《金融大数据处理技术》作业8

191870068 嵇泽同

1.简述Spark的技术特点

①RDD：

RDD是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。

RDD（Resilient Distributed Dataset）叫做**弹性分布式数据集**，**是Spark中最基本的数据抽象**，它代表一个不可变、可分区、里面的元素可并行计算的集合。RDD具有数据流模型的特点：自动容错、位置感知性调度和可伸缩性。RDD允许用户在执行多个查询时显式地将工作集缓存在内存中，后续的查询能够重用工作集，这极大地提升了查询速度。

RDD的属性：

（1）一组分片（Partition），即数据集的基本组成单位。对于RDD来说，每个分片都会被一个计算任务处理，并决定并行计算的粒度。用户可以在创建RDD时指定RDD的分片个数，如果没有指定，那么就会采用默认值。默认值就是程序所分配到的CPU Core的数目。

（2）一个计算每个分区的函数。Spark中RDD的计算是以分片为单位的，每个RDD都会实现compute函数以达到这个目的。compute函数会对迭代器进行复合，不需要保存每次计算的结果。

（3）RDD之间的依赖关系。RDD的每次转换都会生成一个新的RDD，所以RDD之间就会形成类似于流水线一样的前后依赖关系。在部分分区数据丢失时，Spark可以通过这个依赖关系重新计算丢失的分区数据，而不是对RDD的所有分区进行重新计算。

（4）一个Partitioner，即RDD的分片函数。当前Spark中实现了两种类型的分片函数，一个是基于哈希的HashPartitioner，另外一个是基于范围的RangePartitioner。只有对于于key-value的RDD，才会有Partitioner，非key-value的RDD的Parititioner的值是None。Partitioner函数不但决定了RDD本身的分片数量，也决定了parent RDD Shuffle输出时的分片数量。

（5）一个列表，存储存取每个Partition的优先位置（preferred location）。对于一个HDFS文件来说，这个列表保存的就是每个Partition所在的块的位置。按照“移动数据不如移动计算”的理念，Spark在进行任务调度的时候，会尽可能地将计算任务分配到其所要处理数据块的存储位置。

②Transformation & Action：

RDD 的操作分为转化（Transformation）操作和行动（Action）操作。转化操作就是从一个 RDD 产生一个新的 RDD，而行动操作就是进行实际的计算。

Transformation具有**lazy**特性(延迟加载)。Transformation算子的代码不会真正被执行。只有当我们的程序里面遇到一个action算子的时候，代码才会真正的被执行。这种设计让Spark更加有效率地运行。

常用的Transformation有：

转换	含义
map(func)	返回一个新的RDD，该RDD由每一个输入元素经过func函数转换后组成
filter(func)	返回一个新的RDD，该RDD由经过func函数计算后返回值为true的输入元素组成
flatMap(func)	类似于map，但是每一个输入元素可以被映射为0或多个输出元素（所以func应该返回一个序列，而不是单一元素）
mapPartitions(func)	类似于map，但独立地在RDD的每一个分片上运行，因此在类型为T的RDD上运行时，func的函数类型必须是Iterator[T] => Iterator[U]
mapPartitionsWithIndex(func)	类似于mapPartitions，但func带有一个整数参数表示分片的索引值，因此在类型为T的RDD上运行时，func的函数类型必须是(Int, Interator[T]) => Iterator[U]
sample(withReplacement, fraction, seed)	根据fraction指定的比例对数据进行采样，可以选择是否使用随机数进行替换，seed用于指定随机数生成器种子
union(otherDataset)	对源RDD和参数RDD求并集后返回一个新的RDD
intersection(otherDataset)	对源RDD和参数RDD求交集后返回一个新的RDD

distinct([numTasks]))	对源RDD进行去重后返回一个新的RDD
groupByKey([numTasks])	在一个(K,V)的RDD上调用，返回一个(K, Iterator[V])的RDD
reduceByKey(func, [numTasks])	在一个(K,V)的RDD上调用，返回一个(K,V)的RDD，使用指定的reduce函数，将相同key的值聚合到一起，与groupByKey类似，reduce任务的个数可以通过第二个可选的参数来设置
aggregateByKey(zeroValue)(seqOp, combOp, [numTasks])	先按分区聚合 再总的聚合　每次要跟初始值交流 例如：aggregateByKey(0)(_+_,_+_) 对k/y的RDD进行操作
sortByKey([ascending], [numTasks])	在一个(K,V)的RDD上调用，K必须实现Ordered接口，返回一个按照key进行排序的(K,V)的RDD
sortBy(func,[ascending], [numTasks])	与sortByKey类似，但是更灵活 第一个参数是根据什么排序 第二个是怎么排序 false倒序　第三个排序后分区数 默认与原RDD一样
join(otherDataset, [numTasks])	在类型为(K,V)和(K,W)的RDD上调用，返回一个相同key对应的所有元素对在一起的(K,(V,W))的RDD 相当于内连接（求交集）
cogroup(otherDataset, [numTasks])	在类型为(K,V)和(K,W)的RDD上调用，返回一个(K,(Iterable<V>,Iterable<W>))类型的RDD
cartesian(otherDataset)	两个RDD的笛卡尔积 的成很多个K/V
pipe(command, [envVars])	调用外部程序

等等。

Action操作用于执行计算并按指定的方式输出结果。Action操作接受 RDD，但是返回非 RDD，即输出一个值或者结果。在 RDD 执行过程中，真正的计算发生在Action操作。常用的 RDD Action操作有：

函数名	作用	示例	结果
collect()	返回 RDD 的所有元素	rdd.collect()	{1,2,3,3}
count()	RDD 里元素的个数	rdd.count()	4
countByValue()	各元素在 RDD 中的出现次数	rdd.countByValue()	{(1,1),(2,1),(3,2)}
take(num)	从 RDD 中返回 num 个元素	rdd.take(2)	{1,2}
top(num)	从 RDD 中，按照默认（降序）或者指定的排序返回最前面的 num 个元素	rdd.top(2)	{3,3}
reduce()	并行整合所有 RDD 数据，如求和操作	rdd.reduce((x,y)=>x+y)	9
fold(zero)(func)	和 reduce() 功能一样，但需要提供初始值	rdd.fold(0)((x,y)=>x+y)	9
foreach(func)	对 RDD 的每个元素都使用特定函数	rdd1.foreach(x=>println(x))	打印每一个元素
saveAsTextFile(path)	将数据集的元素，以文本的形式保存到文件系统中	rdd1.saveAsTextFile(file://home/test)	
saveAsSequenceFile(path)	将数据集的元素，以顺序文件格式保存到指 定的目录下	saveAsSequenceFile(hdfs://home/test)	

③Lineage：

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。

相比其它系统的细颗粒度的内存数据更新级别的备份或者LOG机制，RDD的Lineage记录的是粗颗粒度的特定数据转换（Transformation）操作（filter, map, join etc.)行为。当这个RDD的部分分区数据丢失时，它可以通过Lineage获取足够的信息来重新运算和恢复丢失的数据分区。这种粗颗粒的数据模型，限制了Spark的运用场合，但同时相比细颗粒度的数据模型，也带来了性能的提升。

RDD在Lineage依赖方面分为两种：Narrow Dependencies与Wide Dependencies，用来解决数据容错时的高效性。

Narrow Dependencies是指父RDD的每一个分区最多被一个子RDD的分区所用，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，也就是说一个父RDD的一个分区不可能对应一个子RDD的多个分区。

Wide Dependencies是指子RDD的分区依赖于父RDD的多个分区或所有分区，也就是说存在一个父RDD的一个分区对应一个子RDD的多个分区。对与Wide Dependencies，这种计算的输入和输出在不同的节点上，lineage方法对与输入节点完好，而输出节点宕机时，通过重新计算，这种情况下，这种方法容错是有效的，否则无效，因为无法重试，需要向上其祖先追溯看是否可以重试（这就是lineage，血统的意思），Narrow Dependencies对于数据的重算开销要远小于Wide Dependencies的数据重算开销。

④Spark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。

⑤API：

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。

⑥Spark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。

⑦Spark部署：

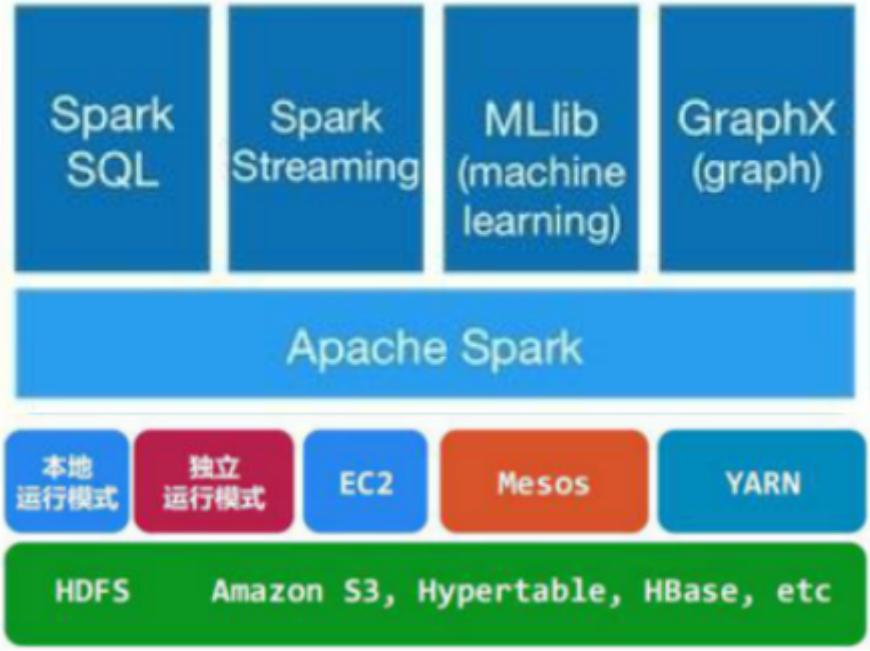
Spark拥有Standalone、Mesos、YARN、K8S等多种部署方式，可以部署在多种底层平台上。

Spark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。

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。

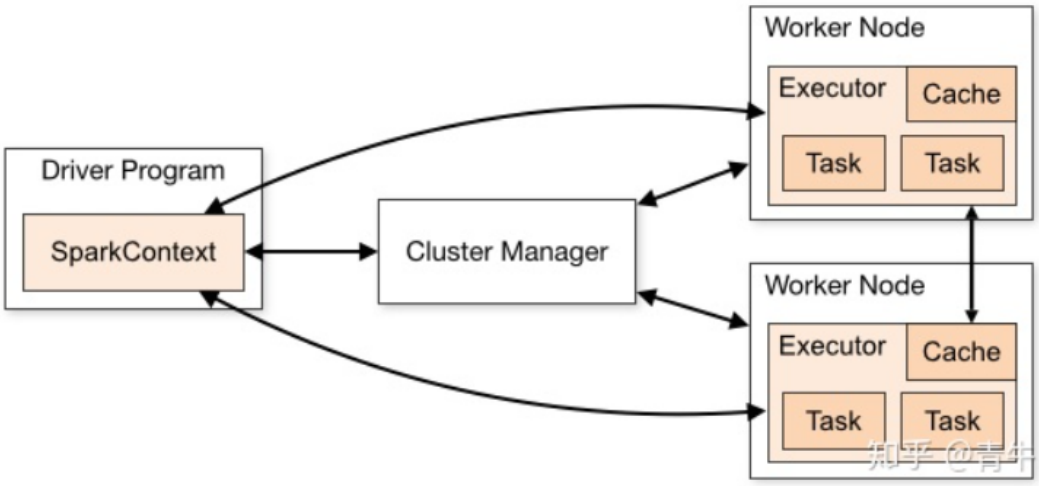
2.简述Spark的基本构架和组件功能

Spark架构及生态如下：



- Spark Core：包含Spark的基本功能；尤其是定义RDD的API、操作以及这两者上的动作。其他Spark的库都是构建在RDD和Spark Core之上的。
- Spark SQL：提供通过Apache Hive的SQL变体Hive查询语言（HiveQL）与Spark进行交互的API。每个数据库表被当做一个RDD，Spark SQL查询被转换为Spark操作。
- Spark Streaming：对实时数据流进行处理和控制。Spark Streaming允许程序能够像普通RDD一样处理实时数据
- MLib：一个常用机器学习的算法库，算法被实现为对RDD的Spark操作。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：控制图、并行图操作和计算的一组算法和工具的集合。GraphX扩展了RDD API，包含控制图、创建子图、访问路径上所有顶点的操作

Spark体系结构可以分为**驱动程序**、**执行程序**和**集群管理器**：



1）驱动程序：驱动程序是运行应用程序的primary（）函数的中心点。它也是Spark Shell（Scala，Python和R）的入口。Spark Context在这里构建。驱动程序的组件包括：

- DAGS调度程序
- 任务计划程序
- 后端调度程序

- 块管理器

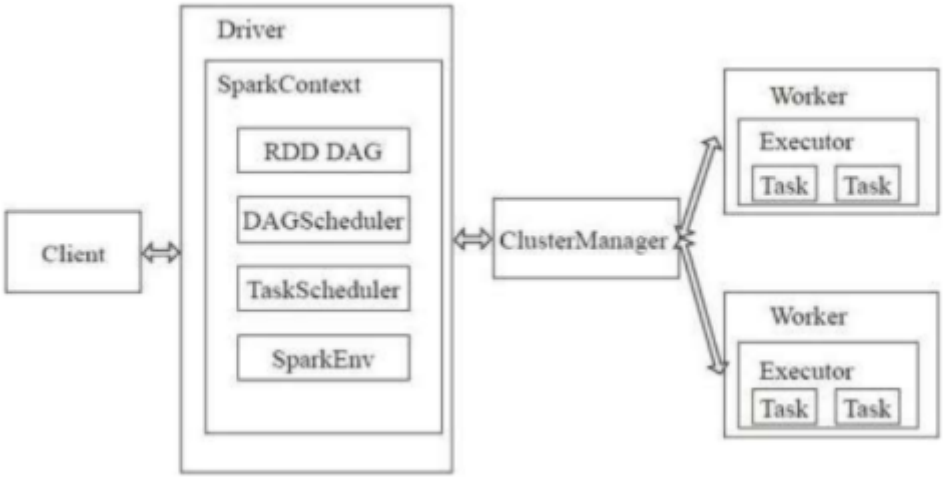
所有这些负责将spark用户代码转换为在集群上实现的实际spark作业。驱动程序的角色是：

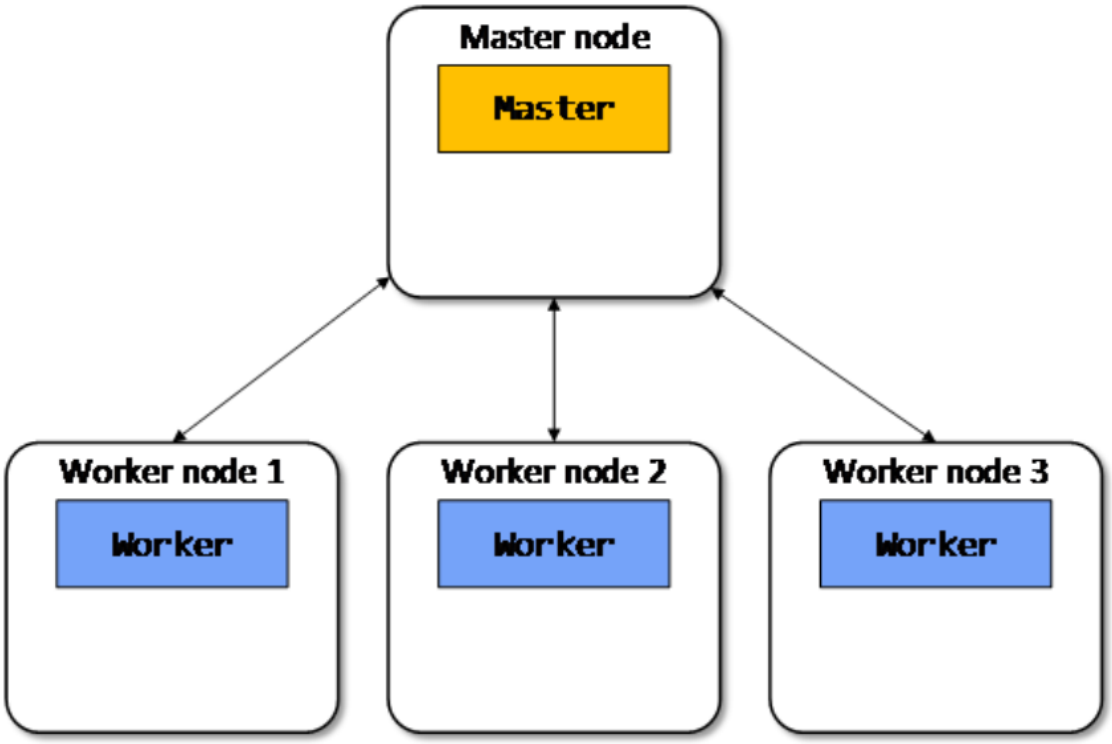
- 在Spark集群的主节点上运行的驱动程序有助于计划作业的执行，并有助于与集群管理器进行协商。
- 它有助于将RDD转换为执行图，然后将其分为多个阶段。
- 有关RDD及其分区的元数据也由驱动程序存储。
- 驱动程序有助于将用户应用程序转换为较小的任务，这些任务又由执行者执行。后一种是处理单个任务的工作进程。
- 它还有助于通过端口4040上的Web UI揭示有关运行spark应用程序的信息。

2）执行者：执行者主要负责执行任务。它是一个分布式代理。每个Spark应用程序都具有自己的流程执行器，该执行器在Spark应用程序的整个生命周期内运行。这称为“执行者的静态分配”。用户还可以选择执行者的动态分配，在其中动态添加或删除执行者以匹配整体工作量。执行者的角色是：

- 数据处理
- 它有助于读取和写入外部数据
- 它有助于将计算结果数据存储在内存，HDD或缓存中。
- 它有助于与存储系统进行交互

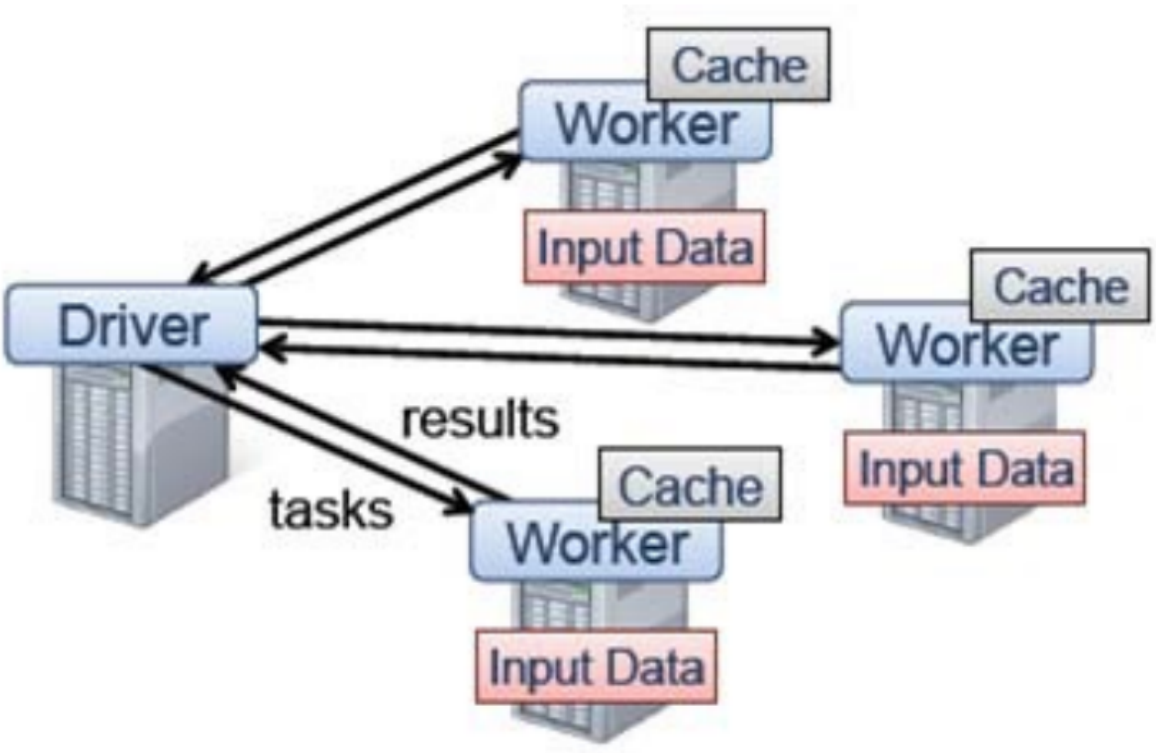
3）集群管理器：为每个应用程序选择集群管理器Spark完全取决于目标，因为集群管理器提供了不同的调度功能集。在开发新的Spark应用程序时，独立集群管理器更易于使用。Spark应用程序可以利用三种类型的集群管理器来分配和取消分配各种物理资源，例如客户端Spark作业，CPU内存等。





Spark架构采用了分布式计算中的Master-Slave模型，Master是对应集群中的含有Master进程的节点，Slave是集群中含有Worker进程的节点。Master作为整个集群的控制器，负责整个集群的正常运行；Worker相当于是计算节点，接收主节点命令与进行状态汇报；Executor负责任务的执行；Client作为用户的客户端负责提交应用，Driver负责控制一个应用的执行。

Spark集群部署后，需要在主节点和从节点分别启动Master进程和Worker进程，对整个集群进行控制。在一个Spark应用的执行过程中，Driver和Worker是两个重要角色。Driver程序是应用逻辑执行的起点，负责作业的调度，即Task任务的分发，而多个Worker用来管理计算节点和创建Executor并行处理任务。在执行阶段，Driver会将Task和Task所依赖的file和jar序列化后传递给对应的Worker机器，同时Executor对相应数据分区的任务进行处理。

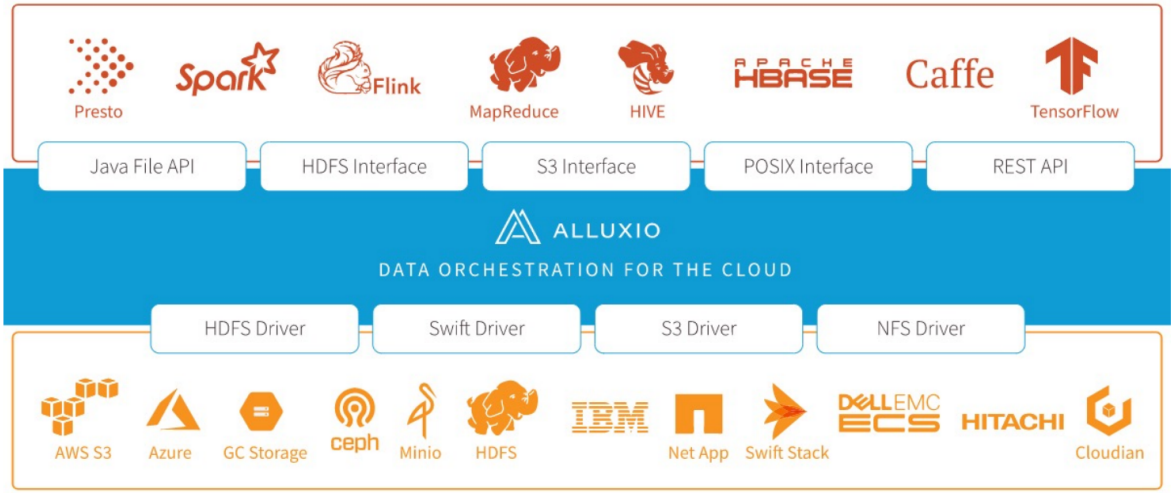


Spark的架构中的基本组件：

- Cluster Manager：在standalone模式中即为Master主节点，控制整个集群，监控worker。在YARN模式中为资源管理器
- Worker：从节点，负责控制计算节点，启动Executor或者Driver。在YARN模式中为NodeManager，负责计算节点的控制。
- Driver：运行Application的main()函数并创建SparkContext。
- Executor：执行器，在worker node上执行任务的组件、用于启动线程池运行任务。每个Application拥有独立的一组Executor。
- SparkContext：整个应用的上下文，控制应用的生命周期。
- RDD：Spark的基础计算单元，一组RDD可形成执行的有向无环图RDD Graph。
- DAG Scheduler：根据作业（task）构建基于Stage的DAG，并提交Stage给TaskScheduler。
- TaskScheduler：将任务（task）分发给Executor执行。
- SparkEnv：线程级别的上下文， 存储运行时的重要组件的引用。

3.简述何为“数据编排”以及Alluxio的特点

在Alluxio中，为了从根本上解决数据访问的挑战，数据世界需要全新一层，称之为“数据编排平台”，架构在计算框架和存储系统之间。数据编排平台跨存储系统将数据访问抽象出来，虚拟化所有数据，并通过具有全局命名空间的标准化API将数据呈现给数据驱动的应用程序。同时，它还应该具有缓存功能，以支持快速访问热数据。总之，数据编排平台为数据驱动的应用程序提供了数据可访问性、数据本地性和数据可伸缩性。



做一个类比，数据编排之于数据，就像容器编排之于容器一样。容器编排是一类技术，它使容器能够在任何环境中运行而不受正在运行的应用程序硬件的影响，并确保应用程序按预期运行。类似地，数据编排也是一种技术，它使应用程序的运行能够与计算无关、与存储无关和与云无关。

现在，基于数据编排平台，应用程序开发人员就可以假设数据随时可以访问，而不需要关注数据驻留在何处或存储的特性如何，并将重点放在编写应用程序上。

除了向应用程序开发人员授权外，数据编排平台还为基础设施工程师带来了巨大的价值。它通过在基础设施层为组织机构提供灵活性来避免被某一家供应商绑死。在不同的存储系统（包括云存储）之间进行转换、采用另一个应用程序框架，甚至采用一个混合或多云环境都是可行的，并且不会带来很大的开发成本。

Alluxio的特点：

Alluxio是一个以内存为中心的虚拟分布式存储系统，统一数据访问和桥梁的计算框架和底层存储系统。应用程序只需要alluxio就可以把访问存储在任何底层存储系统的数据连接。此外，Alluxio以内存为中心的架构实现数据访问的数量级的速度比现有的解决方案快很多。

在大数据的生态系统，Alluxio在于计算框架或jobs之间，如Apache的Spark，Apache的 MapReduce，或Apache Flink，和各种各样的存储系统，如Amazon S3，OpenStack Swift，GlusterFS， HDFS，Ceph，或OSS。Alluxio带来显着的改善生态系统的性能；例如，百度使用alluxio提高加速数据分 析管道30倍的吞吐量。

主要特性：数据存储与计算分离，两部分引擎可以进行独立的扩展。计算引擎（如Hadoop，Spark）可以访问不同数据源（Amazon S3, HDFS）中的数据。

特点：

①提供内存级I/O吞吐率，同时降低具有弹性扩张特性的数据驱动型应用的成本开销；

②简化云存储和对象存储接入；

③简化数据管理：Alluxio提供对多个数据源的单点访问。除了连接不同类型的数据源之外，Alluxio还使用户能够同时连接到同一存储系统的不同版本，例如多个版本的HDFS，而无需复杂的系统配置和管理；

④简单的应用程序部署：Alluxio管理应用程序与文件或对象存储之间的通信，将数据访问请求从应用程序转换为底层存储接口。Alluxio兼容Hadoop，Spark和MapReduce程序，可以无需修改任何代码在Alluxio之上运行。