Spark系统简介

摘要

- 为什么会有Spark?
- Spark的生态圈
- Spark的基本构架和组件
- Spark的程序执行过程
- Spark的技术特点

摘要

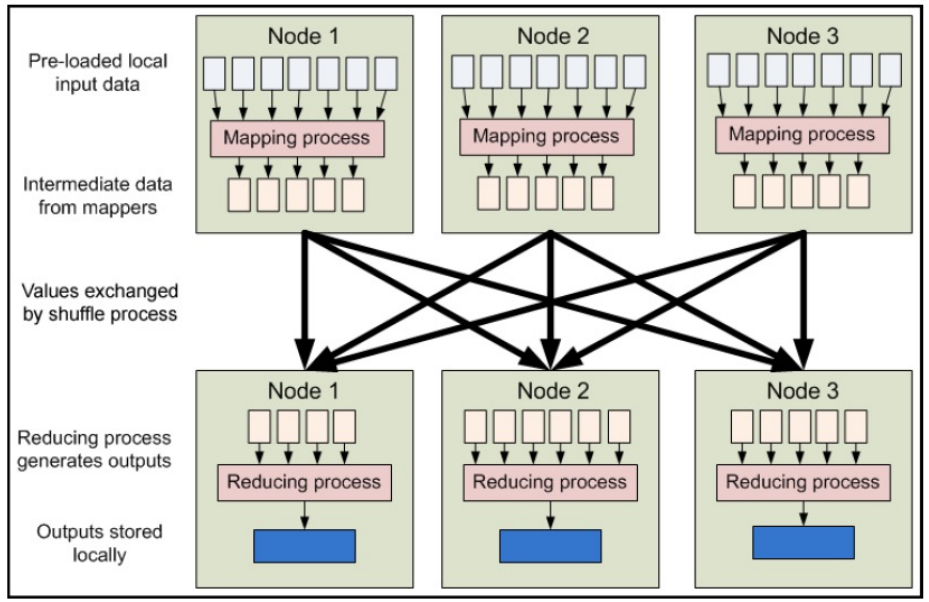
- 为什么会有Spark?
- Spark的生态圈
- Spark的基本构架和组件
- Spark的程序执行过程
- Spark的技术特点

为什么会有Spark?

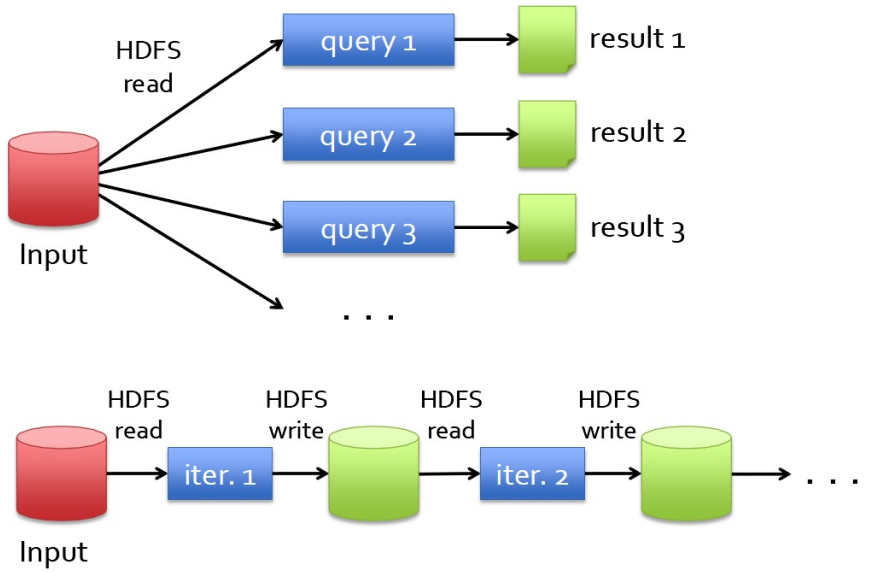
- MapReduce计算模式的缺陷
  - Originally designed for high-throughput batch data processing, not good at low latency
  - Needs to store data into HDFS between jobs, inefficient for data sharing in iterative computing
  - Not designed for making good use of memory, hard to achieve high performance
  - MapReduce is not expressive for complex computing problems, such as graph computing, iterative computing

为什么会有Spark?

- MapReduce计算模式的缺陷
  - 2阶段固定模式，磁盘计算大量I/O性能低下



One input, two-stage data flow.



为什么会有Spark?

- 2013年大数据计算模式的新变化
  - 由于Hadoop计算框架对很多非批处理大数据问题的局限性，除了原有的基于Hadoop HBase的数据存储管理模式和MapReduce计算模式外，人们开始关注大数据处理所需要的其他各种计算模式和系统。
  - 后Hadoop时代新的大数据计算模式和系统出现，其中尤其以内存计算为核心、集诸多计算模式之大成的Spark生态系统的出现为典型代表。

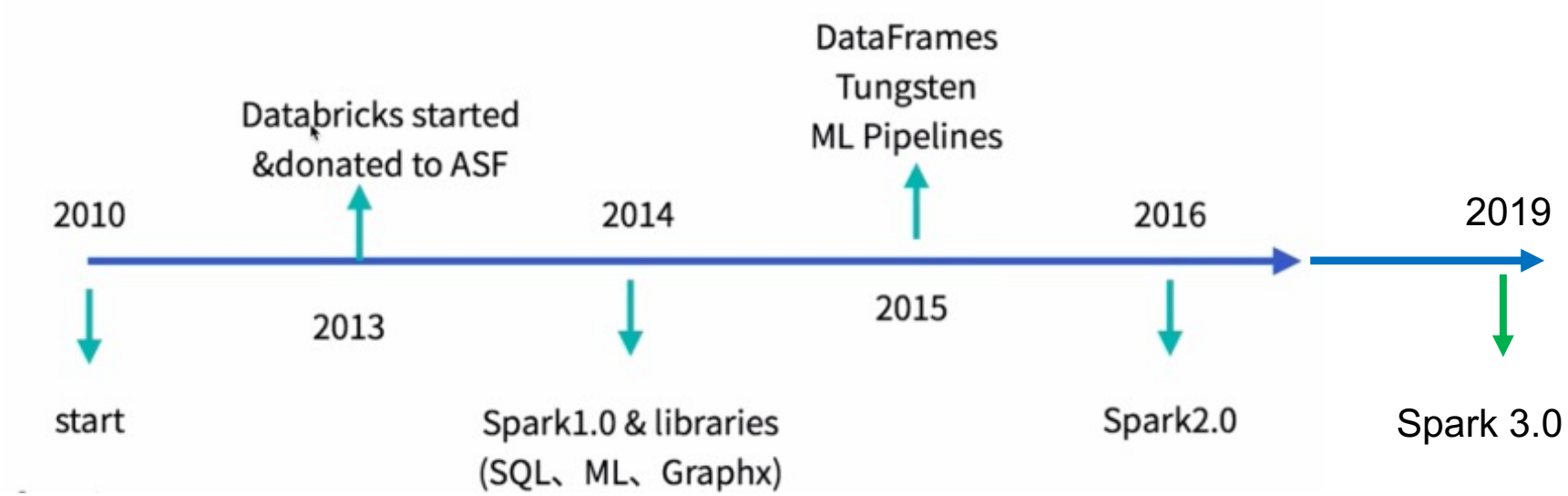
✓ 大数据查询分析计算
✓ 批处理计算

✓ 流式计算
✓ 迭代计算
✓ 图计算
✓ 内存计算

为什么会有Spark?

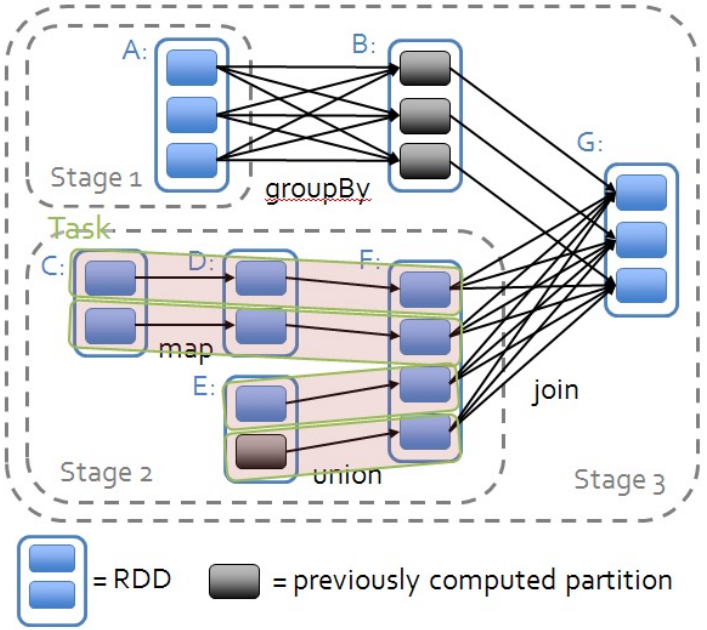
- Spark是加州大学伯克利分校AMP实验室2009年开发的通用内存并行计算框架。2010年开放源代码。
- Spark于2013年6月进入Apache成为孵化项目，8个月后成为Apache顶级项目。
- 围绕Spark推出了Spark SQL、Spark Streaming、MLlib和GraphX等组件，逐渐形成大数据处理一站式解决平台。
- 最新版本
  - 2023年9月13日，Spark 3.5.0

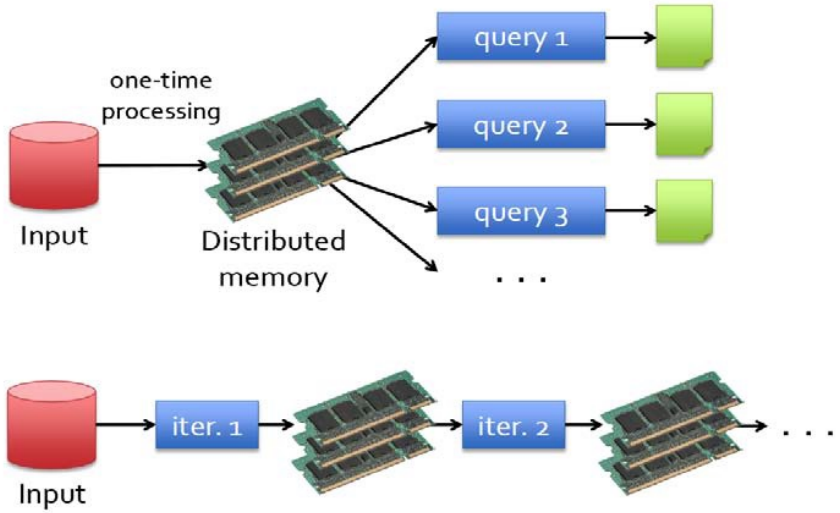
Spark发展历史


DataFrames
Tungsten
ML Pipelines
Databricks started
&donated to ASF
2010
2014
2016
2019
2013
2015
start
Spark1.0 & libraries
(SQL、ML、Graphx)
Spark2.0
Spark 3.0

为什么会有Spark?

- Spark基于内存计算思想提高计算性能
  - Spark提出了一种基于内存的弹性分布式数据集(RDD)，通过对RDD的一系列操作完成计算任务，可以大大提高性能
  - 同时一组RDD形成可执行的有向无环图DAG，构成灵活的计算流图
  - 覆盖多种计算模式





为什么会有Spark?

- 弹性分布式数据集Resilient Distributed Datasets (RDDs)
  - Spark的主要抽象是提供一个弹性分布式数据集(RDD)，RDD 是指能横跨集群所有节点进行并行计算的分区元素集合。RDD可以从Hadoop的文件系统中的一个文件中创建而来(或其他 Hadoop支持的文件系统)，或者从一个已有的Scala集合转换得到。
  - Spark使用RDD以及对应的Transform/Action等操作算子执行分布式计算。

为什么会有Spark?

- 弹性分布式数据集Resilient Distributed Datasets (RDDs)
  - 基于RDD之间的依赖关系组成lineage，通过重计算以及checkpoint等机制来保证整个分布式计算的容错性。
  - 只读、可分区，这个数据集的全部或部分可以缓存在内存中，在多次计算间重用，弹性是指内存不够时可以与磁盘进行交换。

为什么会有Spark?

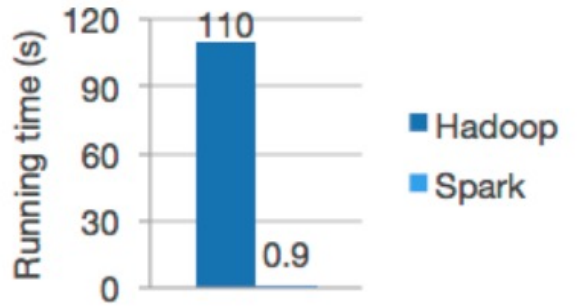
Lightning-fast unified analytics engine

Apache Spark™ is a unified analytics engine for large-scale data processing.

Speed

Run workloads 100x faster.

Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.



Logistic regression in Hadoop and Spark

为什么会有Spark?

Ease of Use

Write applications quickly in Java, Scala, Python, R, and SQL.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python, R, and SQL shells.

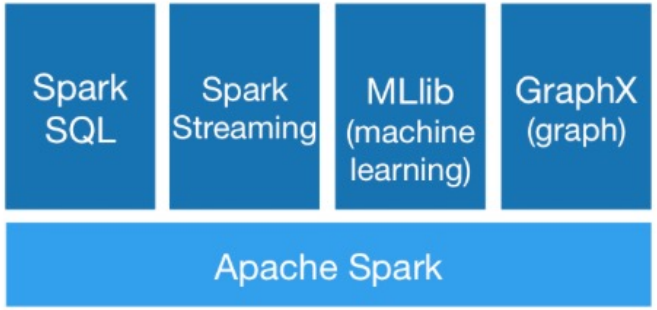
```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Spark's Python DataFrame API
Read JSON files with automatic schema inference

Generality

Combine SQL, streaming, and complex analytics.

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.



为什么会有Spark?

Runs Everywhere

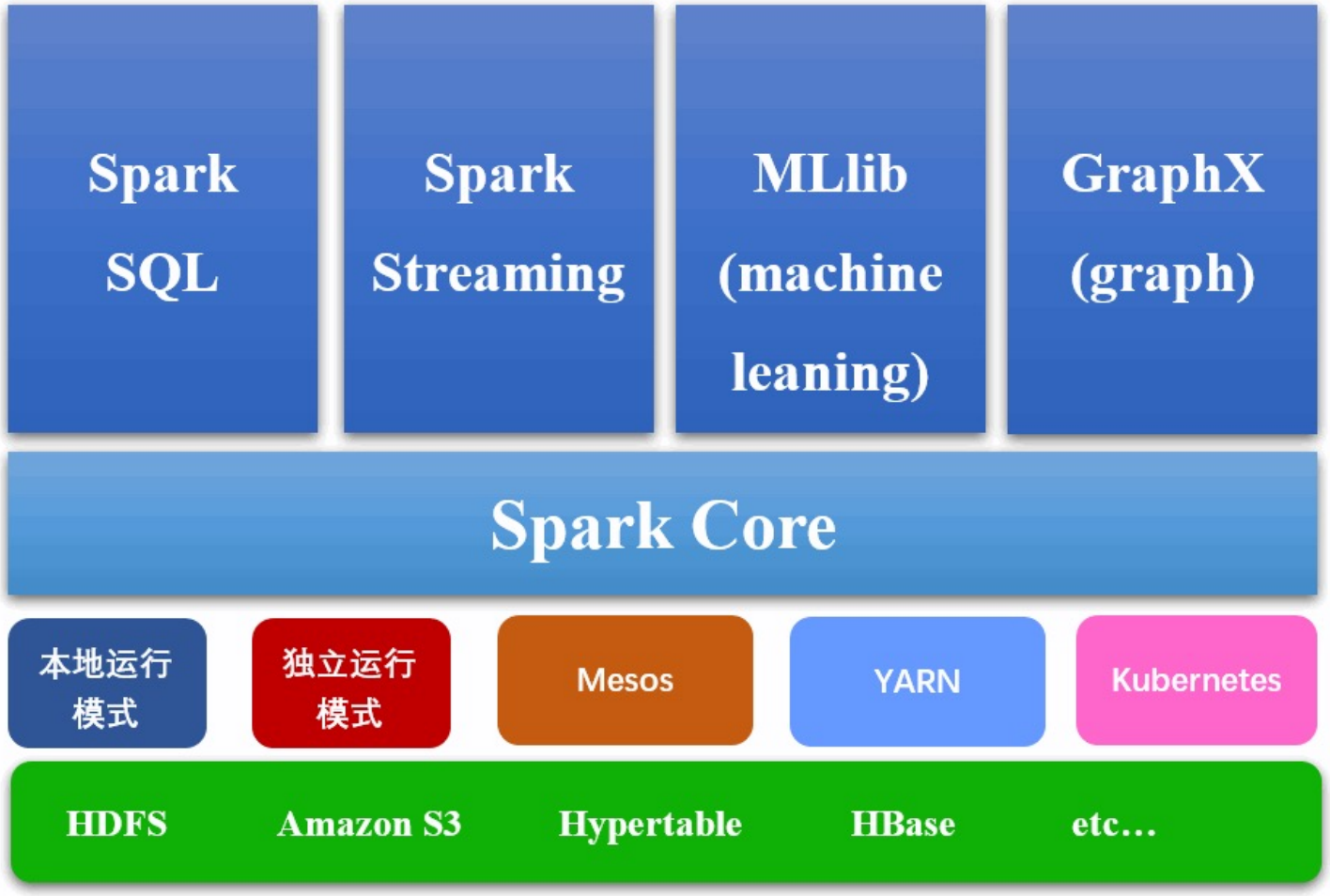
Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.

You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, on Mesos, or on Kubernetes. Access data in HDFS, Alluxio, Apache Cassandra, Apache HBase, Apache Hive, and hundreds of other data sources.

摘要

- 为什么会有Spark?
- Spark的生态圈
- Spark的基本构架和组件
- Spark的程序执行过程
- Spark的技术特点

Spark生态圈


Spark SQL
Spark Streaming
MLlib (machine leaning)
GraphX (graph)
Spark Core
本地运行模式
独立运行模式
Mesos
YARN
Kubernetes
HDFS
Amazon S3
Hypertable
HBase
etc…

Spark vs. Hadoop

- Spark把中间数据放在内存中，迭代运算效率高；MapReduce中计算结果需要落地，保存到磁盘上。Spark支持DAG图的分布式并行计算，减少了迭代过程中数据的落地，提高了处理效率。
- Spark容错性高。Spark引入了RDD的抽象，它是分布在一组节点中的只读对象集合，这些集合是弹性的。在RDD计算时可以通过CheckPoint来实现容错，而CheckPoint有两种方式：CheckPoint Data和Logging The Updates。
- Spark更加通用。Hadoop只提供了Map和Reduce两种操作，Spark提供的数据集操作类型很多。另外各个处理节点之间的通信模型不再像Hadoop只有Shuffle一种，用户可以命名、物化、控制中间结果的存储、分区等。

Spark vs. Hadoop

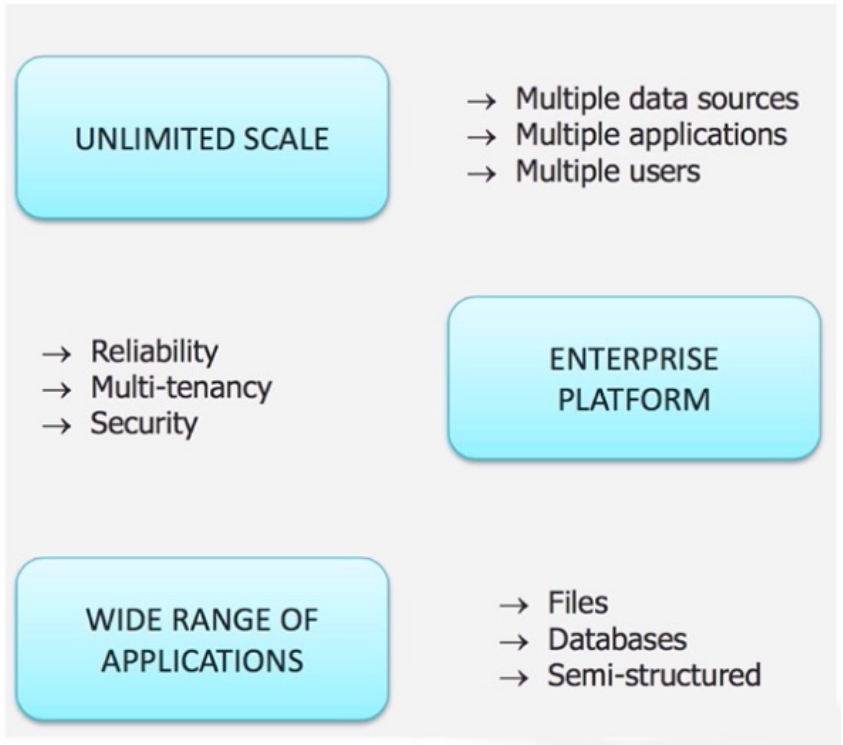
Hadoop	Spark
Distributed Storage + Distributed Compute	Distributed Compute Only
MapReduce framework	Generalized computation
Usually data on disk (HDFS)	On disk / in memory
Not ideal for iterative work	Great at Iterative workloads (machine learning ..etc)
Batch process	- Upto 2x - 10x faster for data on disk - Upto 100x faster for data in memory
	Compact code Java, Python, Scala supported

Spark vs. Hadoop

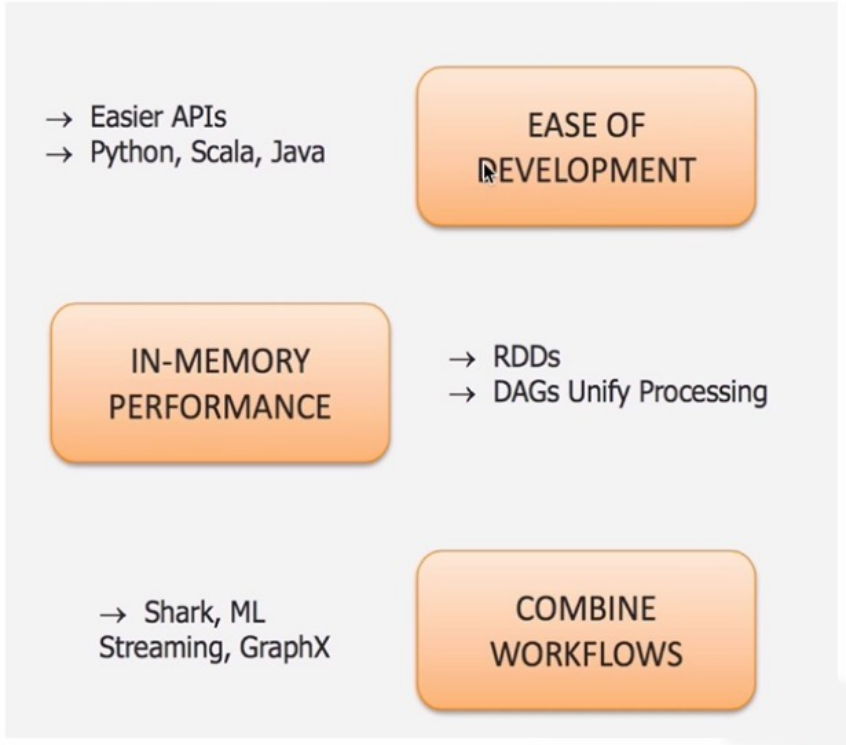
Use Case	Other	Spark
Batch processing	Hadoop's MapReduce (Java, Pig, Hive)	Spark RDDs (java / scala / python)
SQL querying	Hadoop : Hive	Spark SQL
Stream Processing / Real Time processing	Storm Kafka	Spark Streaming
Machine Learning	Mahout	Spark ML Lib
Real time lookups	NoSQL (Hbase, Cassandra ..etc)	No Spark component. But Spark can query data in NoSQL stores

Spark和Hadoop的协作性

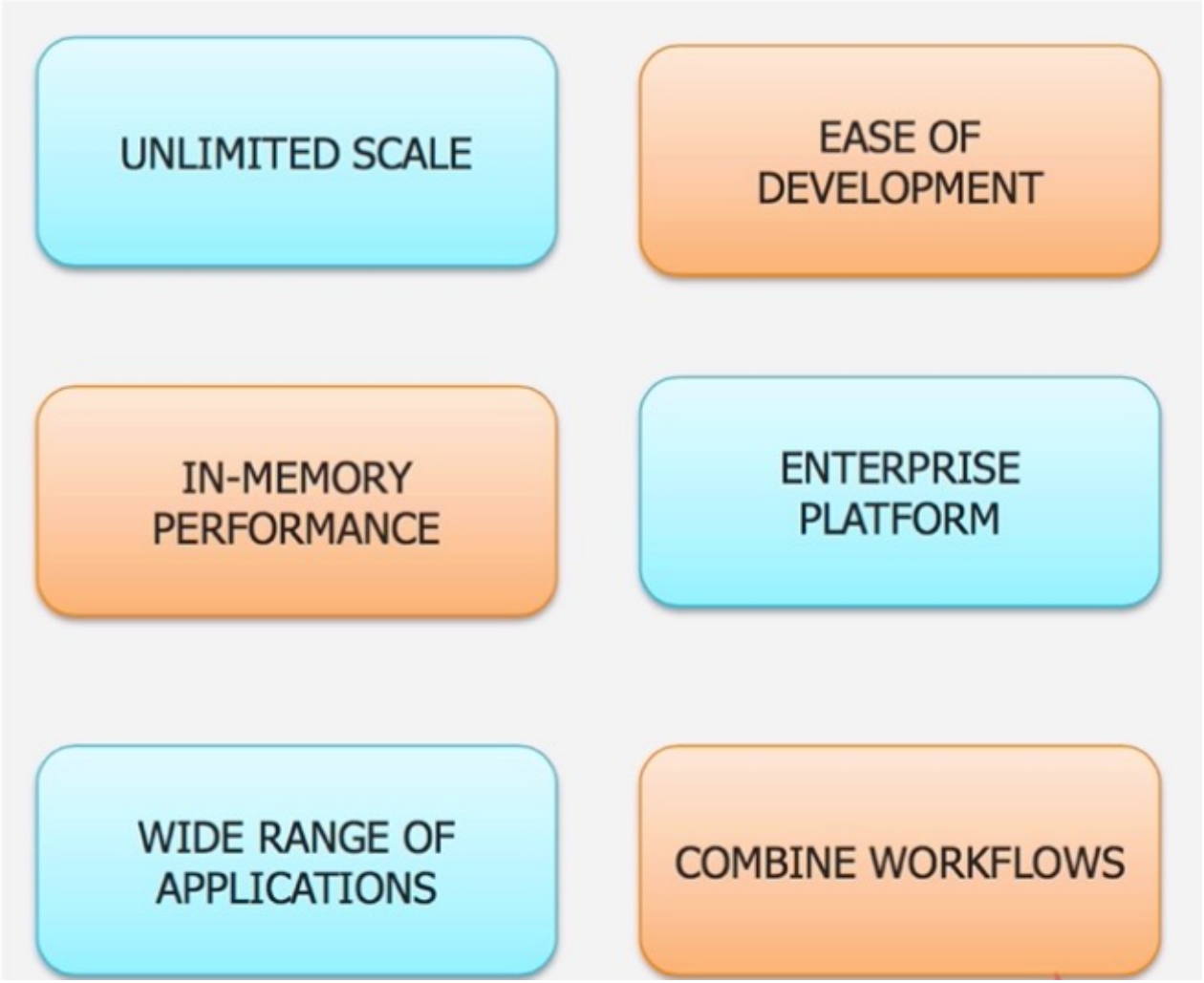
- Hadoop的优势



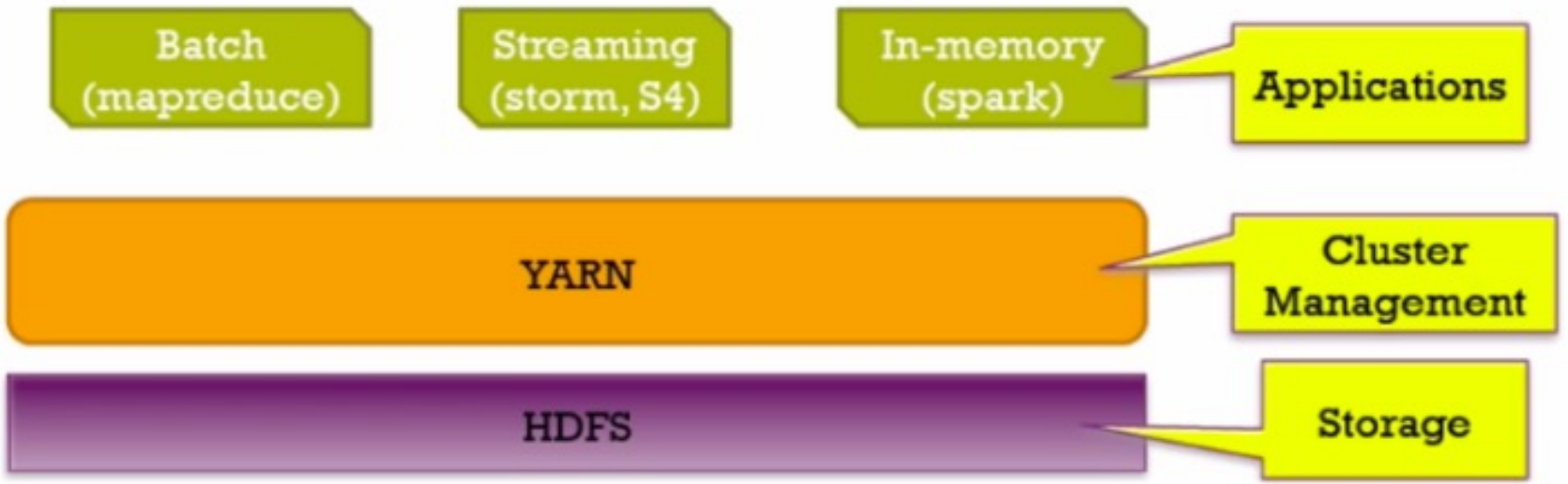
- Spark的优势



Spark和Hadoop的协作性



Spark和Hadoop的协作性



Spark Core

- 提供有向无环图DAG的分布式并行计算框架，并提供Cache机制来支持多次迭代计算或者数据共享。
- 引入RDD抽象，它是分布在一组节点中的只读对象集合，这些集合是弹性的，如果数据集一部分丢失，则可以根据Lineage“血统”对它们进行重建，保证了数据的高容错性。
- 移动计算而非移动数据，RDD Partition可以就近读取分布式文件系统中的数据块到各个节点内存中进行计算。
- 使用多线程池模型来减少task启动开销
- 采用容错的、高可伸缩性的akka作为通信框架

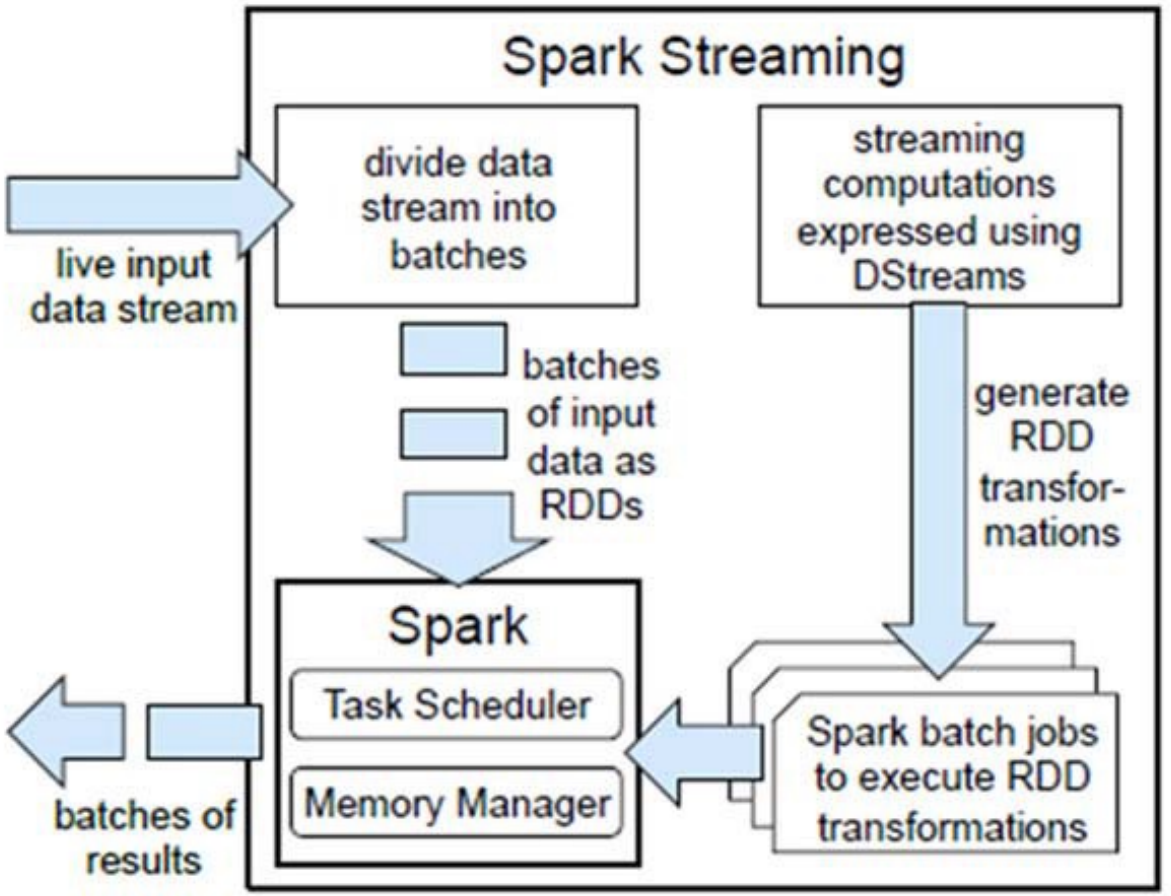
Spark Streaming

- Spark Streaming是一个对实时数据流进行高吞吐量、容错处理的流式处理系统，可以对多种数据源（如Kafka、Flume、Twitter、Zero和TCP 套接字）进行类似Map、Reduce和Join等复杂操作，并将结果保存到外部文件系统、数据库或应用到实时仪表盘。

Spark Streaming

- Spark Streaming 的工作机制是对数据流进行分片，使用Spark计算引擎处理分片数据，并返回相应分片的计算结果。
- Spark Streaming 提供的基本流式数据抽象叫discretized stream，或称DStream。DStream由一系列连续的RDD表示（每个数据流分片被表示为一个RDD），对DStream的操作被转换成对相应RDD序列的操作。

Spark Streaming



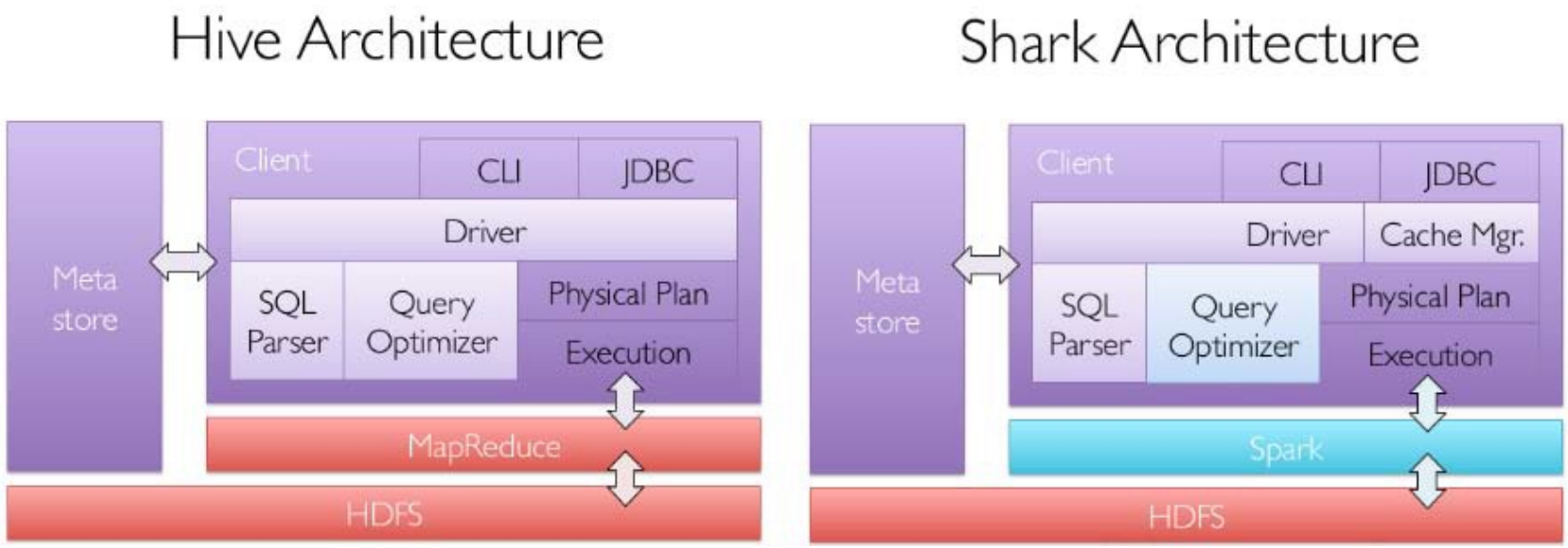
Spark SQL

- Shark，即Hive on Spark，本质上是通过Hive的HQL解析，把HQL翻译成Spark上的RDD操作，然后通过Hive的metadata获取数据库里的表信息，实际HDFS上的数据和文件，会由Shark获取并放到Spark上运算。与Hive完全兼容。
- 2014.7.1，Shark → Spark SQL
- Spark SQL，允许直接处理RDD，同时也可查询例如在Hive上存在的外部数据。能够统一处理关系表和RDD，使得开发人员可以轻松地使用SQL命令进行外部查询，同时进行更复杂的数据分析。

Spark SQL

- Spark SQL 是一个用来处理结构化数据的分布式SQL查询引擎，具有以下几个特点：
  - 与Spark程序无缝对接。使用集成的API，Spark SQL允许使用RDD模型来查询结构化数据，这使得在复杂程序里运行SQL查询变得容易。
  - 统一数据访问接口。Spark SQL提供统一的接口来访问各种结构化数据，包括Hive、Parquet和Json文件。
  - 与Hive高度兼容。对已经存在的Hive数据、Hive查询语句和UDFs等，Spark SQL都可以完美兼容，方便了应用迁移。
  - 使用标准链接。Spark SQL可以使用工业标准JDBC和ODBC进行链接，减小了开发人员的学习成本。

Spark SQL

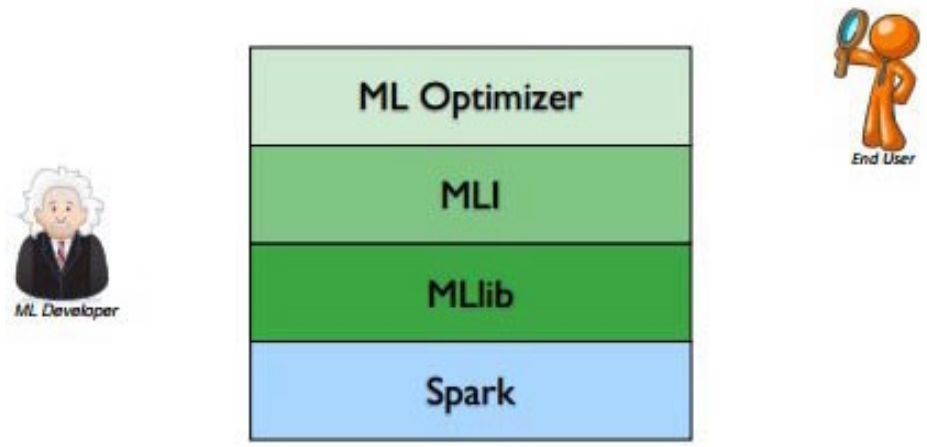

Hive Architecture
Meta store
Client
CLI
JDBC
Driver
SQL Parser
Query Optimizer
Physical Plan
Execution
MapReduce
HDFS
Shark Architecture
Meta store
Client
CLI
JDBC
Driver
Cache Mgr.
SQL Parser
Query Optimizer
Physical Plan
Execution
Spark
HDFS

Spark SQL

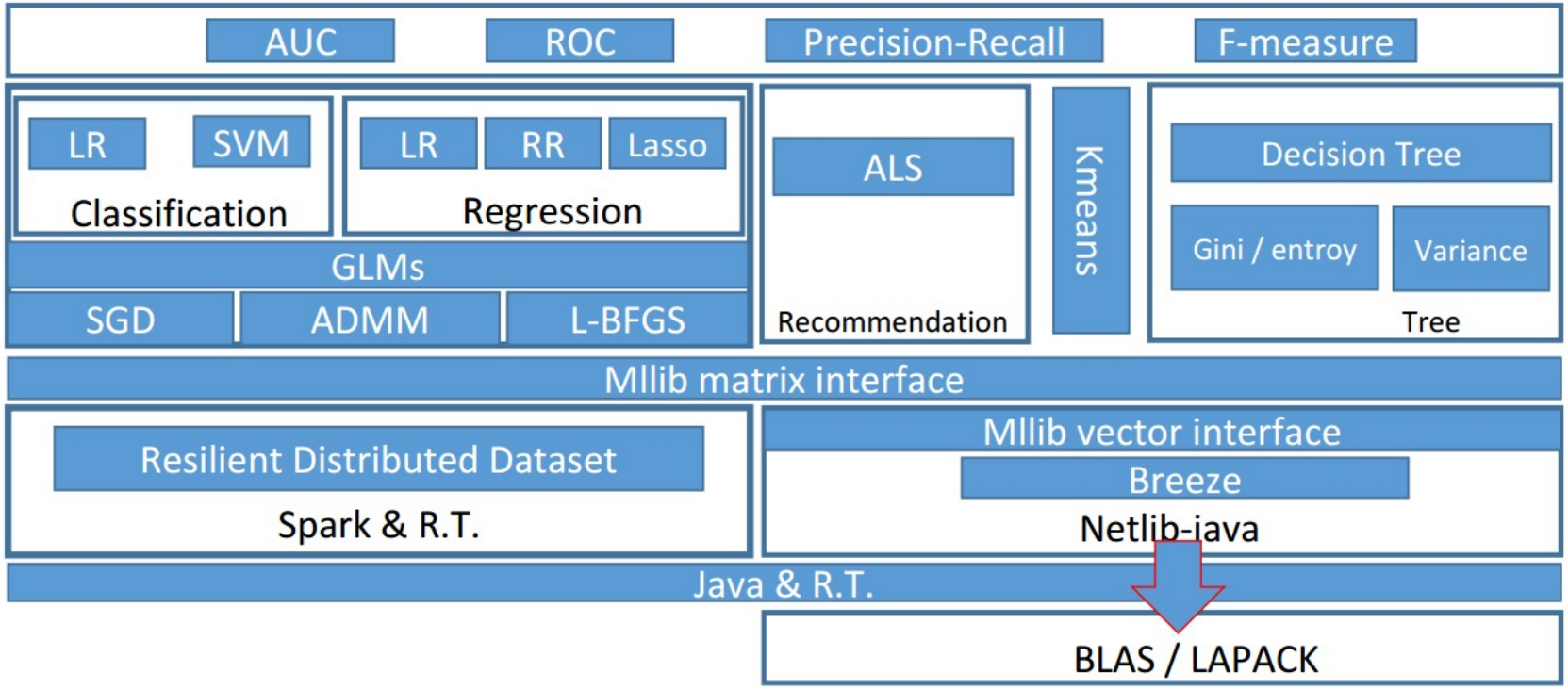
- 性能提升原因：
  - 内存列存储（In-Memory Columnar Storage）： Spark SQL的表数据在内存中存储不是采用原生态的JVM对象存储方式，而是采用内存列存储；
  - 字节码生成技术（Bytecode Generation）： Spark1.1.0在Catalyst模块的expressions增加了codegen模块，使用动态字节码生成技术，对匹配的表达式采用特定的代码动态编译。另外对SQL表达式都作了CG优化， CG优化的实现主要还是依靠Scala2.10的运行时反射机制（runtime reflection）；
  - Scala代码优化： Spark SQL在使用Scala编写代码的时候，尽量避免低效的、容易GC的代码；尽管增加了编写代码的难度，但对于用户来说接口统一。

Spark MLlib

- ML Optimizer会选择它认为最适合的已经在内部实现好了的机器学习算法和相关参数，来处理用户输入的数据，并返回模型或别的帮助分析的结果；
- MLI 是一个进行特征抽取和高级ML编程抽象的算法实现的API或平台；
- MLlib是Spark实现一些常见的机器学习算法和实用程序，包括分类、回归、聚类、协同过滤、降维以及底层优化，算法可以进行扩充；
- MLRuntime 基于Spark计算框架，将Spark的分布式计算应用到机器学习领域。MLlib面向RDD。



Spark MLlib


AUC
ROC
Precision-Recall
F-measure
LR
SVM
Classification
LR
RR
Lasso
Regression
GLMs
SGD
ADMM
L-BFGS
ALS
Recommendation
Kmeans
Decision Tree
Gini / entroy
Variance
Tree
Mllib matrix interface
Resilient Distributed Dataset
Spark & R.T.
Mllib vector interface
Breeze
Netlib-java
Java & R.T.
BLAS / LAPACK

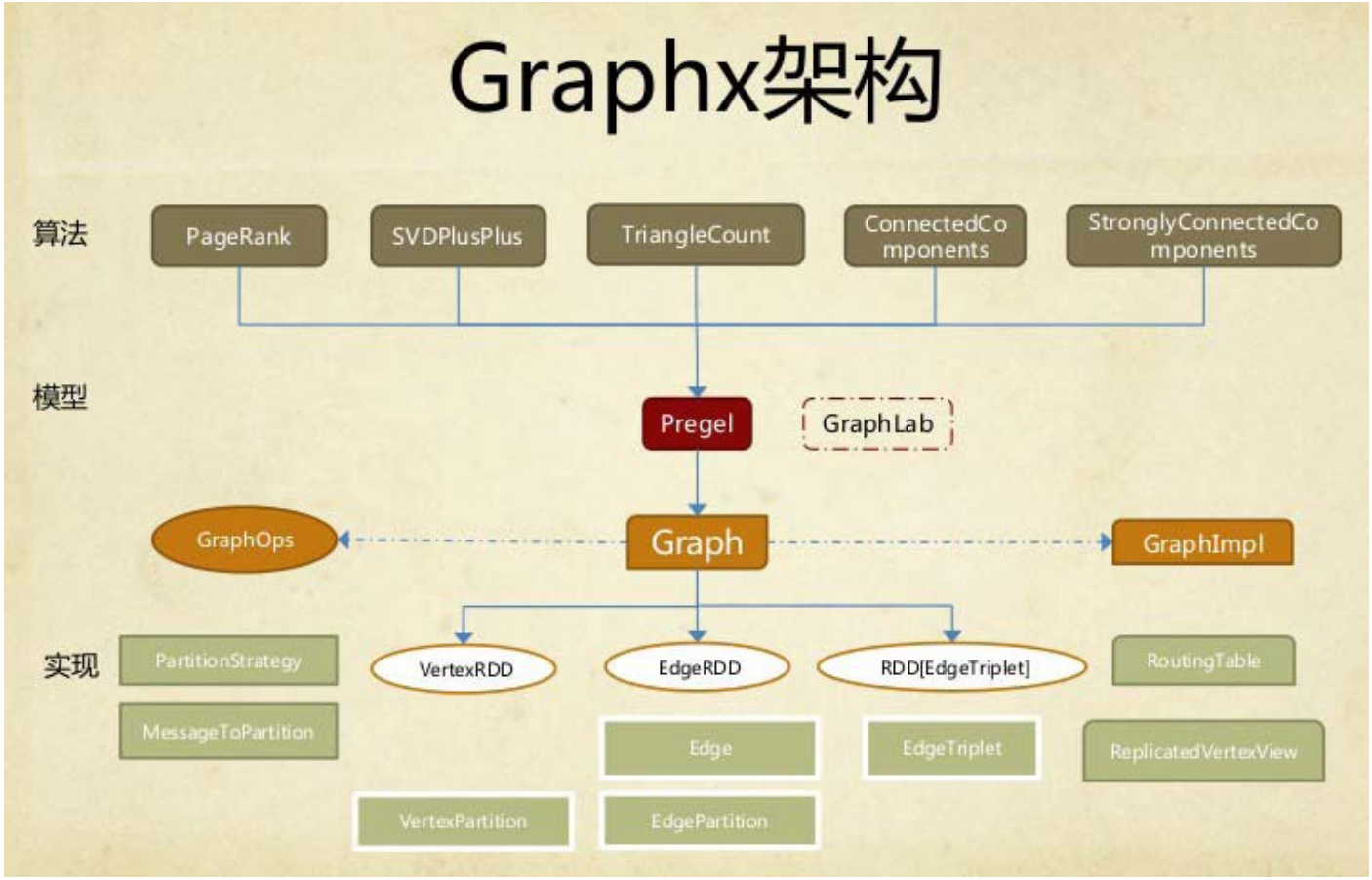
Spark ML

- 从Spark 2.0开始，基于RDD的API（spark.mllib）进入维护模式，取而代之的是基于DataFrame的API（spark.ml）
- Spark ML 是基于DataFrame进行机器学习API的开发，抽象层次更高。把数据处理的流水线抽象出来，算法相当于流水线的一个组件，可以被其他算法随意的替换，这样就让算法和数据处理的其他流程分割开来，实现低耦合。
- http://spark.apache.org/docs/latest/ml-guide.html

GraphX

- GraphX是Spark中用于图(e.g. Web-Graphs and Social Networks)和图并行计算(e.g. PageRank and Collaborative Filtering)的API。
- GraphX的核心抽象是Resilient Distributed Property Graph，一种点和边都带属性的有向多重图。它扩展了Spark RDD的抽象，有Table和Graph两种视图，而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。

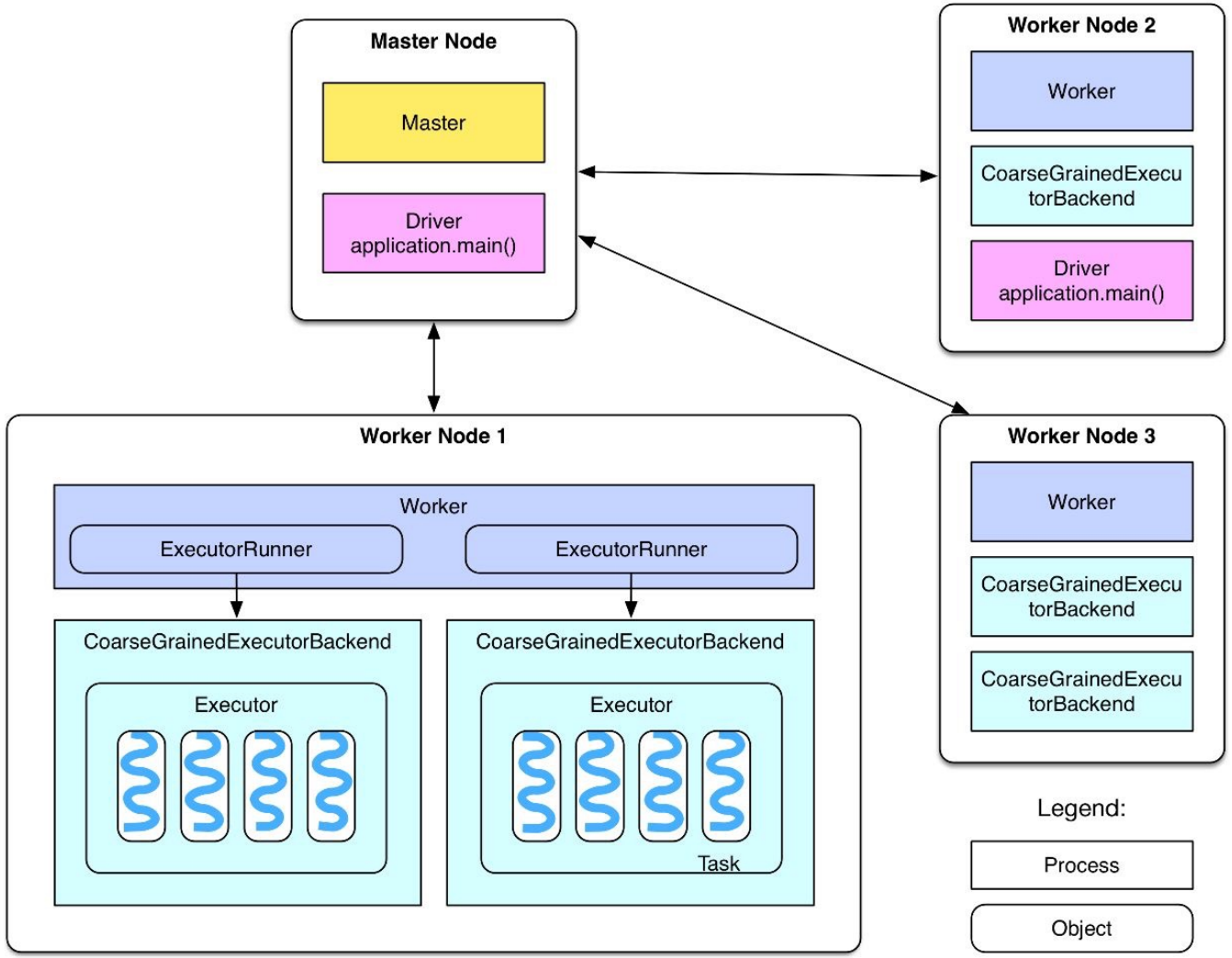
GraphX


Graphx架构
算法
PageRank
SVDPlusPlus
TriangleCount
ConnectedComponents
StronglyConnectedComponents
模型
Pregel
GraphLab
GraphOps
Graph
GraphImpl
实现
PartitionStrategy
MessageToPartition
VertexRDD
EdgeRDD
RDD[EdgeTriplet]
RoutingTable
Edge
EdgeTriplet
ReplicatedVertexView
VertexPartition
EdgePartition

摘要

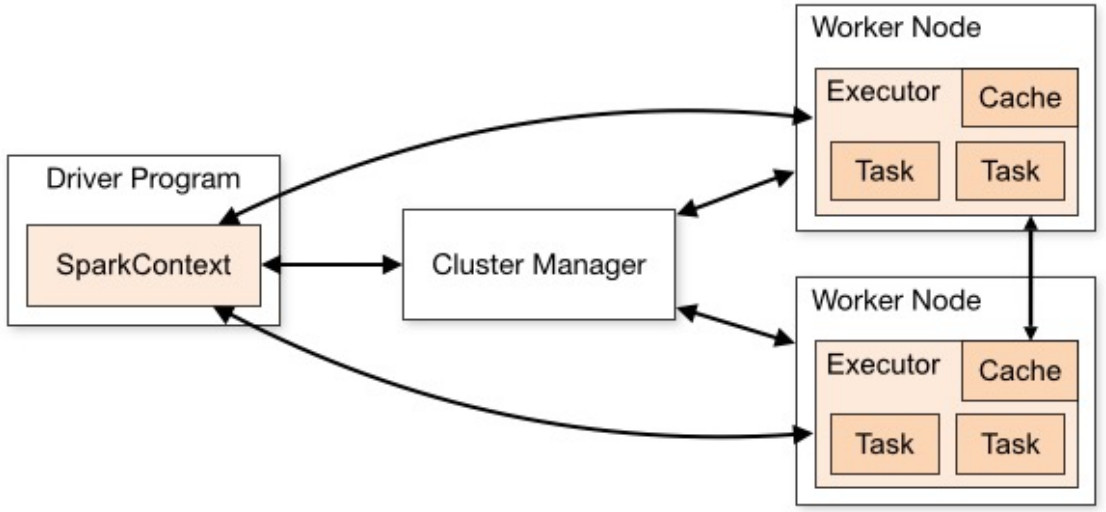
- 为什么会有Spark?
- Spark的生态圈
- Spark的基本构架和组件
- Spark的程序执行过程
- Spark的技术特点

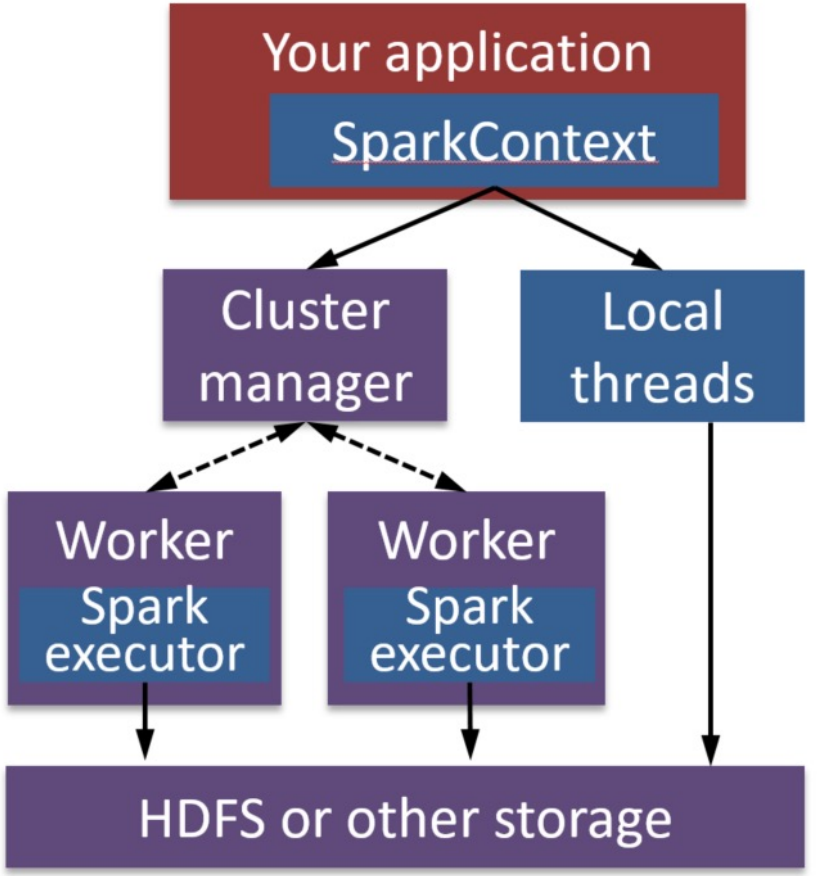
Spark集群整体运行架构



- Master node：是集群部署时的概念，是整个集群的控制器，负责整个集群的正常运行，管理Worker node。
- Worker node：是计算节点，接收主节点命令与进行状态汇报
- Executors：每个Worker上有一个Executor，负责完成Task程序的执行
- Spark集群部署后，需要在主从节点启动Master进程和Worker进程，对整个集群进行控制

Spark的基本构架和组件





- Spark runs as library in your program (1 instance per app)
- Run tasks locally or on cluster
  - YARN, Kubernetes, or standalone mode
- Access storage systems via Hadoop InputFormat API
  - Can use HBase, HDFS, S3…

Spark的基本构架和组件

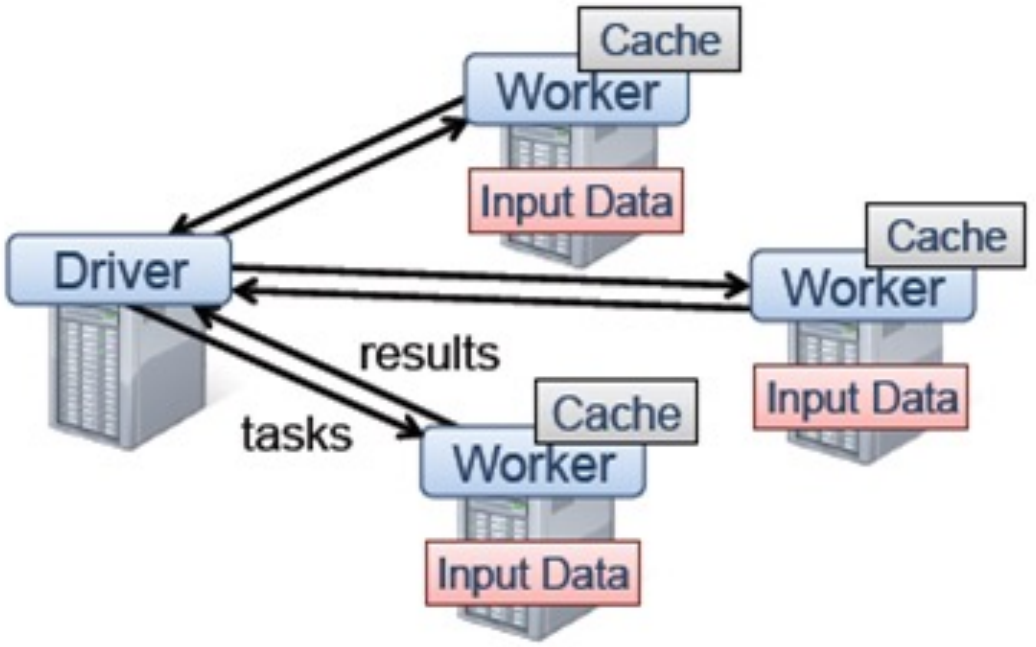
- **Application**：基于 Spark 的用户程序，即由用户编写的调用 Spark API 的应用程序，它由集群上的一个驱动（Driver）程序和多个执行器（Executor）程序组成。其中应用程序的入口为用户所定义的 main 方法。
- **SparkContext**：是 Spark 所有功能的主要入口点，它是用户逻辑与 Spark 集群主要的交互接口。通过**SparkContext**，可以连接到集群管理器（**Cluster Manager**），能够直接与集群 Master 节点进行交互，并能够向 Master 节点申请计算资源，也能够将应用程序用到的 JAR 包或 Python 文件发送到多个执行器（**Executor**）节点上。
- **Cluster Manager**：即集群管理器，它存在于 Master 进程中，主要用来对应用程序申请的资源进行管理。

Spark的基本构架和组件

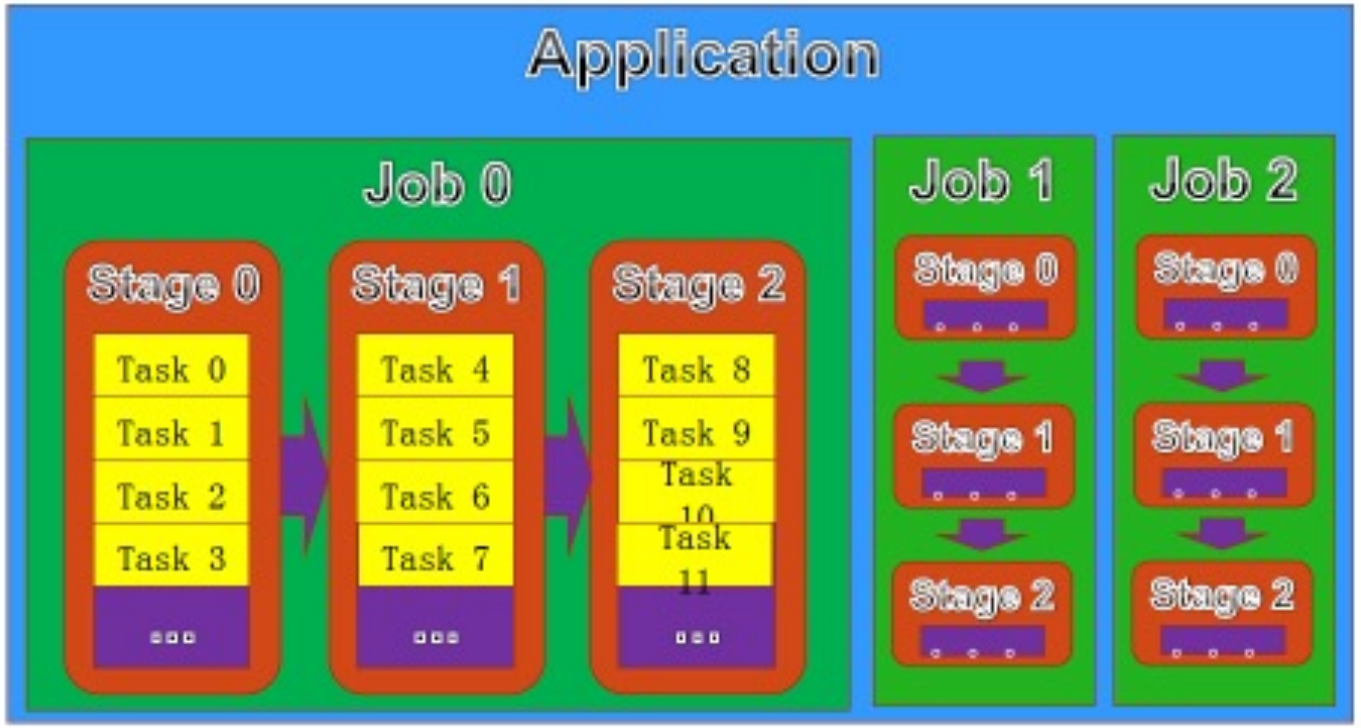
- **Worker Node**：任何能够在集群中运行 Spark 应用程序的节点。
- **Task**：由**SparkContext**发送到**Executor**节点上执行的一个工作单元。
- **Driver**：也即驱动器节点，它是一个运行**Application**中**main()**函数并创建**SparkContext**的进程。**Driver**节点也负责提交**Job**，并将**Job**转化为**Task**，在各个**Executor**进程间协调 **Task** 的调度。**Driver**节点可以不运行于集群节点机器上。
- **Executor**：也即执行器节点，它是在一个在工作节点（**Worker Node**）上为**Application**启动的进程，它能够运行 **Task** 并将数据保存在内存或磁盘存储中，也能够将结果数据返回给**Driver**。

Spark的基本构架和组件

- 在Spark应用程序执行过程中，Driver和Worker扮演着最重要的角色
  - Driver 是应用执行起点，负责作业调度
  - Worker管理计算节点及创建并行处理任务
  - Cache存储中间结果等
  - Input Data为输入数据

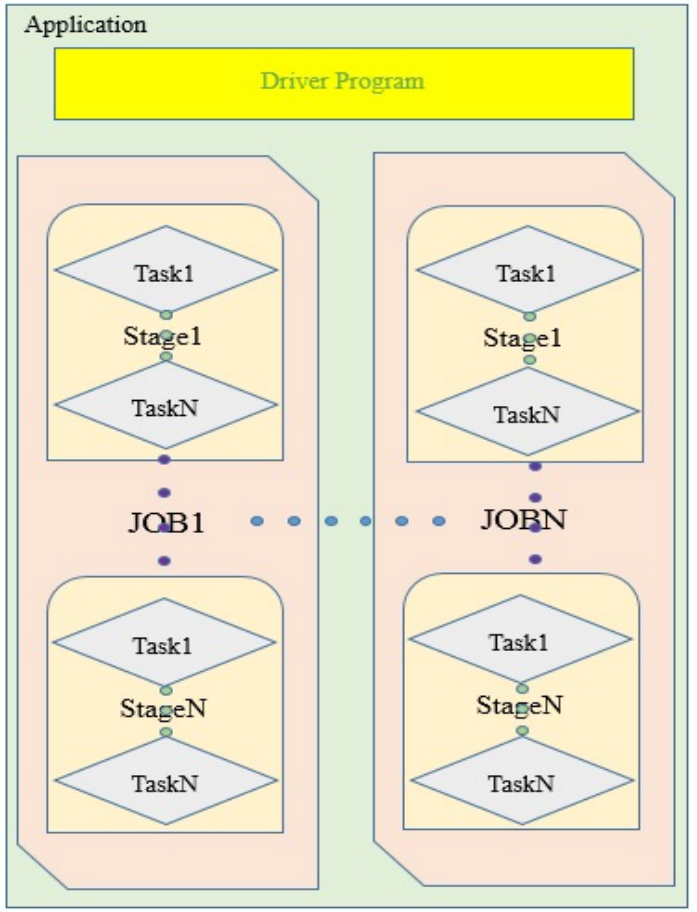


Spark应用程序的组成结构



Spark应用程序的组成结构

- Application：基于Spark的应用程序，包含了一个Driver Program和多个executor（worker中）
- Job：包含多个Task的并行计算，由Spark Action催生
- Stage：Job拆分成多组Task，每组任务被称为Stage，也可称为TaskSet
- Task：基本程序执行单元，在一个Executor上执行



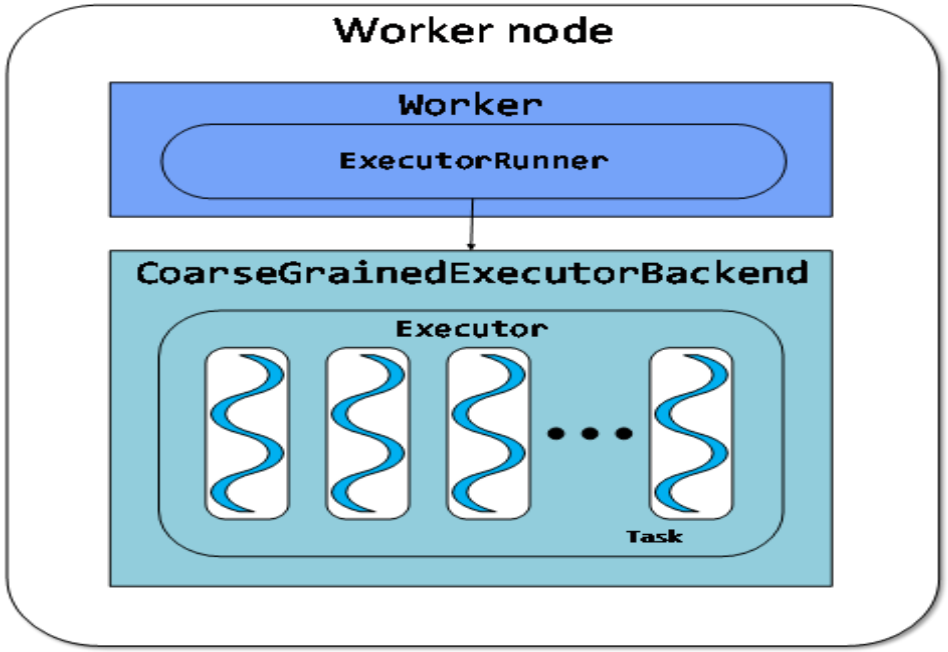
Spark应用程序的组成结构

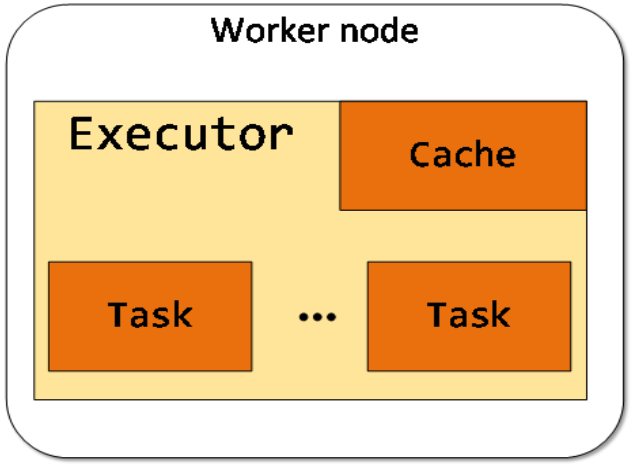
- SparkContext：SparkContext由用户程序启动，是Spark运行的核心模块，它对一个Spark程序进行了必要的初始化过程，其中包括了：
  - 创建SparkConf类的实例：这个类中包含了用户自定义的参数信息和Spark配置文件中的一些信息等等(用户名、程序名、Spark版本等)
  - 创建SparkEnv类的实例：这个类中包含了Spark执行时所需要的许多环境对象，例如底层任务通讯的Akka Actor System、block manager、serializer等
  - 创建调度类的实例：Spark中的调度分为TaskScheduler和DAGScheduler两种，而它们的创建都在SparkContext的初始化过程中。

Spark Driver的组成

- Driver：执行Application的main()函数并创建SparkContext。
- RDD：Spark基本计算单元，一组RDD形成可执行的有向无环图（操作主要有：Transformation和Action）
- DAG Scheduler：根据Job构建的基于Stage的DAG，并提交Stage给TaskScheduler
- TaskScheduler：将Task分发给Executor执行
- SparkEnv：线程级别运行环境，存储运行时的重要组件的引用，创建并包含了：
  - MapOutPutTracker：存储Shuffle元信息
  - BroadcastManager：控制广播变量并存储其元信息
  - BlockManager：存储管理、创建和查找块
  - MetricsSystem：监控运行时性能指标信息
  - SparkConf：存储配置信息

Worker node的结构





此处Cache部分即为Spark编程模型中对RDD进行内存持久化存储的部位。

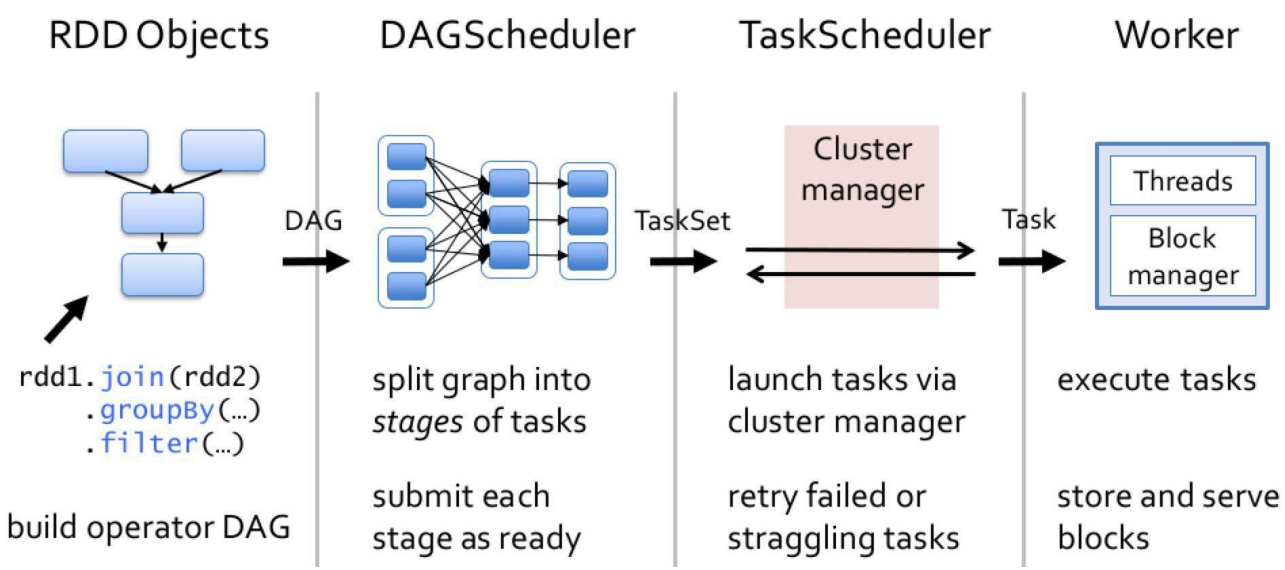
- 在Standalone模式中，ExecutorBackend被实例化成 CoarseGrainedExecutorBackend 进程。
- Spark是一个多线程模型。CoarseGrainedExecutorBackend进程包含一个Executor对象，该对象持有一个线程池，每个线程可以执行一个task。同一个Executor进程内，多个task之间可以共享内存资源。
- 对同一个Application，它在一个Worker上只能拥有一个Executor，Worker与Executor之间是一一对应的关系，而每个Worker node可以有多个Worker。
- Worker通过持有ExecutorRunner对象来控制CoarseGrainedExecutorBackend的启停。

Spark调度器

- Spark 中主要有两种调度器：DAGScheduler 和 TaskScheduler，DAGScheduler 主要是把一个 Job 根据 RDD 间的依赖关系，划分为多个 Stage，对于划分后的每个 Stage 都抽象为一个由多个 Task 组成的任务集（TaskSet），并交给 TaskScheduler 来进行进一步的任务调度。TaskScheduler 负责对每个具体的 Task 进行调度。

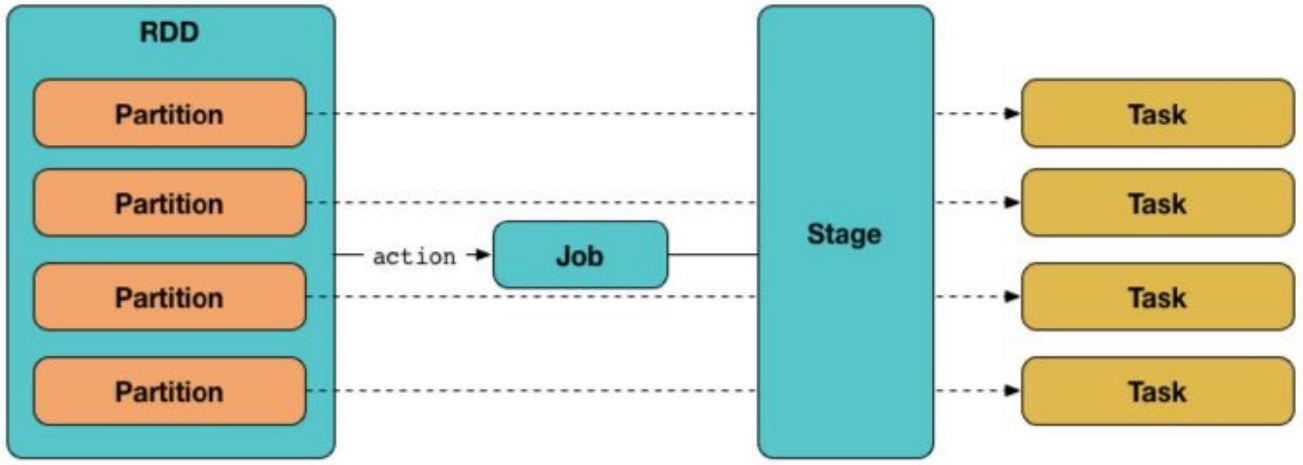
Spark RDD调度过程

- Spark 对 RDD 执行调度的过程，创建 RDD 并生成 DAG，由 DAGScheduler 分解 DAG 为包含多个 Task（即 TaskSet）的 Stages，再将 TaskSet 发送至 TaskScheduler，由 TaskScheduler 来调度每个 Task，并分配到 Worker 节点上执行，最后得到计算结果。



DAGScheduler

- 当创建一个 RDD 时，每个 RDD 中包含一个或多个分区，当执行 Action 操作时，相应的产生一个 Job，而一个 Job 会根据 RDD 间的依赖关系分解为多个 Stage，每个 Stage 由多个 Task 组成（即 TaskSet），每个 Task 处理 RDD 中的一个 Partition。一个 Stage 里面所有分区的任务集合被包装为一个 TaskSet 交给 TaskScheduler 来进行任务调度。这个过程是由 DAGScheduler 来完成的。



TaskScheduler

- DAGScheduler 将一个 TaskSet 交给 TaskScheduler 后，TaskScheduler 会为每个 TaskSet 进行任务调度，Spark 中的任务调度分为两种：FIFO（先进先出）调度和 FAIR（公平调度）调度。
  - FIFO调度：即谁先提交谁先执行，后面的任务需要等待前面的任务执行。这是 Spark 默认的调度模式。
  - FAIR调度：支持将作业分组到池中，并为每个池设置不同的调度权重，任务可以按照权重来决定执行顺序。
  - spark.scheduler.mode 可选FIFO或FAIR，默认是FIFO

FIFO调度算法

FIFOSchedulingAlgorithm

```
override def comparator(s1: Schedulable, s2: Schedulable): Boolean = {
    val priority1 = s1.priority  // priority实际为Job ID
    val priority2 = s2.priority
    var res = math.signum(priority1 - priority2)
    if (res == 0) {
      val stageId1 = s1.stageId
      val stageId2 = s2.stageId
      res = math.signum(stageId1 - stageId2)
    }
    res < 0
  }
```

FAIR调度算法

FairSchedulingAlgorithm

```
override def comparator(s1: Schedulable, s2: Schedulable): Boolean = {
    val minShare1 = s1.minShare
    val minShare2 = s2.minShare
    val runningTasks1 = s1.runningTasks
    val runningTasks2 = s2.runningTasks
    val s1Needy = runningTasks1 < minShare1
    val s2Needy = runningTasks2 < minShare2
    val minShareRatio1 = runningTasks1.toDouble / math.max(minShare1, 1.0)
    val minShareRatio2 = runningTasks2.toDouble / math.max(minShare2, 1.0)
    val taskToWeightRatio1 = runningTasks1.toDouble / s1.weight.toDouble
    val taskToWeightRatio2 = runningTasks2.toDouble / s2.weight.toDouble
```

```
  var compare = 0
  if (s1Needy && !s2Needy) {
    return true
  } else if (!s1Needy && s2Needy) {
    return false
  } else if (s1Needy && s2Needy) {
    compare = minShareRatio1.compareTo(minShareRatio2)
  } else {
    compare = taskToWeightRatio1.compareTo(taskToWeightRatio2)
  }
  if (compare < 0) {
    true
  } else if (compare > 0) {
    false
  } else {
    s1.name < s2.name
  }
}
```

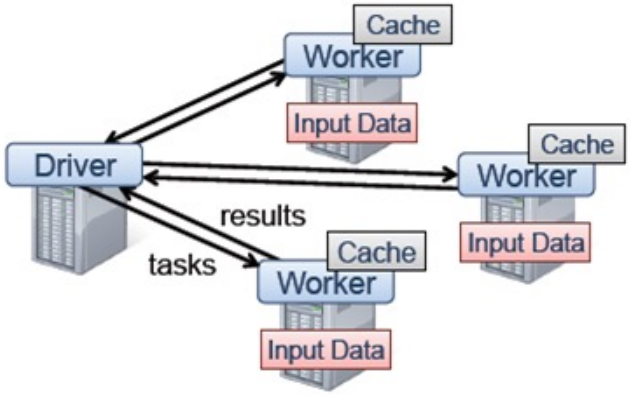
- weight：控制池在集群中的份额，默认情况下，所有池的权值为1；
- minShare：最小CPU核心数，默认为0。权重相同时，越小可以获得更多的资源。

摘要

- 为什么会有Spark?
- Spark的生态圈
- Spark的基本构架和组件
- Spark的程序执行过程
- Spark的技术特点

Spark程序执行过程

- 1. 用户编写的Spark程序提交到相应的Spark运行框架中。
- 2. Spark创建SparkContext作为本次程序的运行环境。
- 3. SparkContext连接相应的集群配置，来确定程序的资源配置使用情况。
- 4. 连接集群资源成功后，Spark获取当前集群上存在Executor的节点，即当前集群中Spark部署的子节点中处于活动并且可用状态的节点（Spark准备运行你的程序并且确定数据存储）
- 5. Spark分发程序代码到各个节点。
- 6. 最终，SparkContext发送tasks到各个运行节点来执行



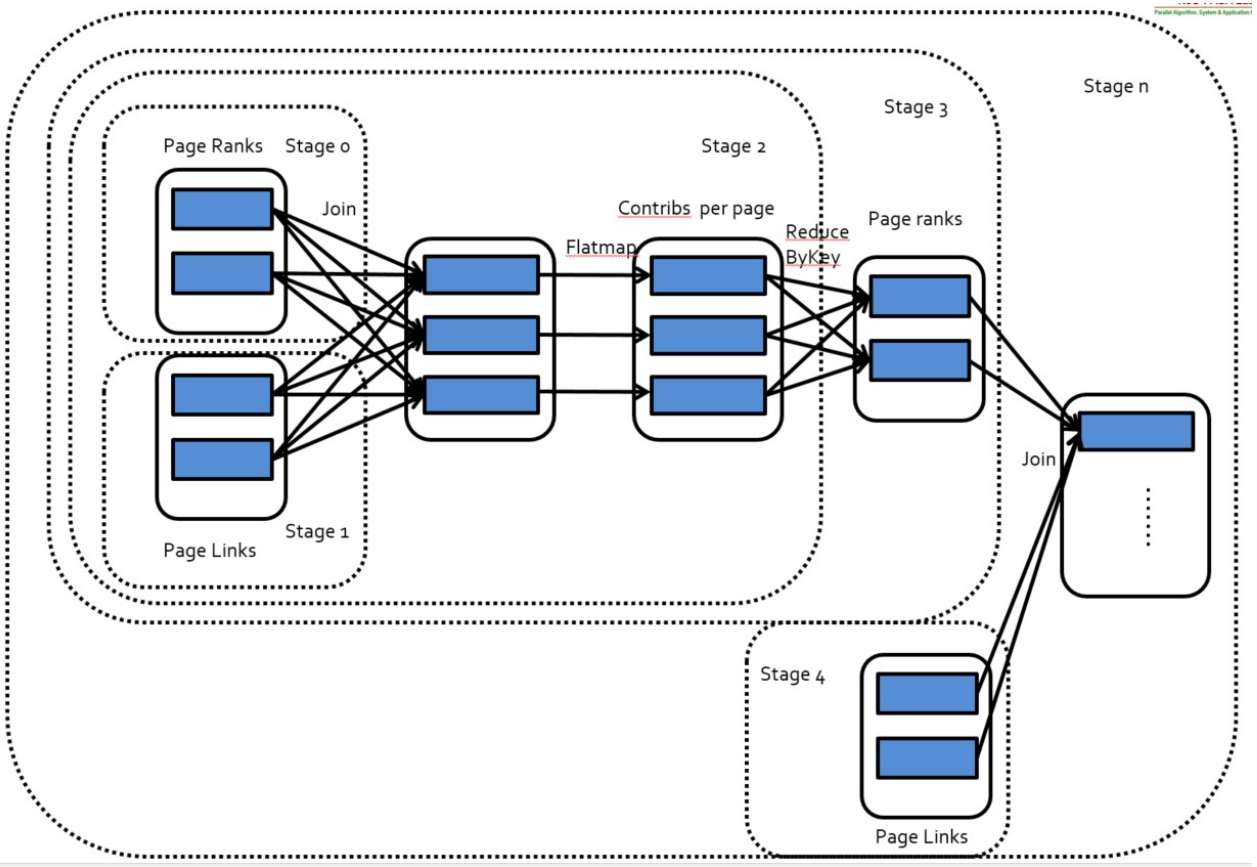
Spark程序执行过程

- 几个基本概念
  - 更详细地说，一个作业（Job）就是一组Transformation操作和一个action操作的集合。每执行一次action操作，那么就会提交一个Job。
  - Stage分为两种，Shuffle Stage和final Stage，每个作业必然只有一个final Stage，即每个action操作会生成一个final stage，如果一个作业中还包含Shuffle操作，那么每进行一次Shuffle操作，便会生成一个Shuffle Stage。
  - Shuffle操作只有在宽依赖的时候才会触发。
  - 任务（Task）作用的单位是Partition，针对同一个Stage，分发到不同的Partition上进行执行。

- 总而言之，Job和Stage是针对一个RDD执行过程的划分，而Task则是具体到了RDD中每个分区的执行

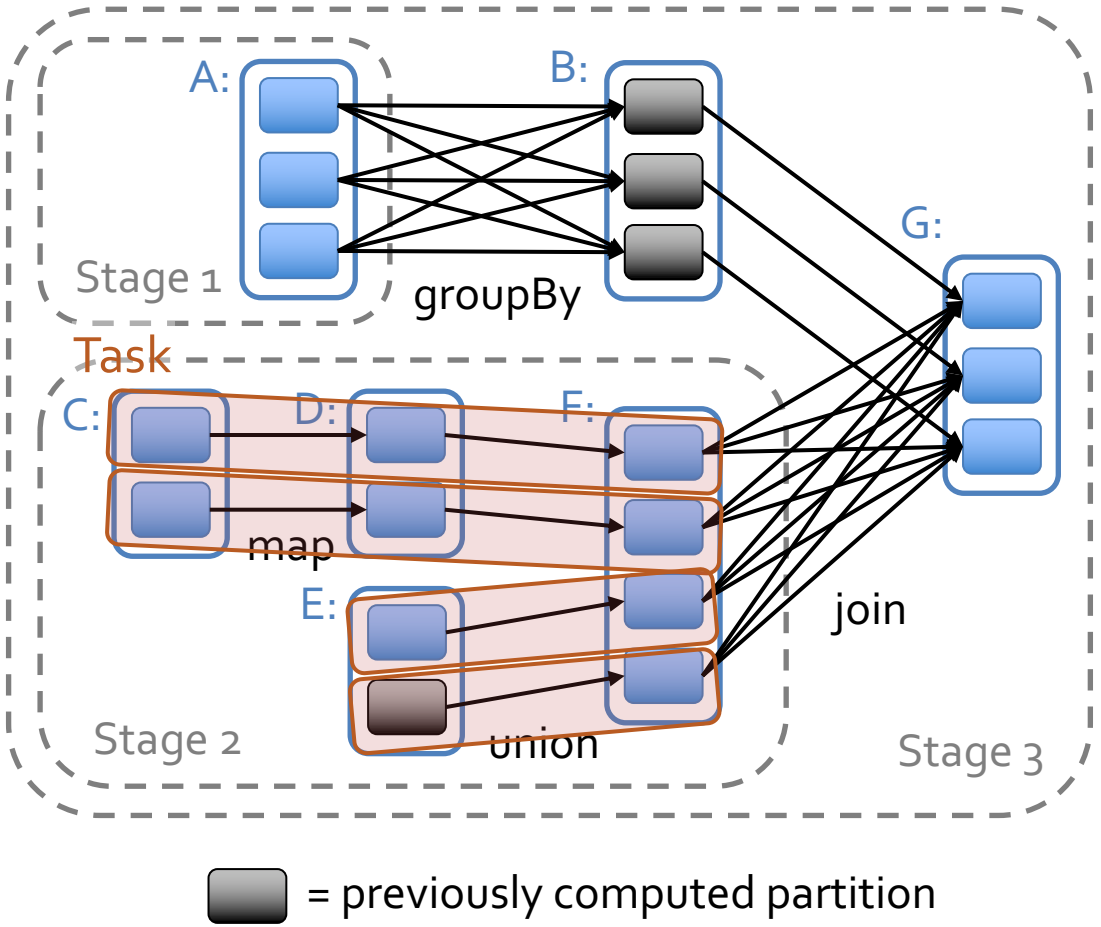
Spark程序执行过程

- 一个作业就是一张RDD世系（Lineage）图：DAG图



Spark程序执行过程

- 构建RDD世系关系的优势
  - 基于RDD世系的并行执行优化
  - 更快速，更细粒度的容错



摘要

- 为什么会有Spark?
- Spark的生态圈
- Spark的基本构架和组件
- Spark的程序执行过程
- Spark的技术特点

Spark的技术特点

- RDD：Spark提出的弹性分布式数据集，是Spark最核心的分布式数据抽象，Spark的很多特性都和RDD密不可分。
- Transformation & Action：Spark通过RDD的两种不同类型的运算实现了惰性计算，即在RDD的Transformation运算时，Spark并没有进行作业的提交；而在RDD的Action操作时才会触发SparkContext提交作业。

Spark的技术特点

- Lineage：为了保证RDD中数据的鲁棒性，Spark系统通过血统关系（lineage）来记录一个RDD是如何通过其他一个或者多个父类RDD转变过来的，当这个RDD的数据丢失时，Spark可以通过它父类的RDD重新计算。
- Spark调度：Spark采用了事件驱动的Scala库类Akka来完成任务的启动，通过复用线程池的方式来取代MapReduce进程或者线程启动和切换的开销。

Spark的技术特点

- API：Spark使用Scala语言进行开发，并且默认Scala作为其编程语言。因此，编写Spark程序比MapReduce程序要简洁得多。同时，Spark系统也支持Java、Python语言进行开发
- Spark生态：Spark SQL、Spark Streaming、GraphX等为Spark的应用提供了丰富的场景和模型，适合应用于不同的计算模式和计算任务
- Spark部署：Spark拥有Standalone、YARN、K8S等多种部署方式，可以部署在多种底层平台上

Spark的技术特点

- 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
- 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
- 数据量不是特别大，但是要求实时统计分析需求

Spark的技术特点

- 综上所述，Spark是一种为大规模数据处理而设计的快速通用的分布式计算引擎，适合于完成一些迭代式、关系查询、流式处理等计算密集型任务。

THANK YOU

南京大学
NANJING UNIVERSITY

南京大学计算机软件研究所
Institute of Computer Software, Nanjing University