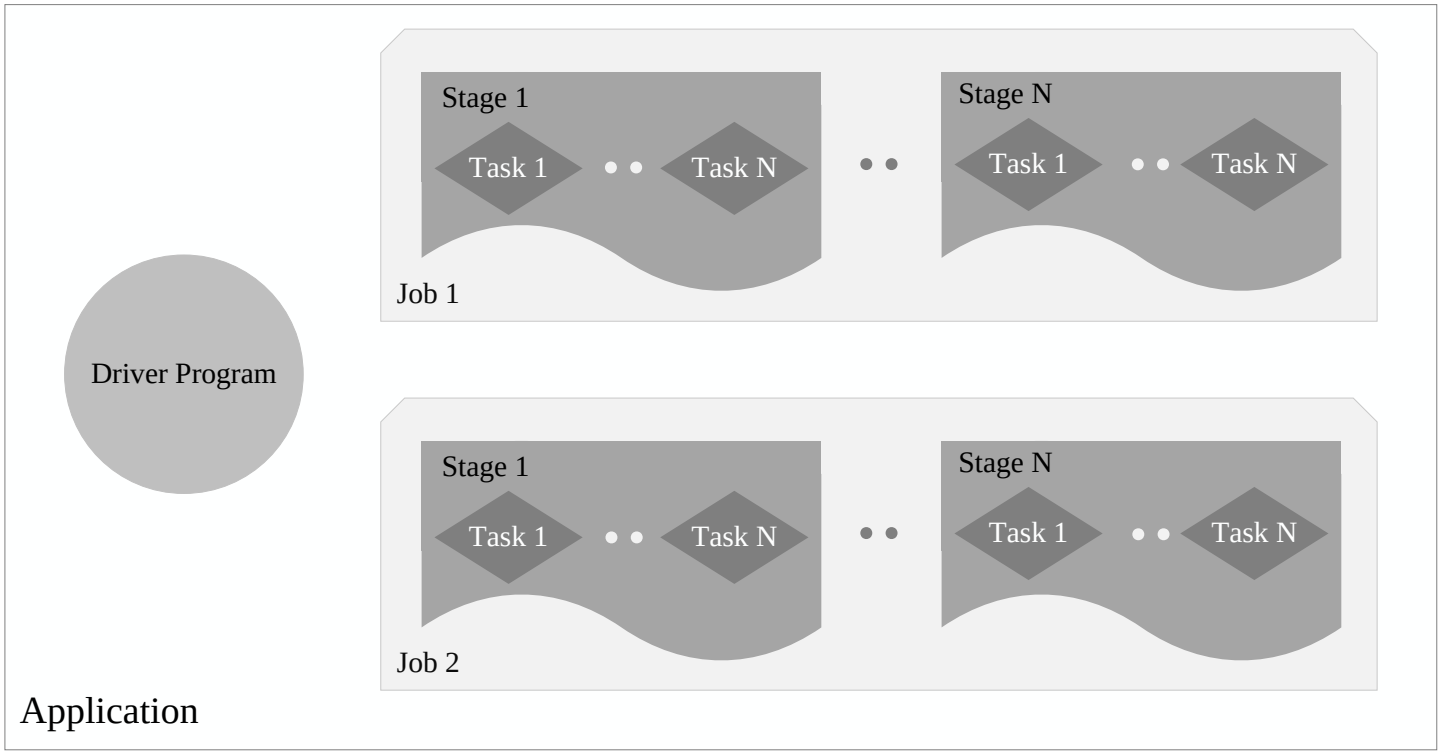
与Hadoop MapReduce计算框架相比，Spark所采用的Executor有两个优点：

- 一是利用多线程来执行具体的任务，减少任务的启动开销
- 二是Executor中有一个BlockManager存储模块，会将内存和磁盘共同作为存储设备，有效减少IO开销

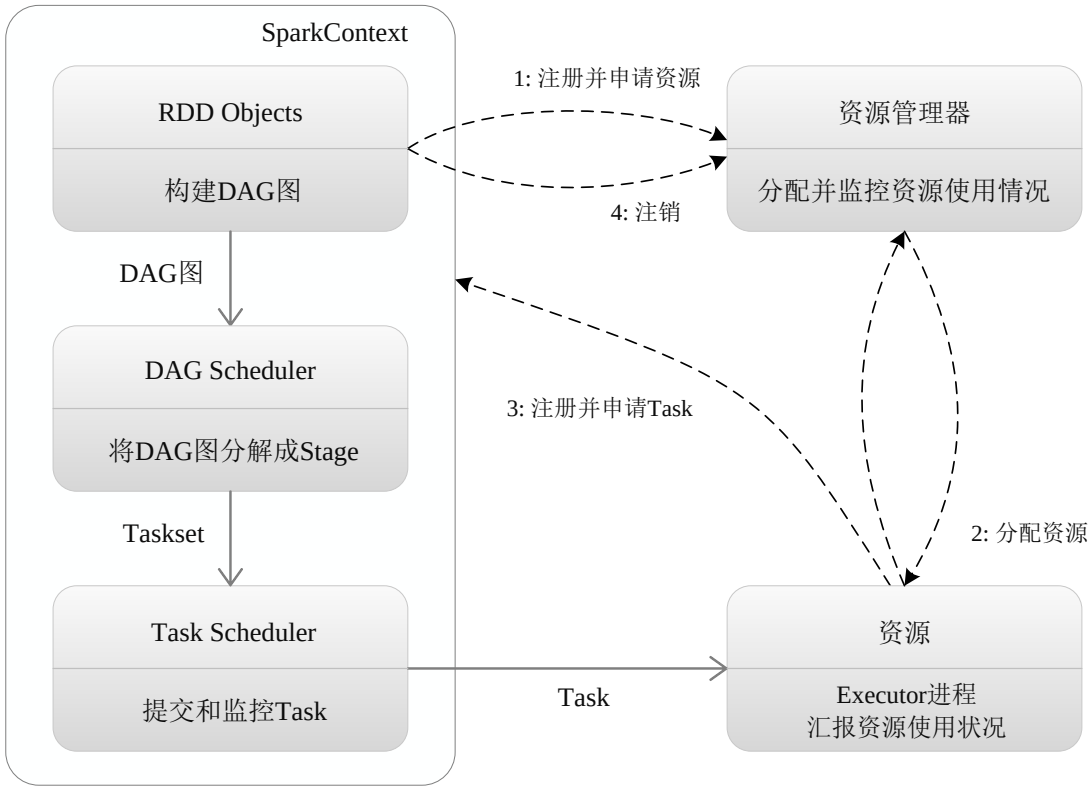
架构设计

- 一个Application由一个Driver和若干个Job构成，一个Job由多个Stage构成，一个Stage由多个没有Shuffle关系的Task组成
- 当执行一个Application时，Driver会向集群管理器申请资源，启动Executor，并向Executor发送应用程序代码和文件，然后在Executor上执行Task，运行结束后，执行结果会返回给Driver，或者写到HDFS或者其他数据库中

Spark中各种概念之间的相互关系



Spark运行基本流程



Spark运行基本流程图

（1）首先为应用构建起基本的运行环境，即由Driver创建一个SparkContext，进行资源的申请、任务的分配和监控

（2）资源管理器为Executor分配资源，并启动Executor进程

（3）SparkContext根据RDD的依赖关系构建DAG图，DAG图提交给DAGScheduler解析成Stage，然后把一个个TaskSet提交给底层调度器TaskScheduler处理；Executor向SparkContext申请Task，Task Scheduler将Task发放给Executor运行，并提供应用程序代码

（4）Task在Executor上运行，把执行结果反馈给TaskScheduler，然后反馈给DAGScheduler，运行完毕后写入数据并释放所有资源

Spark运行基本流程

总体而言，Spark运行架构具有以下特点：

（1）每个Application都有自己专属的Executor进程，并且该进程在Application运行期间一直驻留。Executor进程以多线程的方式运行Task

（2）Spark运行过程与资源管理器无关，只要能够获取Executor进程并保持通信即可

（3）Task采用了数据本地性和推测执行等优化机制

RDD运行原理

1.设计背景
2.RDD概念
3.RDD特性
4.RDD之间的依赖关系
5.Stage的划分
6.RDD运行过程

RDD运行原理

1.设计背景

- 许多迭代式算法（比如机器学习、图算法等）和交互式数据挖掘工具，共同之处是，不同计算阶段之间会重用中间结果
- 目前的MapReduce框架都是把中间结果写入到HDFS中，带来了大量的数据复制、磁盘IO和序列化开销
- RDD就是为了满足这种需求而出现的，它提供了一个抽象的数据架构，我们不必担心底层数据的分布式特性，只需将具体的应用逻辑表达为一系列转换处理，不同RDD之间的转换操作形成依赖关系，可以实现管道化，避免中间数据存储

RDD运行原理

2.RDD概念

- 一个RDD就是一个分布式对象集合，本质上是一个只读的分区记录集合，每个RDD可分成多个分区，每个分区就是一个数据集片段，并且一个RDD的不同分区可以被保存到集群中不同的节点上，从而可以在集群中的不同节点上进行并行计算
- RDD提供了一种高度受限的共享内存模型，即RDD是只读的记录分区的集合，不能直接修改，只能基于稳定的物理存储中的数据集创建RDD，或者通过在其他RDD上执行确定的转换操作（如map、join和group by）而创建得到新的RDD

RDD运行原理

- RDD提供了一组丰富的操作以支持常见的数据运算，分为“动作”（Action）和“转换”（Transformation）两种类型
- RDD提供的转换接口都非常简单，都是类似map、filter、groupBy、join等粗粒度的数据转换操作，而不是针对某个数据项的细粒度修改（不适合网页爬虫）
- 表面上RDD的功能很受限、不够强大，实际上RDD已经被实践证明可以高效地表达许多框架的编程模型（比如MapReduce、SQL、Pregel）
- Spark用Scala语言实现了RDD的API，程序员可以通过调用API实现对RDD的各种操作

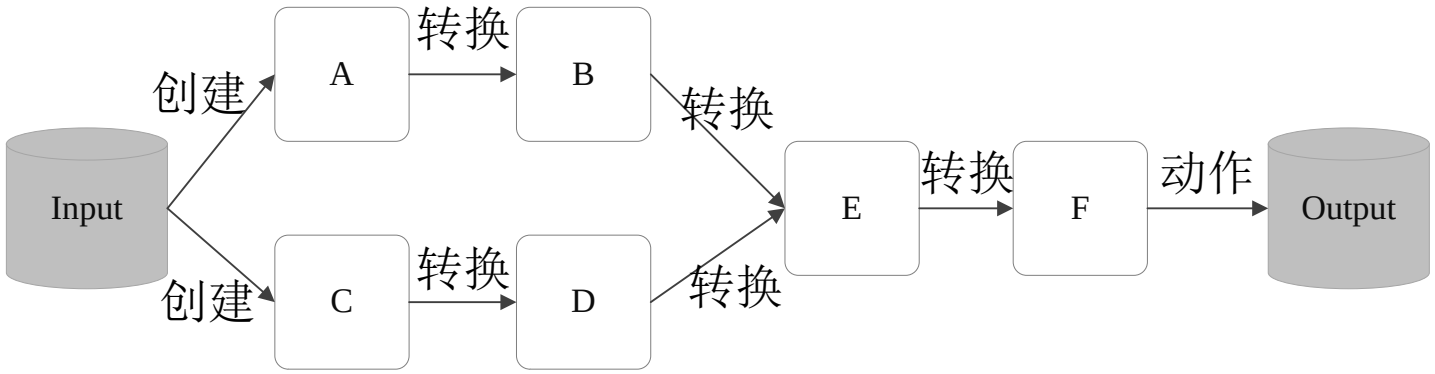
RDD运行原理

RDD典型的执行过程如下：

- RDD读入外部数据源进行创建
- RDD经过一系列的转换（Transformation）操作，每一次都会产生不同的RDD，供给下一个转换操作使用
- 最后一个RDD经过“动作”操作进行转换，并输出到外部数据源

这一系列处理称为一个Lineage（血缘关系），即DAG拓扑排序的结果

优点：惰性调用、管道化、避免同步等待、不需要保存中间结果、每次操作变得简单



RDD执行过程的一个实例

RDD运行原理

3.RDD特性

Spark采用RDD以后能够实现高效计算的原因主要在于：

（1）高效的容错性

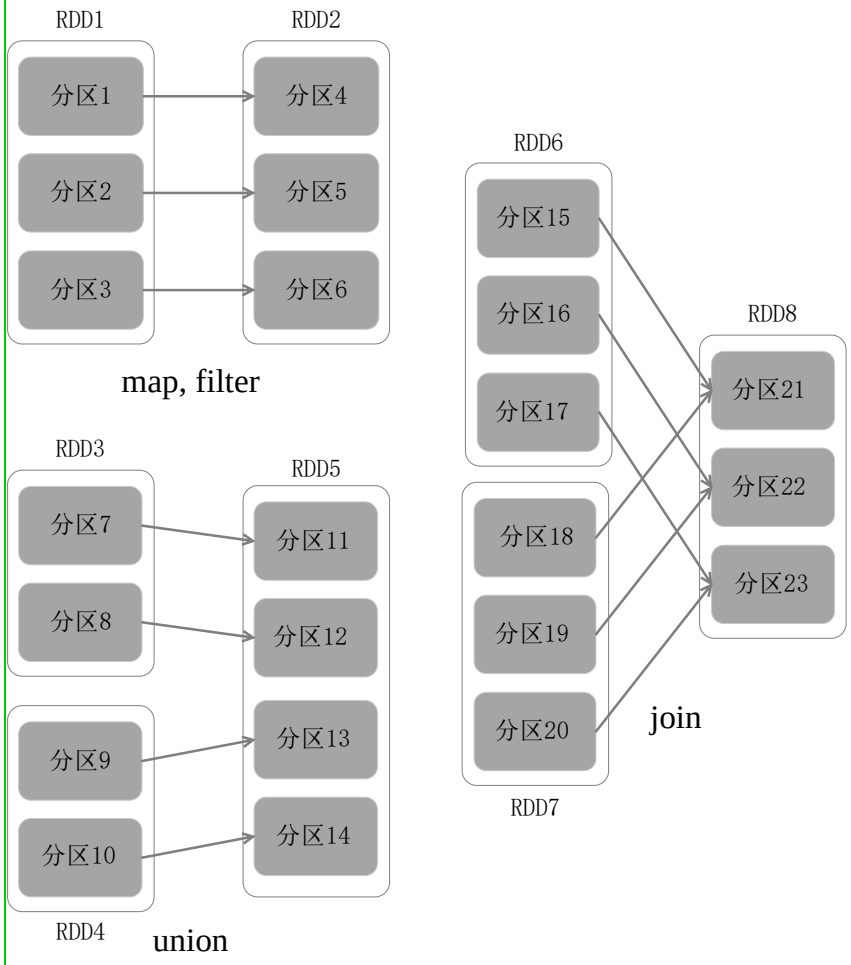
- 现有容错机制：数据复制或者记录日志
- RDD：血缘关系、重新计算丢失分区、无需回滚系统、重算过程在不同节点之间并行、只记录粗粒度的操作

（2）中间结果持久化到内存，数据在内存中的多个RDD操作之间进行传递，避免了不必要的读写磁盘开销

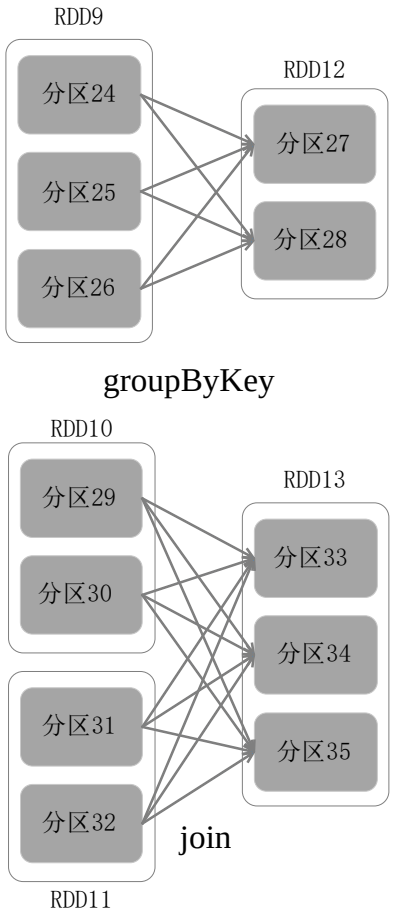
（3）存放的数据可以是Java对象，避免了不必要的对象序列化和反序列化

RDD运行原理

4. RDD之间的依赖关系



(a)窄依赖



(b)宽依赖

- 窄依赖表现为一个父RDD的分区对应于一个子RDD的分区或多个父RDD的分区对应于一个子RDD的分区
- 宽依赖则表现为存在一个父RDD的一个分区对应一个子RDD的多个分区

窄依赖与宽依赖的区别

RDD运行原理

5.Stage的划分

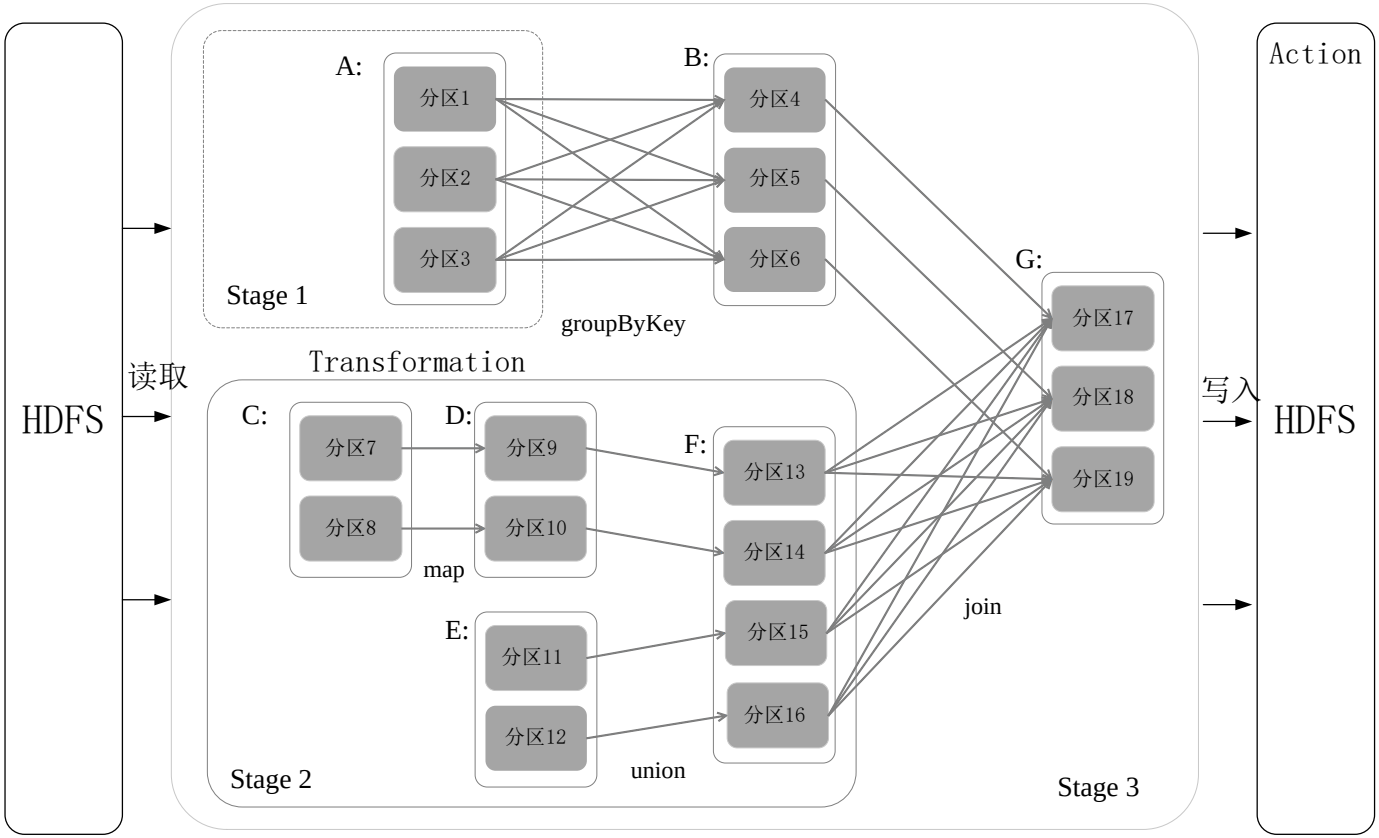
Spark通过分析各个RDD的依赖关系生成了DAG，再通过分析各个RDD中的分区之间的依赖关系来决定如何划分Stage，具体划分方法是：

- 在DAG中进行反向解析，遇到宽依赖就断开
- 遇到窄依赖就把当前的RDD加入到Stage中
- 将窄依赖尽量划分在同一个Stage中，可以实现流水线计算

RDD运行原理

5.Stage的划分

被分成三个Stage，在Stage2中，从map到union都是窄依赖，这两步操作可以形成一个流水线操作



根据RDD分区的依赖关系划分Stage

流水线操作实例

分区7通过map操作生成的分区9，可以不用等待分区8到分区10这个map操作的计算结束，而是继续进行union操作，得到分区13，这样流水线执行大大提高了计算的效率

RDD运行原理

5.Stage的划分

Stage的类型包括两种：ShuffleMapStage和ResultStage，具体如下：

（1）ShuffleMapStage：不是最终的Stage，在它之后还有其他Stage，所以，它的输出一定需要经过Shuffle过程，并作为后续Stage的输入；这种Stage是以Shuffle为输出边界，其输入边界可以是从外部获取数据，也可以是另一个ShuffleMapStage的输出，其输出可以是另一个Stage的开始；在一个Job里可能有该类型的Stage，也可能没有该类型Stage；

（2）ResultStage：最终的Stage，没有输出，而是直接产生结果或存储。这种Stage是直接输出结果，其输入边界可以是从外部获取数据，也可以是另一个ShuffleMapStage的输出。在一个Job里必定有该类型Stage。

因此，一个Job含有一个或多个Stage，其中至少含有一个ResultStage。

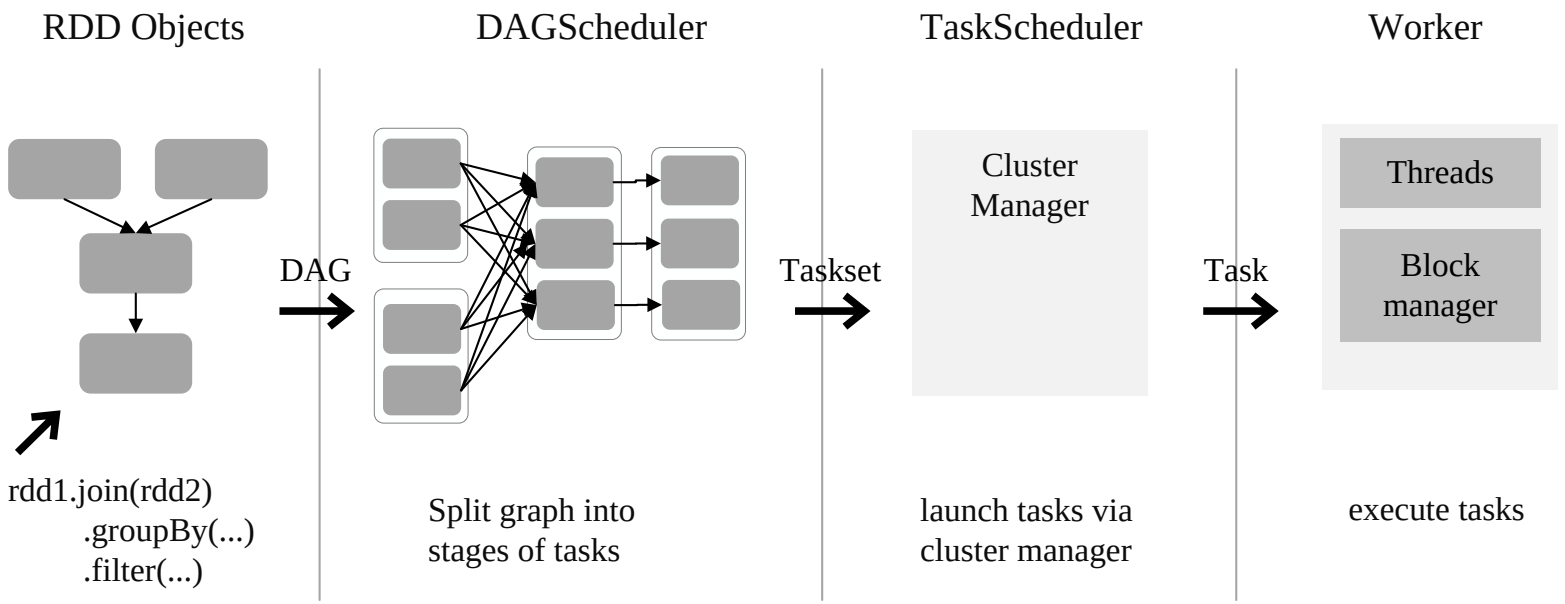
RDD运行原理

通过上述对RDD概念、依赖关系和Stage划分的介绍，结合之前介绍的Spark运行基本流程，再总结一下RDD在Spark架构中的运行过程：

（1）创建RDD对象；

（2）SparkContext负责计算RDD之间的依赖关系，构建DAG；

（3）DAGScheduler负责把DAG图分解成多个Stage，每个Stage中包含了多个Task，每个Task会被TaskScheduler分发给各个WorkerNode上的Executor去执行。



RDD在Spark中的运行过程

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Spark SQL

Spark SQL

从Shark说起

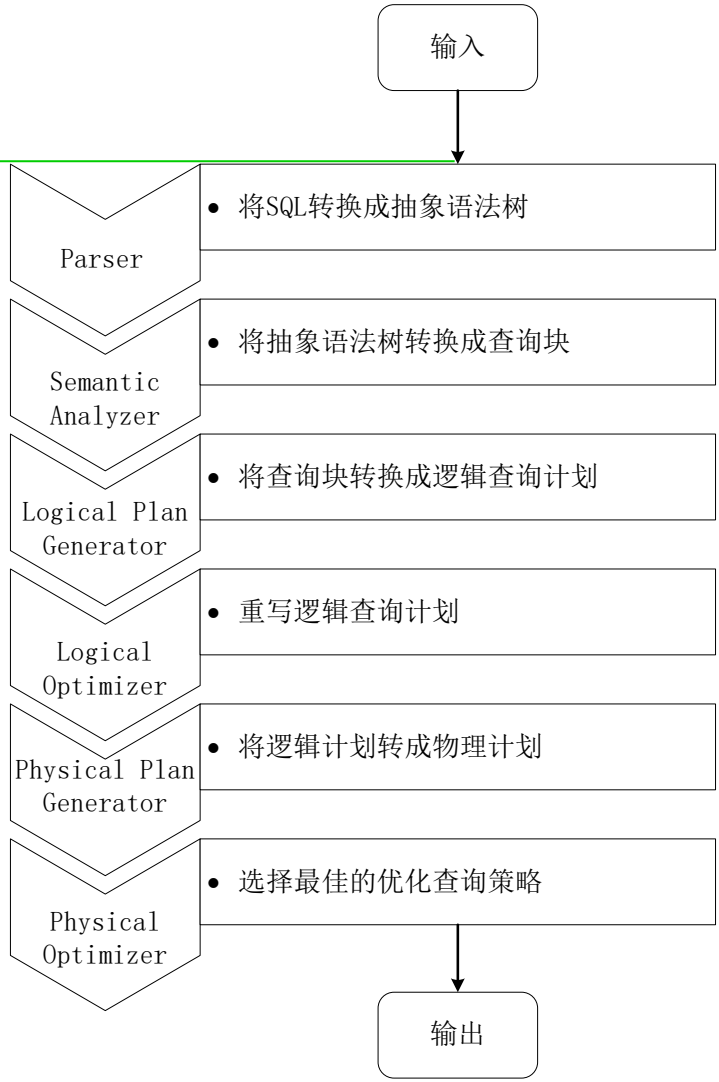
Spark SQL设计

从Shark说起

Shark即Hive on Spark，为了实现与Hive兼容，Shark在HiveQL方面重用了Hive中HiveQL的解析、逻辑执行计划翻译、执行计划优化等逻辑，可以近似认为仅将物理执行计划从MapReduce作业替换成了Spark作业，通过Hive的HiveQL解析，把HiveQL翻译成Spark上的RDD操作。

Shark的设计导致了两个问题：

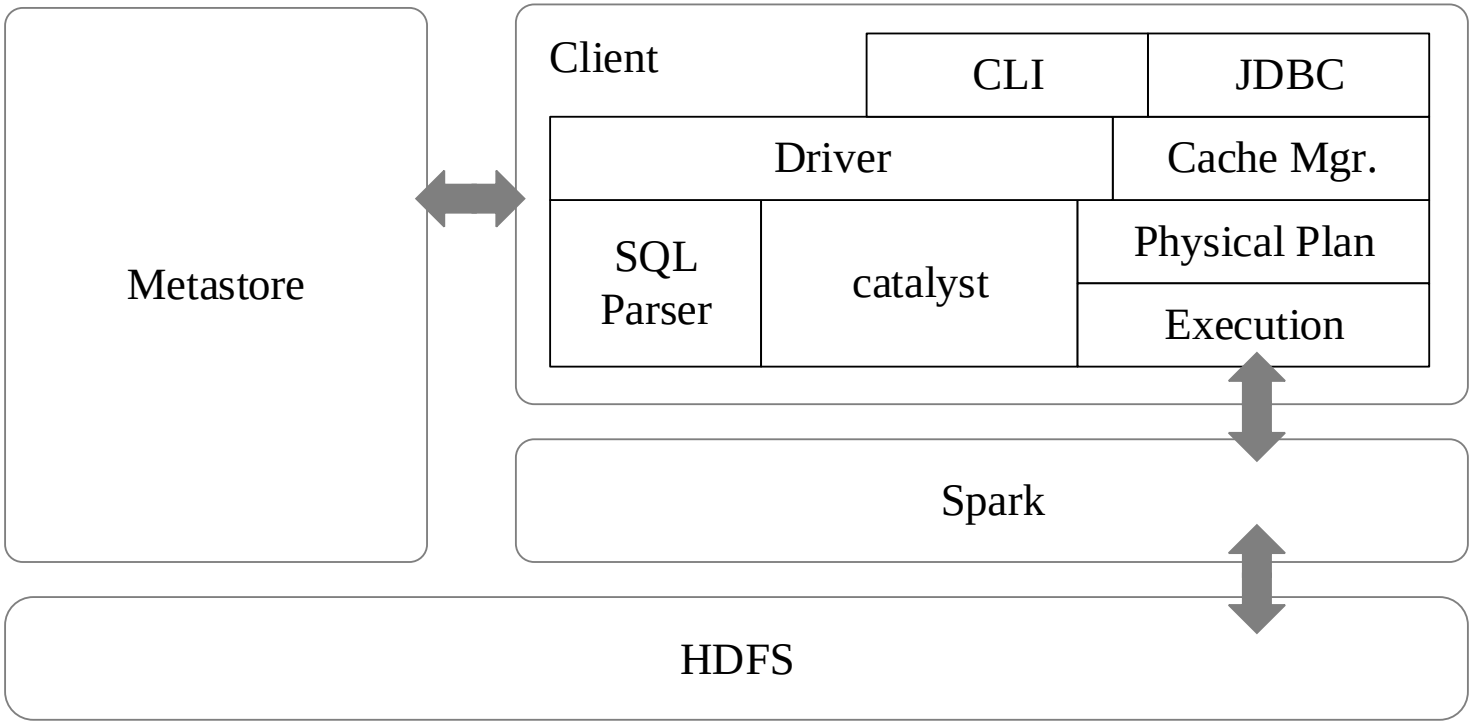
- 一是执行计划优化完全依赖于Hive，不方便添加新的优化策略；
- 二是因为Spark是线程级并行，而MapReduce是进程级并行，因此，Spark在兼容Hive的实现上存在线程安全问题，导致Shark不得不使用另外一套独立维护的打了补丁的Hive源码分支



Hive中SQL查询的MapReduce作业转化过程

Spark SQL设计

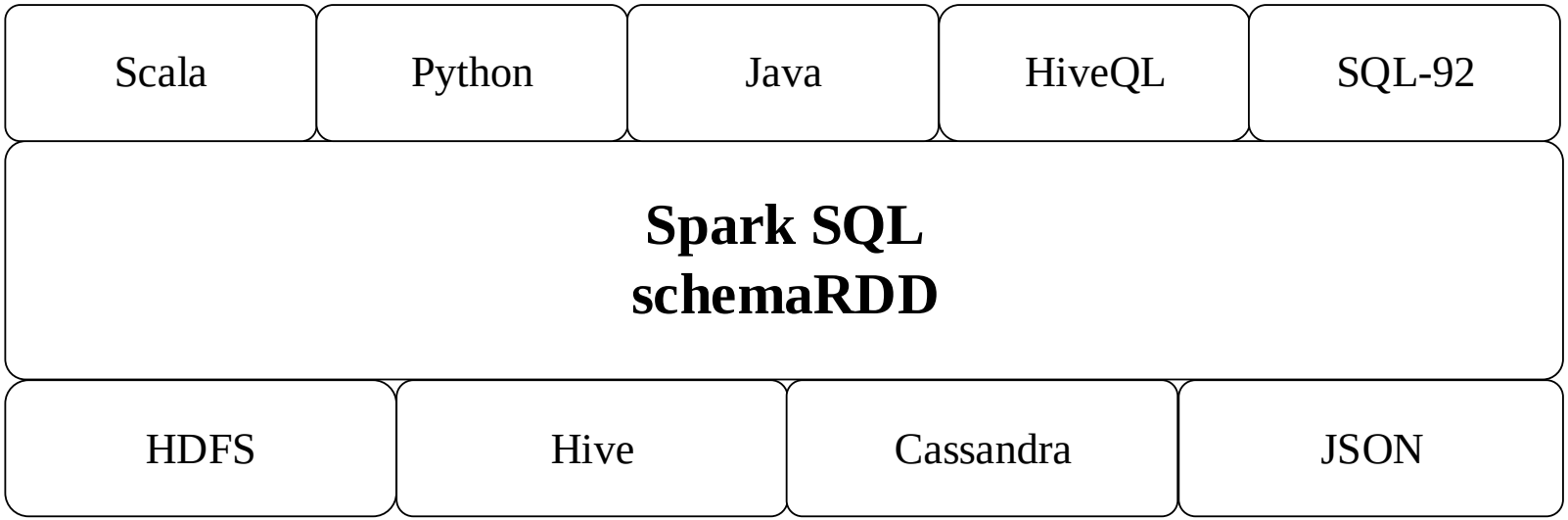
Spark SQL在Hive兼容层面仅依赖HiveQL解析、Hive元数据，也就是说，从HQL被解析成抽象语法树（AST）起，就全部由Spark SQL接管了。Spark SQL执行计划生成和优化都由Catalyst（函数式关系查询优化框架）负责



Spark SQL架构

Spark SQL设计

- Spark SQL增加了SchemaRDD（即带有Schema信息的RDD），使用户可以在Spark SQL中执行SQL语句，数据既可以来自RDD，也可以是Hive、HDFS、Cassandra等外部数据源，还可以是JSON格式的数据
- Spark SQL目前支持Scala、Java、Python三种语言，支持SQL-92规范



Spark SQL支持的数据格式和编程语言

Spark的部署和应用方式

Spark的部署和应用方式

Spark三种部署方式

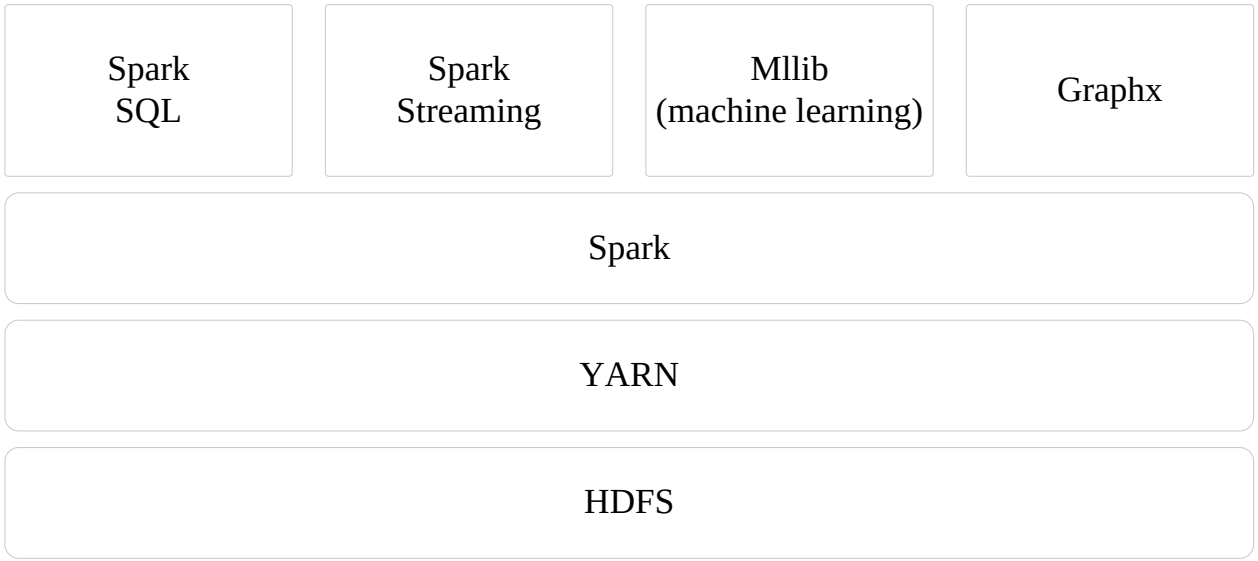
从Hadoop+Storm架构转向Spark架构

Hadoop和Spark的统一部署

Spark三种部署方式

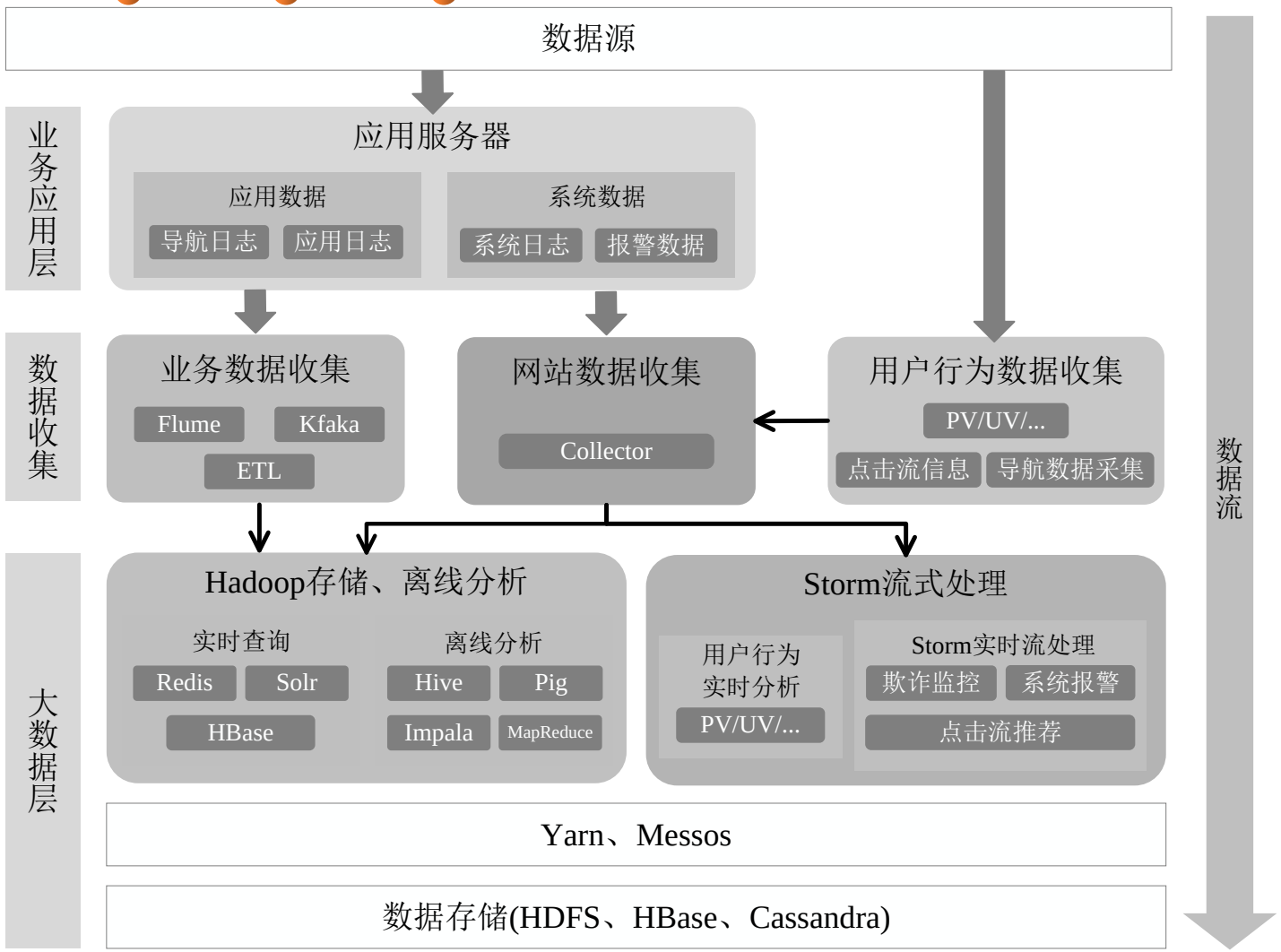
Spark支持三种不同类型的部署方式，包括：

- Standalone（类似于MapReduce1.0，slot为资源分配单位）
- Spark on Mesos（和Spark有血缘关系，更好支持Mesos）
- Spark on YARN



Spark on Yarn架构

从Hadoop+Storm架构转向Spark架构



采用Hadoop+Storm部署方式的一个案例

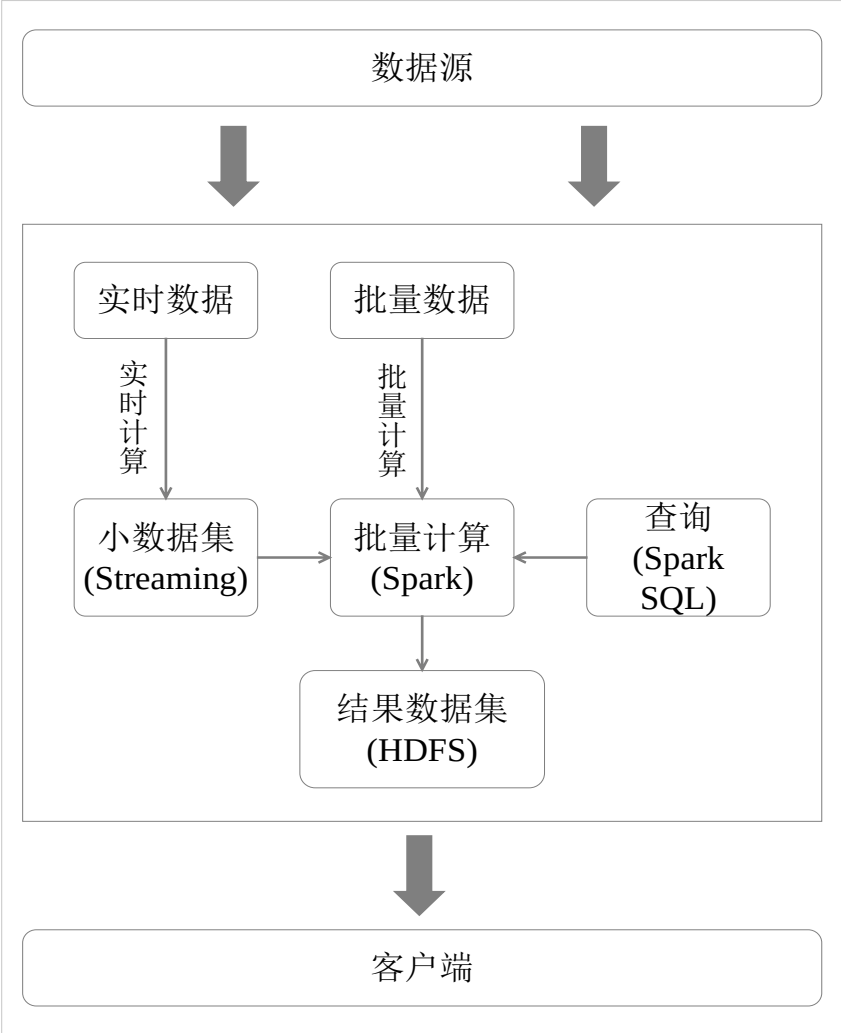
这种架构部署较为繁琐

从Hadoop+Storm架构转向Spark架构

用Spark架构具有如下优点：

- 实现一键式安装和配置、线程级别的任务监控和告警
- 降低硬件集群、软件维护、任务监控和应用开发的难度
- 便于做成统一的硬件、计算平台资源池

需要说明的是，Spark Streaming无法实现毫秒级的流计算，因此，对于需要毫秒级实时响应的企业应用而言，仍然需要采用流计算框架（如Storm）



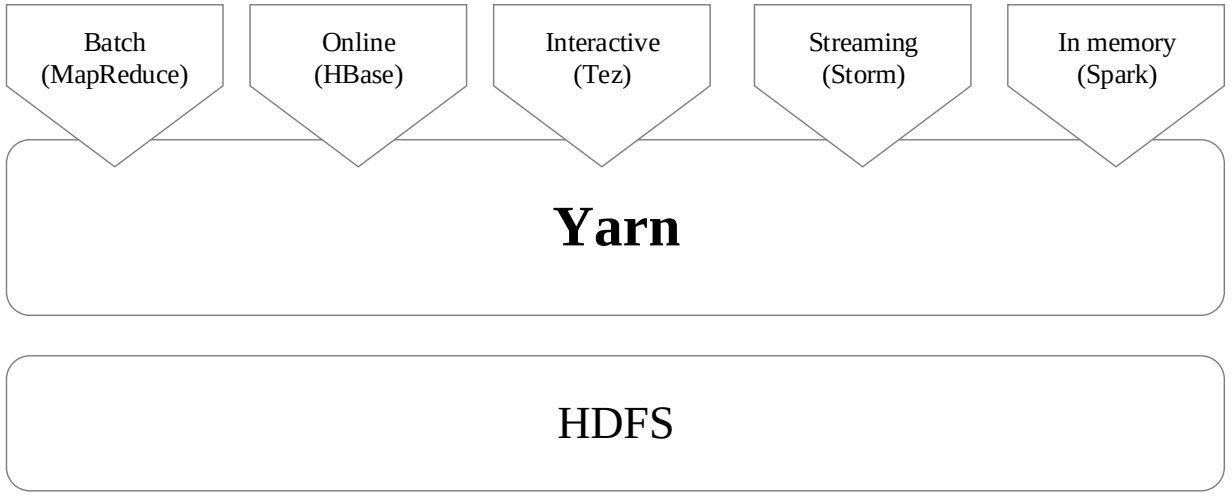
用Spark架构满足批处理和流处理需求

Hadoop和Spark的统一部署

- 由于Hadoop生态系统中的一些组件所实现的功能，目前还是无法由Spark取代的，比如，Storm
- 现有的Hadoop组件开发的应用，完全转移到Spark上需要一定的成本

不同的计算框架统一运行在YARN中，可以带来如下好处：

- 计算资源按需伸缩
- 不用负载应用混搭，集群利用率高
- 共享底层存储，避免数据跨集群迁移



Hadoop和Spark的统一部署

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

致谢

部分图表、文字来自互联网，在此表示感谢！如有版权要求请联系：lics@hit.edu.cn，谢谢