

第六讲 批处理系统Spark



徐辰
cxu@dase.ecnu.edu.cn

华东师范大学



Spark简介

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



Michael Franklin



Michael Jordan



Sanjiv



Matei Zaharia



Spark简介

- 2013年Spark加入Apache孵化器项目后发展迅猛
- 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软件栈

Access and Interfaces	Spark Streaming	BlinkDB Spark SQL	GraphX	MLBase MLlib
Processing Engine	Spark Core			
Storage	Tachyon			
	HDFS, S3			
Resource Virtualization	Mesos		Hadoop Yarn	

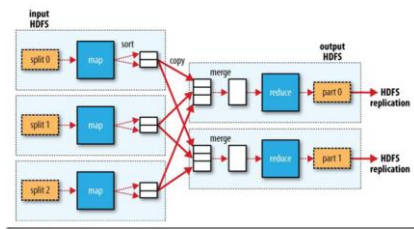
大纲

- 设计思想
 - MapReduce的局限性
 - 数据模型：RDD
 - 计算模型：DAG
- 体系架构
- 工作原理
- 容错机制

MapReduce

7

MapReduce implementation principles



MapReduce编程范型

8

- 编程容易，不需要掌握分布式并行编程细节，很容易把程序运行在分布式系统上
- 单个基本算子太少：**例如，如何join?**

```

class X {
    map() {           //map函数的实现
        ...
    }

    reduce() {        //reduce函数的实现
        ...
    }

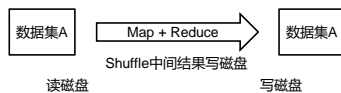
    main(){
        Job job = ... //定义分布式作业
        job.config = //作业参数设置
    }
}

```

单个MapReduce作业

9

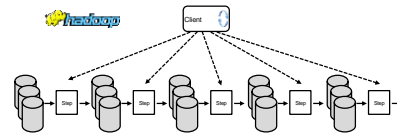
- Map的结果要先写入本地磁盘，再由Reduce端来拉取



多个MapReduce作业

10

- 例子：迭代计算过程中每一迭代步结束时讲结果写入HDFS，下一步将该结果再次从HDFS读出



MapReduce局限性

11

- 编程框架的表达能力有限，用户编程复杂
 - 仅map和reduce函数，无法直接用join等操作
- 单个作业Shuffle阶段的数据以阻塞方式传输，磁盘IO开销大、延迟高
 - 输入、输出及shuffle中间结果都需要读写磁盘
- 多个作业之间衔接涉及IO开销，应用程序的延迟高
 - 特别是迭代计算，中间结果的反复读写，使得整个应用的延迟非常高

大纲

12

- 设计思想
 - MapReduce的局限性
 - 数据模型：RDD
 - 计算模型：DAG
- 体系架构
- 工作原理
- 容错机制

RDD概念

13

- RDD: Resilient Distributed Dataset (弹性分布式数据集)
 - ✚ 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
- RDD是可恢复的记录分区的数据集合
 - ✚ Resilient: 具有可恢复的容错特性
 - ✚ Distributed: 每个RDD可分成多个分区, 一个RDD的不同分区可以存到集群中不同的节点上
 - ✚ Dataset: 每个分区就是一个数据集片段

大纲

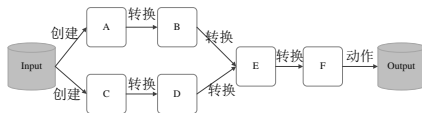
14

- 设计思想
 - ✚ MapReduce的局限性
 - ✚ 数据模型: RDD
 - ✚ 计算模型: DAG
- 体系架构
- 工作原理
- 容错机制

针对RDD的操作算子

15

- RDD操作支持数据运算, 分为
 - ✚ 转换Transformation: 描述RDD的转换逻辑
 - ✚ 动作Action: 标志转换结束



RDD运算操作

16

- 常用的API, 更详细需查阅官方文档: <http://spark.apache.org/docs/latest/rdd-programming-guide.html>

常用的几个Transformation API介绍

Transformation API	说明
filter(func)	筛选出满足函数func的元素, 并返回一个新的数据集
map(func)	将每个元素传递到函数func中, 并将结果返回为一个新的数据集
flatMap(func)	与map相似, 但每个输入元素都可以映射到零个或多个输出结果
groupByKey()	应用于(K,V)键值对的数据集时, 返回一个新的(Iterable<V>)形式的数据集
reduceByKey(func)	应用于(K,V)键值对的数据集时, 返回一个新的(K,V)形式的数据集, 其中的每个值是通过对每个key值调用函数func进行聚合

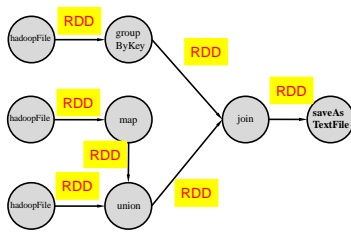
常用的几个Action API介绍

Action API	说明
count()	返回数据集的元素个数
collect()	以数组形式返回数据集的所有元素
first()	返回数据集的第一个元素
take(n)	以数组形式返回数据集的前n个元素
reduce(func)	通过函数func (输入两个参数并返回一个值) 聚合数据集的元素
foreach(func)	将数据集的每个元素传递到函数func中运行

逻辑计算模型: Operator DAG

17

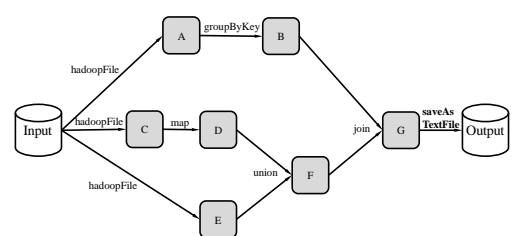
- 从算子操作的角度来描述计算的过程



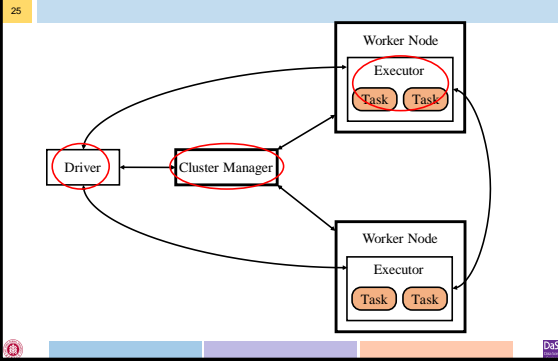
逻辑计算模型: RDD Lineage

18

- 从RDD变换的角度来描述计算过程



抽象架构图



Cluster Manager

- 集群管理器，负责管理整个系统的资源、监控工作节点
- 根据Spark部署方式的不同，
 - 在Standalone方式（即不使用Yarn或Mesos等其它资源管理系统）中，集群管理器包含Master和Worker
 - 在Yarn方式中集群管理器包括Resource Manager和Node Manager

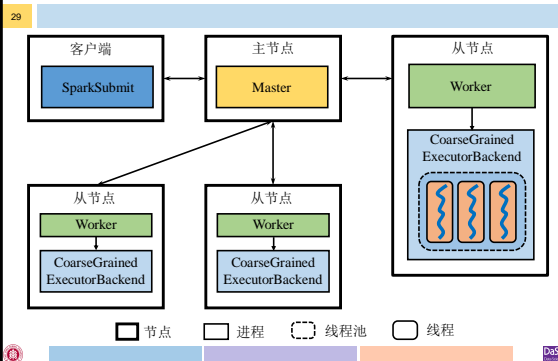
Executor

- 执行器，负责任务执行。Executor本身是运行在工作节点上的一个进程，它启动若干个线程Task或线程组TaskSet来进行执行任务
 - MapReduce中的Task是线程还是进程？
- 在Standalone部署方式下，Executor进程的名称为CoarseGrainedExecutorBackend

Driver

- 驱动器，负责启动应用程序的主函数并管理作业运行
- Driver中的SparkContext类维护了DAG、RDD lineage这些至关重要的信息

Standalone架构图



架构图比较

Standalone架构图	抽象架构图
SparkSubmit (Client)	/
Master	Cluster Manager
Worker	
CoarseGrained ExecutorBackend	Executor
物理节点 (从节点)	Worker Node
?	Driver

Standalone中的Driver

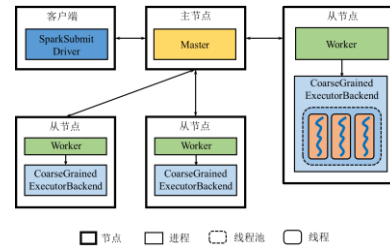
31

- 逻辑上，Driver独立于主节点、从节点以及客户端
- 但是根据应用程序的Client或Cluster部署方式，Driver会以不同的形式存在。
 - Client部署：Driver和客户端以同一个进程存在
 - Cluster部署：系统将由某一Worker启动一个进程作为Driver
- 客户端提交应用程序时可以选择Client或Cluster部署方式

Standalone Client

32

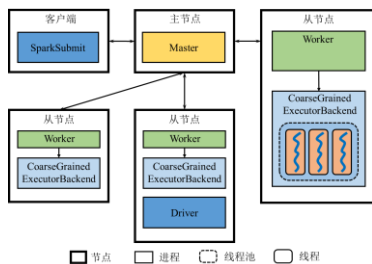
- Driver和客户端以同一个进程存在



Standalone Cluster

33

- 某一Worker启动一个进程作为Driver



Spark vs. MapReduce

34

- 二者架构比较

	MapReduce	Spark
系统进程	JobTracker	Master
	TaskTracker	Worker
工作线程	Child	CoarseGrained ExecutorBackend
任务代码	Task	TaskSet/Task

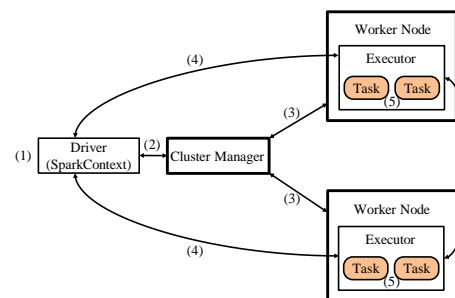
大纲

35

- 设计思想
- 体系架构
 - 架构图
 - 应用程序执行流程
- 工作原理
- 容错机制

抽象执行流程

36



应用程序执行流程

37

1. 启动Driver，以Standalone模式为例
 - 如果使用Client部署方式，客户端直接启动Driver，并向Master注册。
 - 如果使用Cluster部署方式，客户端将应用程序提交给Master，由Master选择一个Worker启动Driver进程(DriverWrapper)。
2. 构建基本运行环境，即由Driver创建SparkContext，向Master进行资源申请，并由Driver进行任务分配和监控。

应用程序执行流程（续）

38

3. Cluster Manager通知工作节点启动Executor进程，该进程内部以多线程方式运行任务
4. Executor进程向Driver注册
5. SparkContext构建DAG并进行任务划分，从而交给Executor进程中的线程来执行任务。

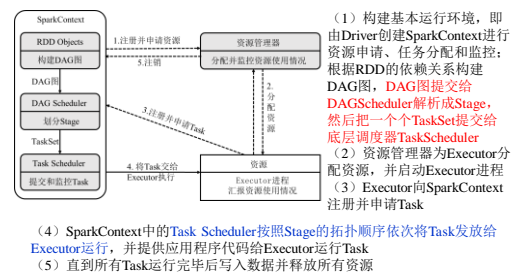
大纲

39

- 设计思想
- 体系架构
- 工作原理
- 容错机制

Spark工作过程

40

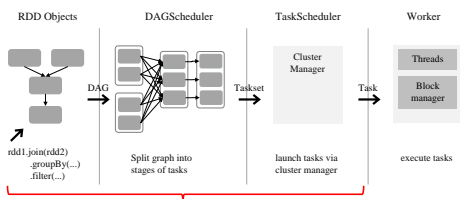


SparkContext工作过程

41

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。

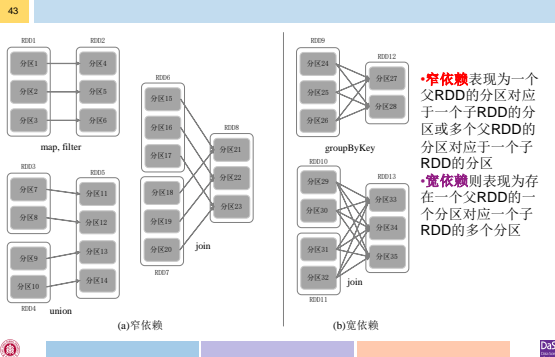


大纲

42

- 设计思想
- 体系架构
- 工作原理
 - ✦ DAG Stage划分
 - ✦ Stage内部数据交换
 - ✦ Stage之间数据交换
 - ✦ 应用、作业与任务
- 容错机制

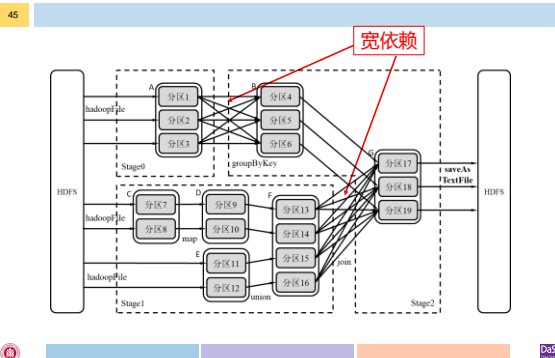
RDD依赖关系



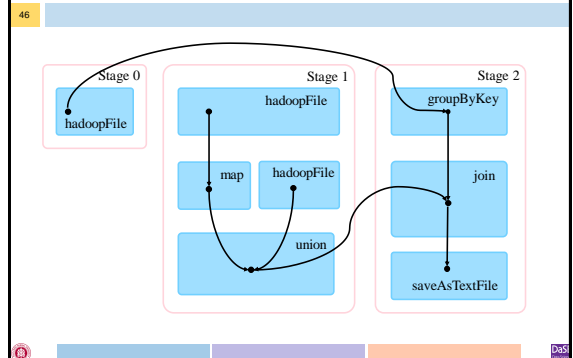
为什么关心依赖关系？

- 44
- 分析各个RDD的偏序关系生成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：
 - 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
 - 为什么将窄依赖尽可能划分在同一个Stage？

基于RDD Lineage的Stage划分



基于Operator DAG的Stage划分



Stage类型

ShuffleMapStage

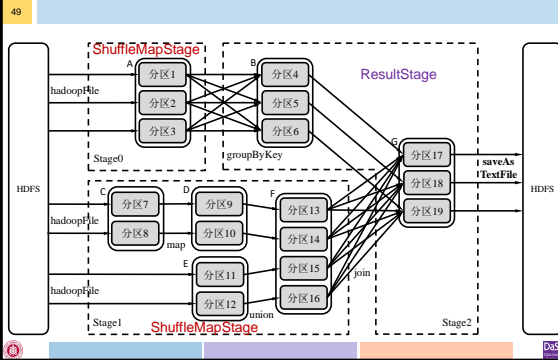
- 47
- 输入/输出
 - 输入可以从外部获取数据，也可以是另一个ShuffleMapStage的输出
 - 以Shuffle为输出，作为另一个Stage开始
 - 特点
 - 不是最终的Stage，在它之后还有其他Stage
 - 它的输出一定需要经过Shuffle过程，并作为后续Stage的输入
 - 在一个DAG里可能有该类型的Stage，也可能没有该类型Stage

Stage类型

ResultStage

- 48
- 输入/输出
 - 其输入可以从外部获取数据，也可以是另一个ShuffleMapStage的输出
 - 输出直接产生结果或存储
 - 特点
 - 最终的Stage
 - 在一个DAG里必定有该类型Stage
 - 因此，一个DAG含有一个或多个Stage，其中至少含有一个ResultStage

ShuffleMapStage vs. ResultStage

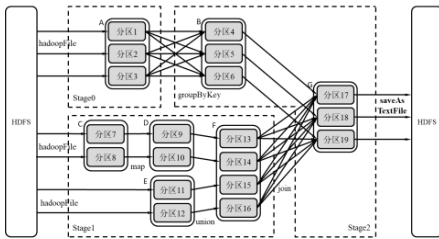


大纲

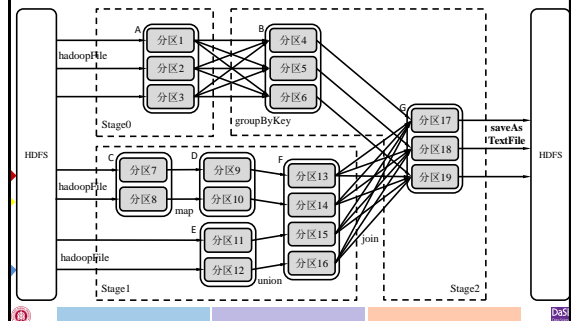
- 设计思想
- 体系架构
- 工作原理
 - ✦ DAG Stage划分
 - ✦ Stage内部数据交换
 - ✦ Stage之间数据交换
 - ✦ 应用、作业与任务
- 容错机制

Stage内部的特点

- 所有依赖关系都是**窄依赖**，可以实现 pipeline方式进行数据传输



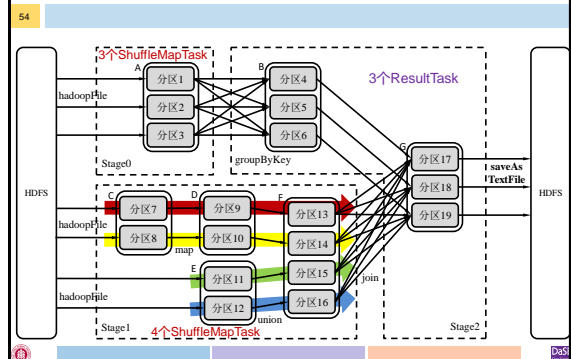
流水线（Pipeline）方式



Spark Pipeline vs. MapReduce Shuffle

- 与MapReduce中**Shuffle方式**不同，流水线方式不要求物化前序算子的所有计算结果
 - ✦ 分区7通过map操作生成的分区9，并不需要物化分区9，而且可以不用等待分区8到分区10这个map操作的计算结束，继续进行union操作，得到分区13
 - ✦ 如果采用MapReduce中的Shuffle方式，那么意味着分区7、8经map计算得到分区9、10并将这两个分区进行物化之后，才可以进行union

ShuffleMapTask vs. ResultTask

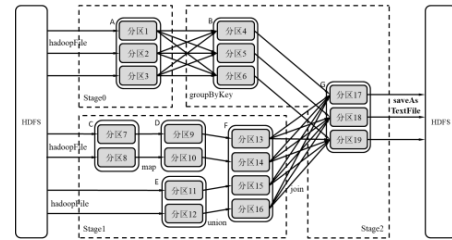


大纲

- 设计思想
- 体系架构
- 工作原理
 - ✚ DAG Stage划分
 - ✚ Stage内部数据交换
 - ✚ Stage之间数据交换
 - ✚ 应用、作业与任务
- 容错机制

Stage之间的特点

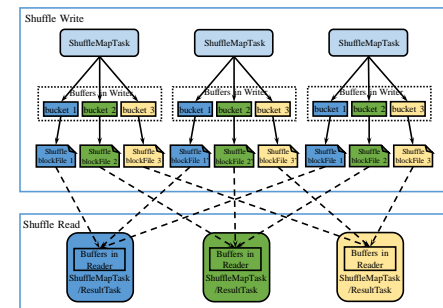
- 所有依赖关系都是**宽依赖**，不可以实现 pipeline方式进行数据传输，只能Shuffle



Stage之间的Shuffle

- Stage之间的数据交换需要进行Shuffle，该过程与MapReduce中的Shuffle类似
- Shuffle过程可能发生在**两个 ShuffleMapStage之间**，或者**ShuffleMapStage与ResultStage之间**
- 从Task的层面来看，该过程表现为**两组 ShuffleMapTask之间**，或**一组 ShuffleMapTask与一组ResultTask之间**的数据交换

Spark Shuffle

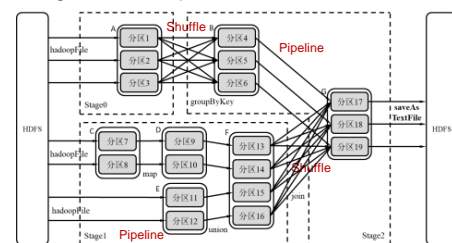


Shuffle Write vs. Shuffle Read

- 在Shuffle Write阶段，ShuffleMapTask需要将输出RDD的记录按照partition函数划分到相应的bucket当中并物化到本地磁盘形成ShuffleblockFile，**之后才可以在 Shuffle Read阶段被拉取**
- 在Shuffle Read阶段，ShuffleMapTask或ResultTask根据partition函数读取相应的ShuffleblockFile，存入buffer并进行继续后续的计算

Shuffle vs. Pipeline

- Stage之间：Shuffle
- Stage内部：Pipeline



大纲

61

- 设计思想
- 体系架构
- 工作原理
 - ✚ DAG Stage划分
 - ✚ Stage内部数据交换
 - ✚ Stage之间数据交换
 - ✚ 应用、作业与任务
- 容错机制

应用、作业与任务

62

- Application: 用户编写的Spark应用程序
- Job: 一个Job包含多个RDD及作用于相应RDD转换操作, 其中最后一个为action
- Stage: 一个Job会分为多组Task, 每组Task被称为Stage, 或者也被称为TaskSet
 - ✚ Job的基本调度单位
 - ✚ 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
- Task: 运行在Executor上的工作单元

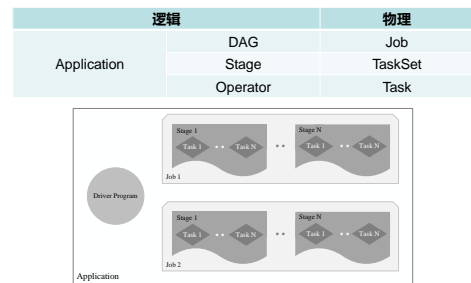
应用、作业与任务

63

- 逻辑执行角度
 - ✚ 一个Application=一个或多个DAG
 - ✚ 一个DAG=一个或多个Stage
 - ✚ 一个Stage=若干窄依赖的RDD操作
- 物理执行角度
 - ✚ 一个Application=一个或多个Job
 - ✚ 一个Job=一个或多个TaskSet
 - ✚ 一个TaskSet=多个没有Shuffle关系的Task

逻辑概念与物理概念

64



Spark vs. MapReduce

65

- 应用与作业

名称	MapReduce	Spark
应用		Application
作业	Job	Job/DAG

大纲

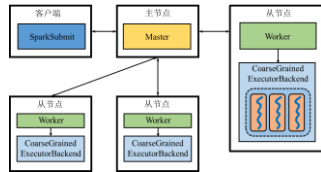
66

- 设计思想
- 体系架构
- 工作原理
- 容错机制

故障类型

67

- Master故障: ZooKeeper配置多个Master
- Worker故障
- Executor故障
- Driver故障: 重启



大纲

68

- 设计思想
- 体系架构
- 工作原理
- 容错机制
 - ✚ RDD持久化
 - ✚ 故障恢复
 - ✚ 检查点

RDD存储机制

69

- RDD提供的持久化（缓存）接口
 - ✚ persist(): 对一个RDD标记为持久化
 - 接受StorageLevel类型参数, 可配置各种级别
 - 持久化后的RDD将会被保留在工作节点的中被后面的行动操作重复使用
 - ✚ cache()
 - 相当于persist(MEMORY_ONLY)
 - ✚ 可以使用unpersist()方法手动地把持久化的RDD从缓存中移除
- 问题: 没有标记持久化的RDD会怎样?

持久化举例

70

```
scala> val list = List("Hadoop", "Spark", "Hive")
list: List[String] = List(Hadoop, Spark, Hive)
scala> val rdd = sc.parallelize(list)
rdd: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[22] at
parallelize at <console>:29
scala> rdd.cache() //会调用persist(MEMORY_ONLY), 但是, 语句执行到这里,
并不会缓存rdd, 因为这时rdd还没有被计算生成
scala> println(rdd.count()) //第一次行动操作, 触发一次真正从头到尾的计算,
这时上面的rdd.cache()才会被执行, 把这个rdd放到缓存中
3
scala> println(rdd.collect().mkString(",")) //第二次行动操作, 不需要触发从头
到尾的计算, 只需要重复使用上面缓存中的rdd
Hadoop,Spark,Hive
```

RDD存储机制

71

- Storage level
 - ✚ MEMORY_ONLY
 - ✚ MEMORY_AND_DISK
 - ✚ MEMORY_ONLY_SER
 - ✚ MEMORY_AND_DISK_SER
 - ✚ DISK_ONLY
 - ✚ MEMORY_ONLY_2,
 - ✚ MEMORY_AND_DISK_2
 - ✚ OFF_HEAP (experimental)

RDD存储机制（续）

72

- persist(*)
 - ✚ MEMORY_ONLY: 在JVM中缓存存储Java的对象。如果内存不足, 直接丢弃某些partition
 - ✚ MEMORY_AND_DISK: 在JVM中缓存Java的对象。如果内存不足, 则将某些partitions写入到磁盘中
 - ✚ MEMORY_ONLY_SER: 在内存为每个partition存储一个byte数组, 数组内容为当前partition中Java对象的序列化结果
 - ✚ MEMORY_AND_DISK_SER: 与MEMORY_AND_DISK类似, 但是每个partition存储的是Java对象序列化后组成的byte数组。

RDD的存储机制（续）

73

persist(*)

- ✚ DISK_ONLY: 将每个partition的数据序列化到磁盘中
- ✚ MEMORY_ONLY_2, MEMORY_AND_DISK_2: 与名字中类型对应，但是每个partition备份到两台机器上
- ✚ OFF_HEAP:
 - RDD的数据被序列化后存储在Tachyon中
 - Tachyon决定内存块的替换
 - 与MEMORY_ONLY_SER对比
 - 数据与executor的分离，避免某个executor失效导致当前节点上数据的丢失
 - executor之间能够共享内存
 - 降低垃圾回收的代价，减少executor的运行空间

大纲

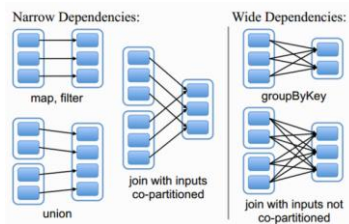
74

- 设计思想
- 体系架构
- 工作原理
- 容错机制
 - ✚ RDD持久化
 - ✚ 故障恢复
 - ✚ 检查点

Lineage机制

75

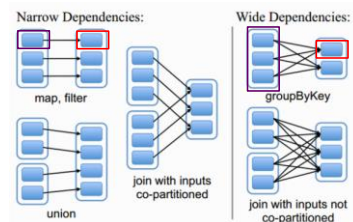
□ RDD dependencies



Lineage机制

76

□ 红色部分丢失，需重算紫色部分



Lineage机制

77

□ 窄依赖(narrow dependency)

- ✚ 执行某个partition时，检查父亲RDD对应的partition是否存在
 - 存在，即可执行当前RDD对应的操作
 - 不存在，则重构父亲RDD对应的partition

□ 宽依赖(wide dependency)

- ✚ 执行某个partition时，检查父亲RDD对应的partition是否存在
 - 存在，即可执行当前RDD对应的操作
 - 不存在，则重构整个父亲RDD

基于RDD Lineage恢复

78

- 利用RDD Lineage的故障恢复
 - ✚ 重新计算丢失分区
 - ✚ 重算过程在不同节点之间可以并行
- 与数据库恢复的比较
 - ✚ RDD Lineage: 记录粗粒度的操作
 - ✚ 数据复制或者日志: 记录细粒度的操作

大纲

79

- 设计思想
- 体系架构
- 工作原理
- 容错机制
 - ✦ RDD持久化
 - ✦ 故障恢复
 - ✦ 检查点

检查点机制

80

- 前述机制的不足之处
 - ✦ Lineage可能**非常长**
 - ✦ RDD持久化机制保存到集群内机器的磁盘，**并不完全可靠**
- 检查点机制将RDD写入**外部可靠的**（本身具有容错机制）分布式文件系统，例如HDFS
 - ✦ 在实现层面，写检查点的过程是一个独立job，在用户作业结束后运行

课后阅读

81

- 论文
 - ✦ Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark : Cluster Computing with Working Sets. In HotCloud (pp. 1–7).
 - ✦ Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mccauley, M., ... Stoica, I. (2012). Resilient Distributed Datasets : A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In NSDI (pp. 15–28).

本讲小节

82

- 设计思想
- 体系架构
- 工作原理
- 容错机制

谢谢！ Q&A

Spark

