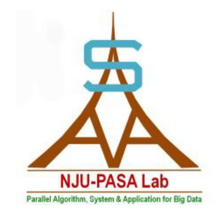


大数据内存计算系统Spark的基本原理、构架与编程技术

南京大学 PASA大数据实验室
江苏鸿程大数据技术与应用研究院

Spark系统及其编程技术

- Spark系统简介
- Spark编程示例
- Spark生态系统中其它功能组件简介
- Spark 2.x新特性介绍
- Spark 安装配置速览

Spark 系统简介

1.Scala编程语言简介

2.为什么会有Spark?

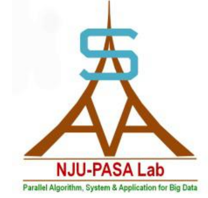
3.Spark的基本构架和组件

4.Spark的程序执行过程

5.Spark的技术特点

6.Spark编程模型与编程接口

1.Scala编程语言简介



1. Scala是一种搭建于JVM之上的语言，设计初衷是实现可伸缩的语言、并集成面向对象编程和函数式编程的各种特性。

2. 语言:指定所有变量的类型

```
scala> val a = 1
a: Int = 1

scala> val b = 1.0
b: Double = 1.0
```

```
scala> val a: Int = 1
a: Int = 1

scala> val b: Double = 1
b: Double = 1.0
```

```
scala> val c = 1L
c: Long = 1

scala> val d = 1.0f
d: Float = 1.0
```

```
scala> val c: Long = 1
c: Long = 1

scala> val d: Float = 1
d: Float = 1.0
```

3. val, var 表示变量是不变引用，还是可变引用。优先使用不变引用。scala.collection.mutable.HashMap里的HashMap是不变引用，但是可以更新集合内的数值。

1.Scala编程语言简介

3. 基于JVM：Scala 可编译为 Java 字节码，在 JVM 上运行，可以与java互操作。因此可以直接利用丰富的java开源项目。

4. 面向对象：每个值都是对象。对象的类型和行为用class和trait进行描述。支持继承。

6. 函数式编程：每个函数都是值。函数可以独立存在，可以定义一个函数为另一个函数的返回值，可以接受函数作为另一个函数的参数。

1.Scala编程语言简介

类与对象

class定义一个类，括号里是构造参数

```scala
class Greeter(prefix: String, suffix: String) {
  def greet(name: String): Unit =
    println(prefix + name + suffix)
}
// 创建类实例
val greeter = new Greeter("Hello, ", "!")
greeter.greet("Scala developer") // Hello, Scala developer!
```

1.Scala编程语言简介

类与对象

object对象，同名类的单例对象或者自定义的单实例

```
object IdFactory {
  private var counter = 0
  def create(): Int = {
    counter += 1
    counter
  }
}
//引用名字来访问对象及其方法
val newId: Int = IdFactory.create()
println(newId) // 1
val newerId: Int = IdFactory.create()
println(newerId) // 2
```

1.Scala编程语言简介

方法与函数

方法：由def定义，方法名、参数列表、返回类型、方法体

```
scala> def addOne(x: Int): Int = x + 1
scala> println(addOne(1))
2
```

函数: 带有参数的表达式

```
scala> val addOne = (x: Int) => x + 1
scala> println(addOne(1))
2
```

Lamda表达式: (x: Int) => x + 1, 返回是一个函数对象

(x: Int) 参数名列表

x + 1 函数体

1.Scala编程语言简介

函数式编程

map对List中的每个元素应用一个函数，返回应用后的元素所组成的List

```
scala> val numbers = List(1, 2, 3, 4)
numbers: List[Int] = List(1, 2, 3, 4)

scala> val doubleSalary = (x: Int) => x * 2
//将函数作为map方法的参数
scala> numbers.map(doubleSalary)
res0: List[Int] = List(2, 4, 6, 8)
```

1.Scala编程语言简介

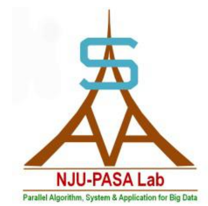
函数式编程—匿名函数

匿名函数：x => x *2

```
scala> val numbers = List(1, 2, 3, 4)
numbers: List[Int] = List(1, 2, 3, 4)
//非匿名函数
scala> val doubleSalary = (x: Int) => x * 2
scala> numbers.map(doubleSalary)
res0: List[Int] = List(2, 4, 6, 8)
//匿名函数
scala> numbers.map(x => x * 2)
res0: List[Int] = List(2, 4, 6, 8)
```

```
scala> numbers.map(_ * 2)
res0: List[Int] = List(2, 4, 6, 8)
```

Spark 系统简介



1.Scala编程语言简介

2.为什么会有Spark?

3.Spark的基本构架和组件

4.Spark的程序执行过程

5.Spark的技术特点

6.Spark编程模型与编程接口

2.为什么会有Spark?

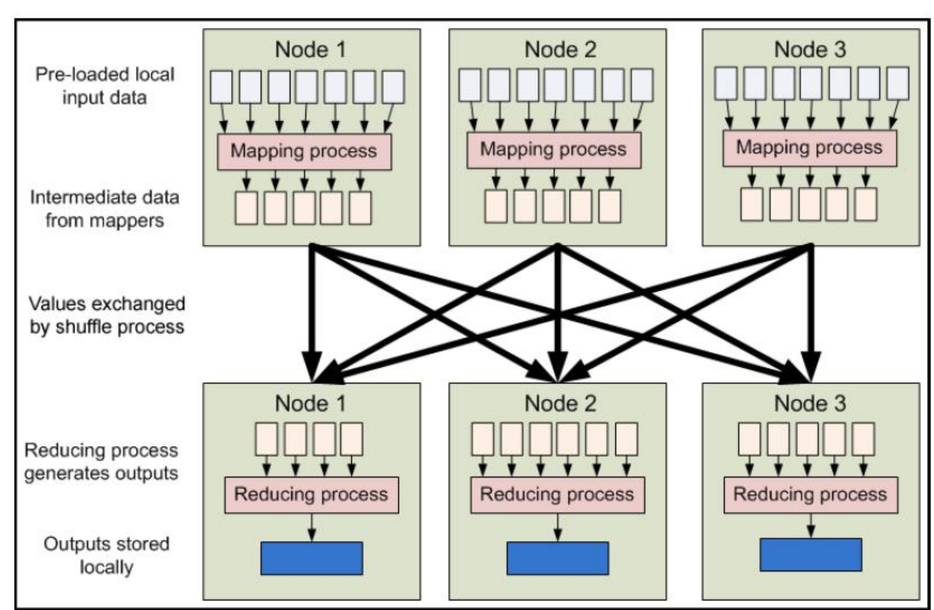
MapReduce计算模式的缺陷

- 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

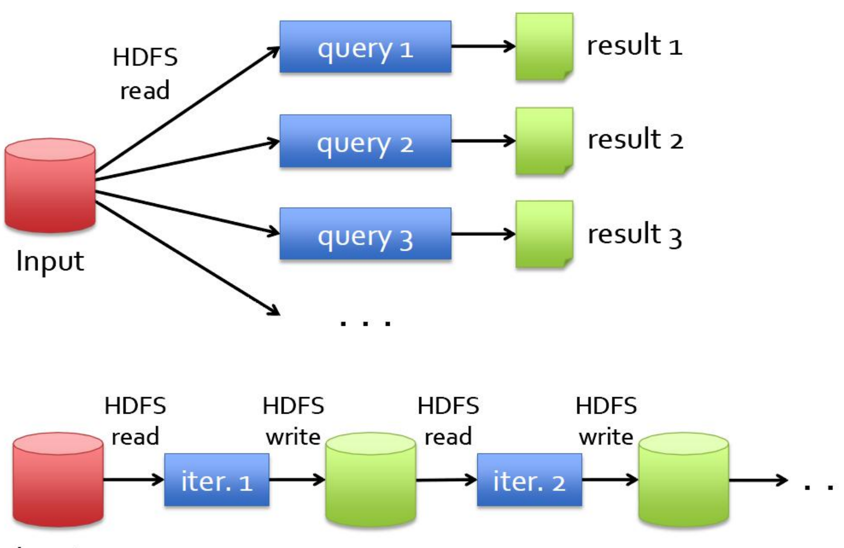
2.为什么会有Spark?

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.



2.为什么会有Spark?

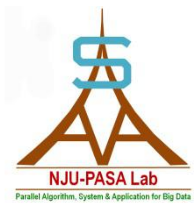
2013年大数据计算模式的新变化

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

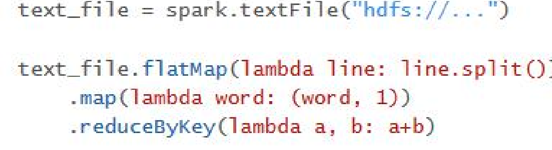
2.为什么会有Spark?



- 速度（Speed）
  - Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Logistic regression in Hadoop and Spark

- 易用性（Ease of Use）
  - Write applications quickly in Java, Scala, Python, R.

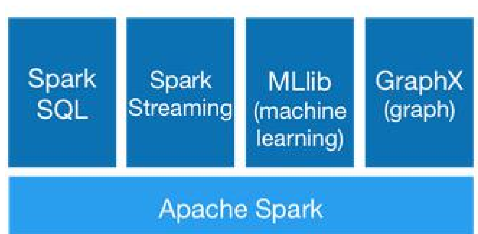


```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Word count in Spark's Python API

- 广泛性（Generality）
  - Combine SQL, streaming, and complex analytics.

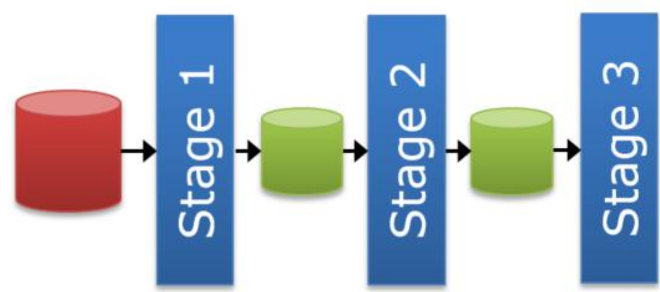


- 多处运行（Runs Everywhere）
  - Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

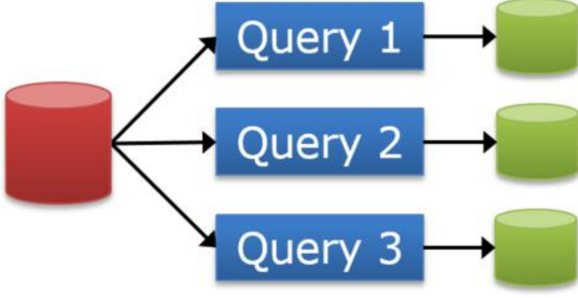
2.为什么会有Spark?

- 很多复杂的作业和交互式分析的查询都需要MapReduce不具备的一个功能：

Efficient primitives for **data sharing**



Iterative algorithm



Interactive data mining

2.为什么会有Spark?

- 很多复杂的作业和交互式分析的查询都需要MapReduce不具备的一个功能：

Efficient primitives for **data sharing**

Query 1

在MapReduce中，跨作业数据分享的方式是通过磁盘文件 (e.g. HDFS) -> **slow!**

Iterative algorithm

Interactive data mining

2.为什么会有Spark?

HDFS read

HDFS write

HDFS read

HDFS write

Input → iter. 1 → iter. 2 → . . .

Input

HDFS read

query 1 → result 1

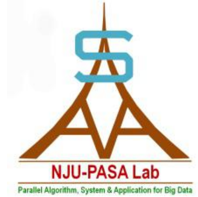
query 2 → result 2

query 3 → result 3

. . .

I/O and serialization can take **90%** of the time

2.为什么会有Spark?



目标：In-Memory Data Sharing

Input → iter. 1 → iter. 2 → . . .

Input → one-time processing → Distributed memory → query 1, query 2, query 3, . . .

10-100× faster than network and disk

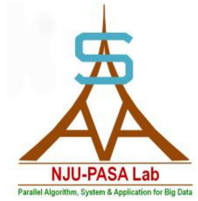
2.为什么会有Spark?

- 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。

2.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，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。

2.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

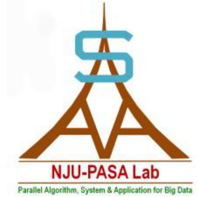
A: B: G: Stage 1 groupBy Task C: D: F: map E: join Stage 2 union Stage 3

= RDD = previously computed partition

one-time processing Input Distributed memory query 1 query 2 query 3 . . .

Input iter. 1 iter. 2 . . .

2.为什么会有Spark?



//在一个存储于HDFS的Log文件中，计算出现ERROR的行数

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split('\t')[2])
messages.cache()

messages.filter(lambda s: "foo" in s).count()
messages.filter(lambda s: "bar" in s).count()
. . .
```

Base RDD

Transformed RDD

Action

Driver

tasks

results

Worker

Cache 1

Block 1

Worker

Cache 2

Block 2

Worker

Cache 3

Block 3

Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

2.为什么会有Spark?

性能随内存使用变化情况

Execution time (s)

100
80
60
40
20
0

69
58
41
30
12

Cache disabled
25%
50%
75%
Fully cached

% of working set in cache

2.为什么会有Spark?

一张图说明一切!

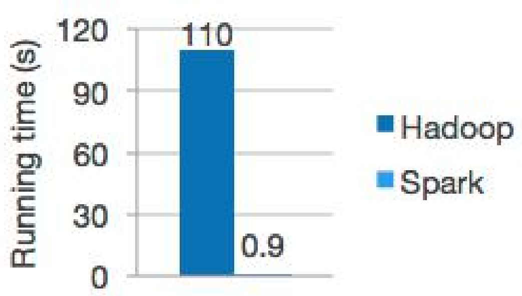
https://spark.apache.org

Apache Spark™ is a fast and general engine for large-scale data processing.

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



Logistic regression in Hadoop and Spark

Spark 系统简介

1.Scala编程语言简介

2.为什么会有Spark?

3.Spark的基本构架和组件

4.Spark的程序执行过程

5.Spark的技术特点

6.Spark编程模型与编程接口

3.Spark的基本构架和组件

主要体系结构和组件

Spark SQL | Spark Streaming | MLlib | GraphX

Spark

MapReduce | RDD | FP

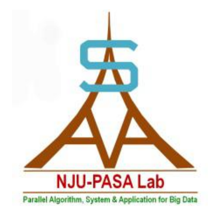
本地运行模式 | 独立运行模式 | Mesos | YARN

HDFS Hypertable，Hbase等

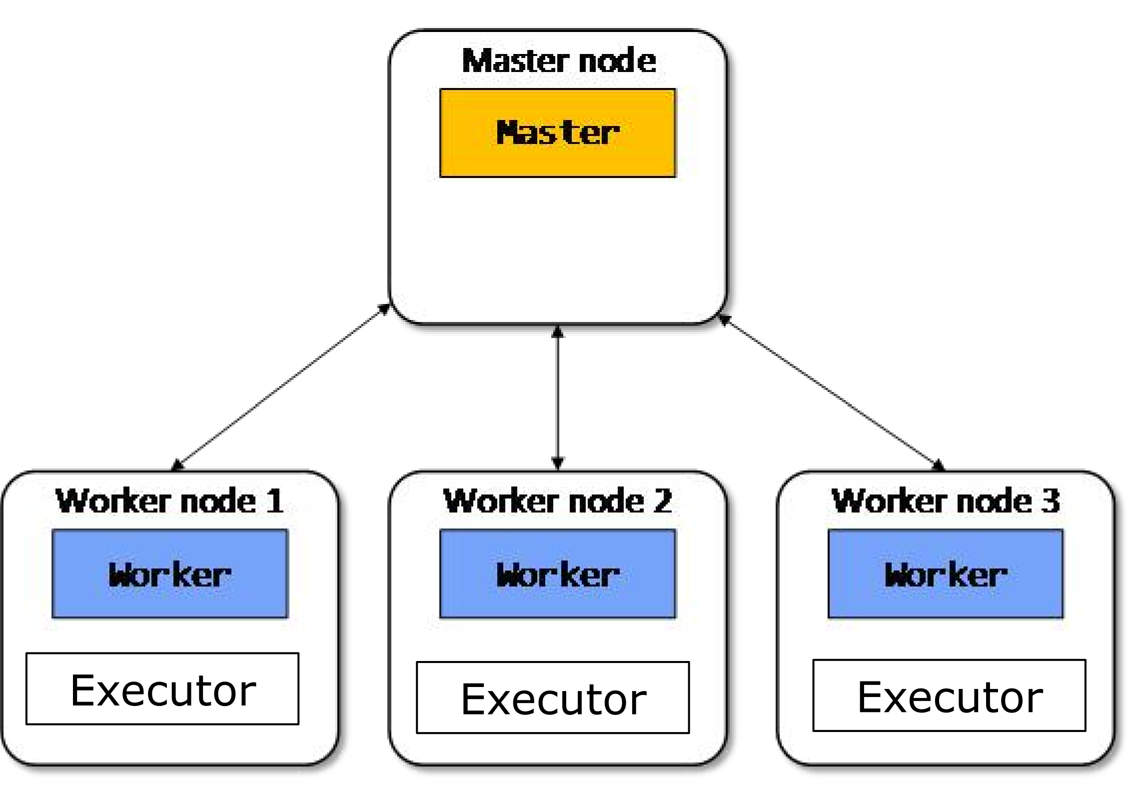
- Spark SQL、Spark Streaming、Mllib、GraphX是Spark提供的一系列高层工具，它们将在后面章节被详细介绍。同级的还有Bagel、shark等工具。
- FP：即函数式编程（function programing ）
- Mesos、YARN：即apache Mesos和YARN（Hadoop NextGen）两套资源管理框架，可以作为Spark的运行模式。同级的还有Amazon EC2等。

3.Spark的基本构架和组件

Spark集群的基本结构

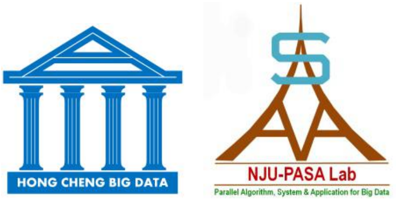


- Master node：是集群部署时的概念，是整个集群的控制器，负责整个集群的正常运行，管理Worker node。
- Worker node：是计算节点，接收主节点命令与进行状态汇报
- Executors：每个Worker上有一个Executor，负责完成Task程序的执行
- Spark集群部署后，需要在主从节点启动Master进程和Worker进程，对整个集群进行控制

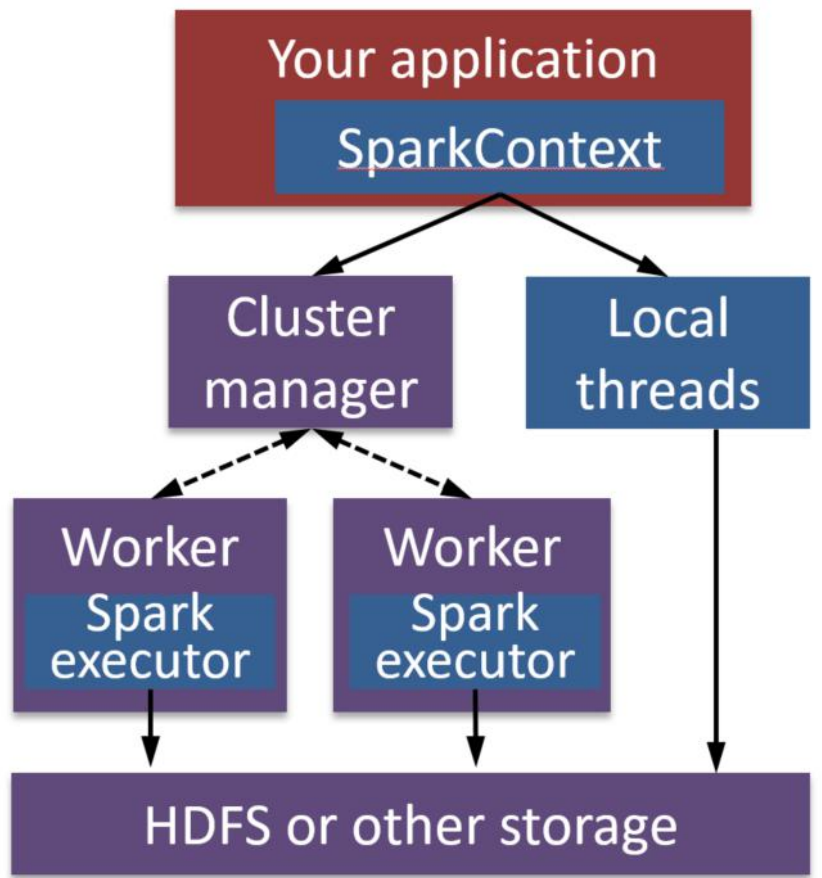


3.Spark的基本构架和组件

Spark集群的基本结构



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

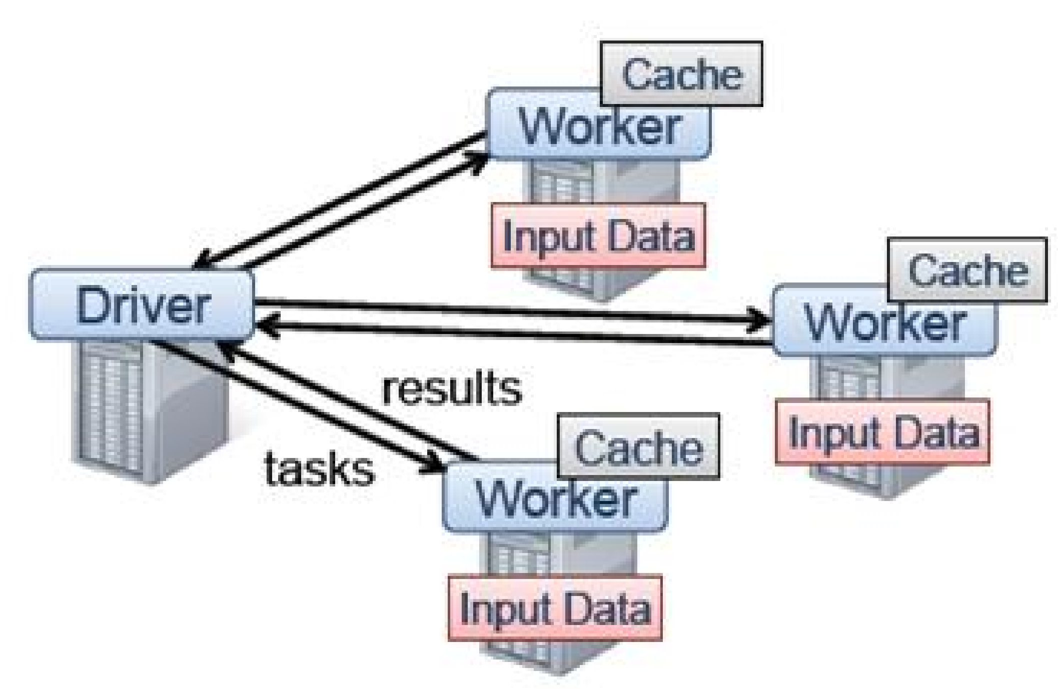


3.Spark的基本构架和组件简介

Spark系统的基本结构

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



3.Spark的基本构架和组件

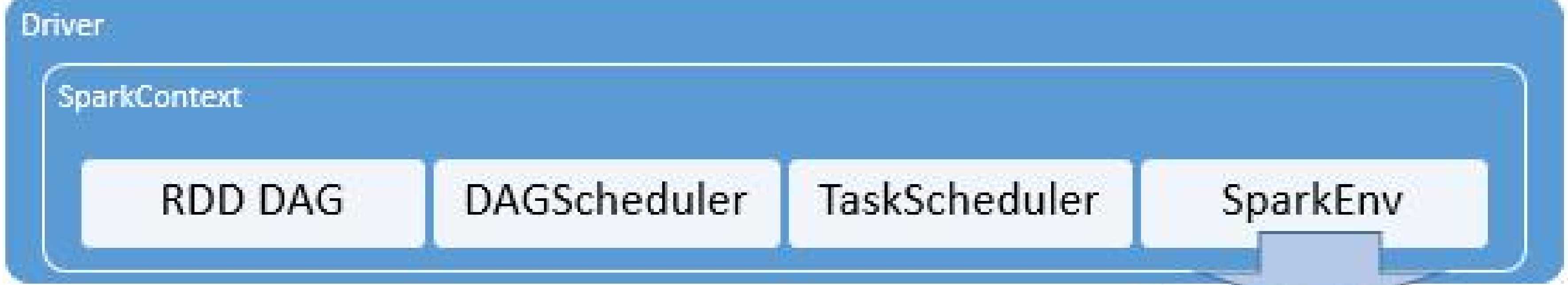
Spark应用程序的基本结构

- Application：基于Spark的用户程序，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

Application
Driver Program
Task1
Stage1
TaskN
JOB1
Task1
StageN
TaskN
Task1
Stage1
TaskN
JOBN
Task1
StageN
TaskN

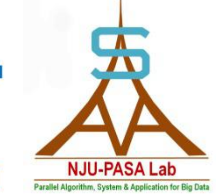
3.Spark的基本构架和组件

Spark Driver的组成

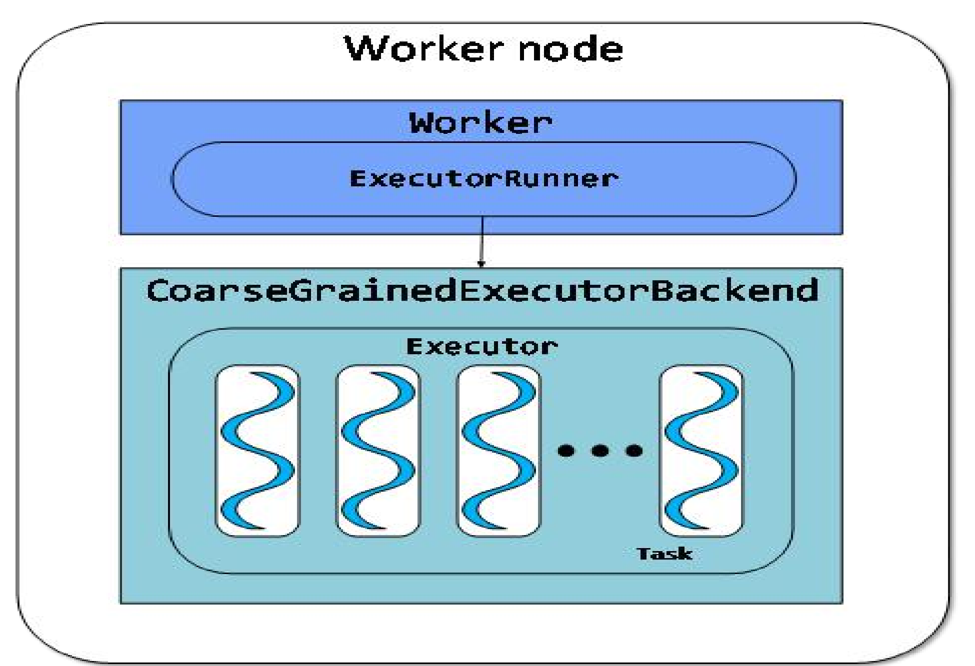


- Driver：执行Application的main()函数并创建SparkContext。
- RDD：Spark基本计算单元，一组RDD形成可执行的有向无环图（操作主要有：Transformation和Action）
- DAG Scheduler：根据Job构建的基于Stage的DAG，并提交Stage给TaskScheduler
- TaskScheduler：将Task分发给Executor执行
- SparkEnv：线程级别运行环境，存储运行时的重要组件的引用，创建并包含了：
  - MapOutPutTracker：存储Shuffle元信息
  - BroadcastManager：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

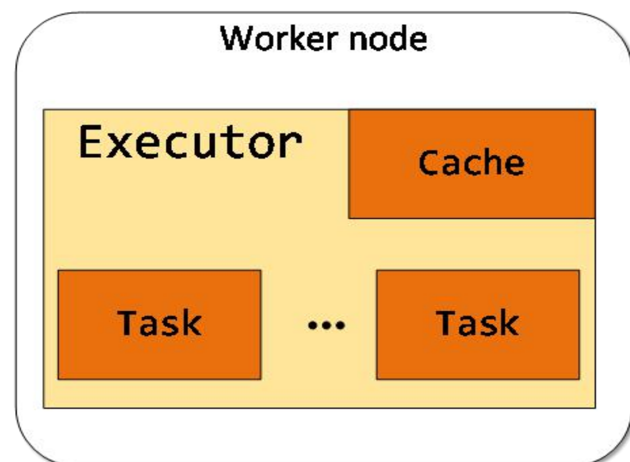
3.Spark的基本构架和组件



Worker node的结构



另一种图示：



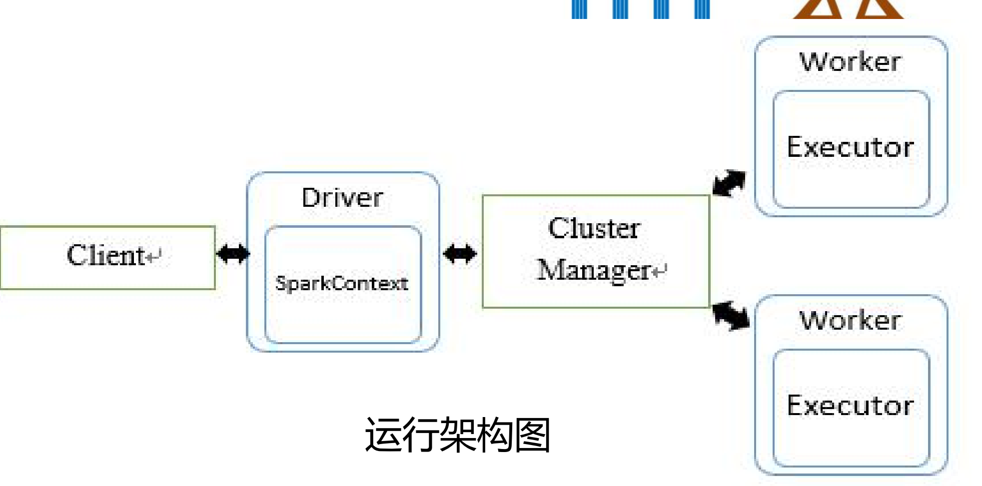
此处Cache部分即为Spark编程模型中对RDD进行内存持久化存储的部位。

- 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。

3.Spark的基本构架和组件

Spark程序运行机制

Client：用户的客户端
Driver：负责控制一个应用的执行
Cluster Manager：在集群上获取资源的外部服务，例如Standalone、Mesos、YARN。
Executor：负责执行Task任务



运行架构图

- Client 提交应用，Master节点启动Driver
- Driver向Cluster Manager申请资源，并构建Application的运行环境，即启动SparkContext
- SparkContext向ClusterManager申请Executor资源，并启动CoarseGrainedExecutorBackend
- Executor向SparkContext申请Task，SparkContext将代码发放给Executor
- Standalone模式下，ClusterManager即为Master。在YARN下，ClusterManager为资源管理器
- Driver Program可以在Master上运行，此时Driver就在Master节点上。如果是YARN集群，那么Driver可能被调度到Worker node上运行。为了防止Driver和Executor间通信过慢，一般原则上要使它们分布在同一个局域网中

Spark 系统简介

1.Scala编程语言简介

2.为什么会有Spark?

3.Spark的基本构架和组件

4.Spark的程序执行过程

5.Spark的技术特点

6.Spark编程模型与编程接口

4.Spark的程序执行过程

Spark应用程序的组成结构

Application

Job 0

Stage 0
Task 0
Task 1
Task 2
Task 3
...

Stage 1
Task 4
Task 5
Task 6
Task 7
...

Stage 2
Task 8
Task 9
Task 10
Task 11
...

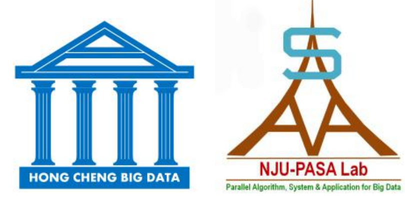
Job 1

Stage 0
Stage 1
Stage 2

Job 2

Stage 0
Stage 1
Stage 2

4.Spark的程序执行过程



Spark程序运行结构图

Driver Program

SparkContext

Cluster Manager

Worker Node

Executor

Cache

Task

Task

Worker Node

Executor

Cache

Task

Task

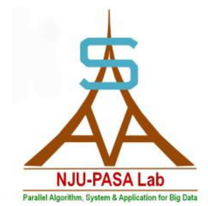
4.Spark的程序执行过程

Spark运行框架主节点

- Application：由用户编写的Spark应用程序，其中包括driver program和executor
- Driver Program：执行用户代码的main()函数，并创建SparkContext
- Cluster manager：集群当中的资源调度服务选取。

  例：standalone manager, Mesos, YARN
- Job：由某个RDD的Action算子生成或者提交的一个或者多个一系列的调度阶段，称之为一个或者多个Job，类似于MapReduce中Job的概念

4.Spark的程序执行过程



Spark运行框架主节点

- 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。

4.Spark的程序执行过程

Spark运行框架的从节点

- Executor：executor负责在子节点上执行Spark任务，每个application都有自身的Executor
- Stage：每一个Job被分成一系列的任务的集合，这些集合被称之为Stage，用于Spark阶段的调度

  例：在MapReduce作业中，Spark将划分为Map的Stage和Reduce的Stage进行调度
- Task：被分发到一个Executor上的最小处理单元

4.Spark的程序执行过程

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(Mesos/YARN),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。

4.Spark的程序执行过程

从RDD的转换和存储角度看这个过程

ERROR:error1
INFO:info2
ERROR:error2
ERROR:error3
...

textFile()

RDD 0
Partiton 0
Partiton 1
...
Partiton n

filter()

RDD 1
Partiton 0
Partiton 1
...
Partiton n

count()

Result RDD
Result partiton
...

Put
Get
Put
Get
Put

Block Manager

Block(Array)
ERROR:error1
INFO:info2
...

Block(Array)
ERROR:error4
ERROR:error5
ERROR:error6

Block(Array)
ERROR:error1
ERROR:error2
...

Block(Array)
ERROR:error6
ERROR:error7

4.Spark的程序执行过程

一个作业就是一张RDD世系图（DAG图）

Stage n

Stage 3

Stage 2

Page Ranks Stage 0

Join

Contribs per page

Reduce ByKey

Page ranks

Flatmap

Join

Stage 1

Page Links

Stage 4

Page Links

4.Spark的程序执行过程

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。

4.Spark的程序执行过程

几个基本概念

- 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

4.Spark的程序执行过程

几个基本概念

http://<Standalone Master>:8080 (by default)

Spark Master at spark://m
localhost:8080

Spark Master at spa

URL: spark://mbp-2.local:7077
Workers: 3
Cores: 24 Total, 24 Used
Memory: 45.0 GB Total, 1536.0 MB Used
Applications: Running, 0 Completed

Workers

Id
worker-20131202231645-192.168.1.106-56789
worker-20131202231657-192.168.1.106-56801
worker-20131202231705-192.168.1.106-56806

Running Applications

ID	Name
app-20131202231712-0000	Spark shell

Spark shell – Spark Stages
localhost:4040/stages/

Stages Storage Environment Executors **Spark shell** application UI

Spark Stages

Total Duration: 3.8 m
Scheduling Mode: FIFO
Active Stages: 0
Completed Stages: 2
Failed Stages: 0

Active Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write
0	count at <console>:13	2013/12/02 21:07:55	83 ms	2/2	754.0 B	
1	reduceByKey at <console>:13	2013/12/02 21:07:55	345 ms	2/2		1506.0 B

Failed Stages (0)

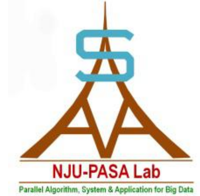
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write

4.Spark的程序执行过程

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

4.Spark的程序执行过程



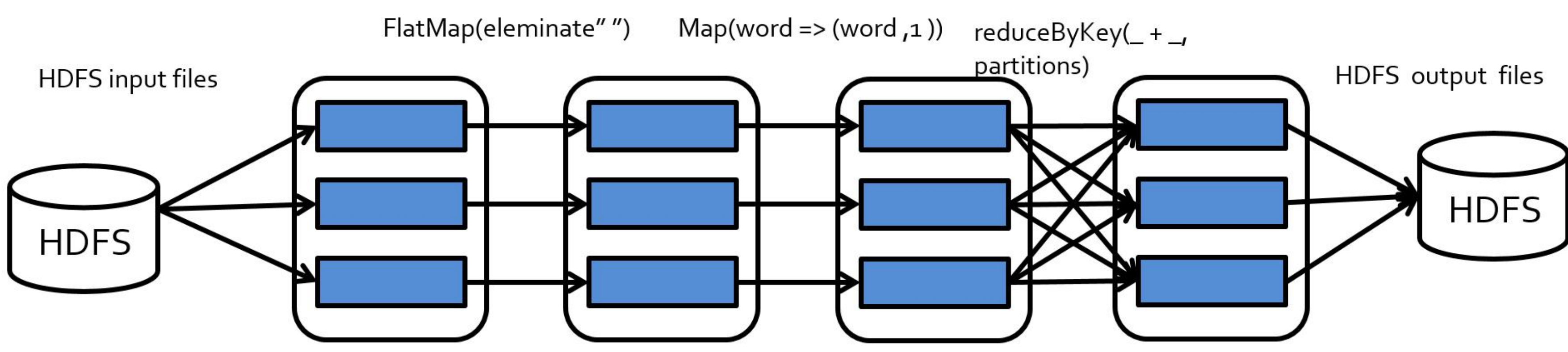
基于RDD世系的并行执行优化

A:
B:
G:
groupBy
Stage 1
Task
C:
D:
F:
map
E:
join
union
Stage 2
Stage 3

= previously computed partition

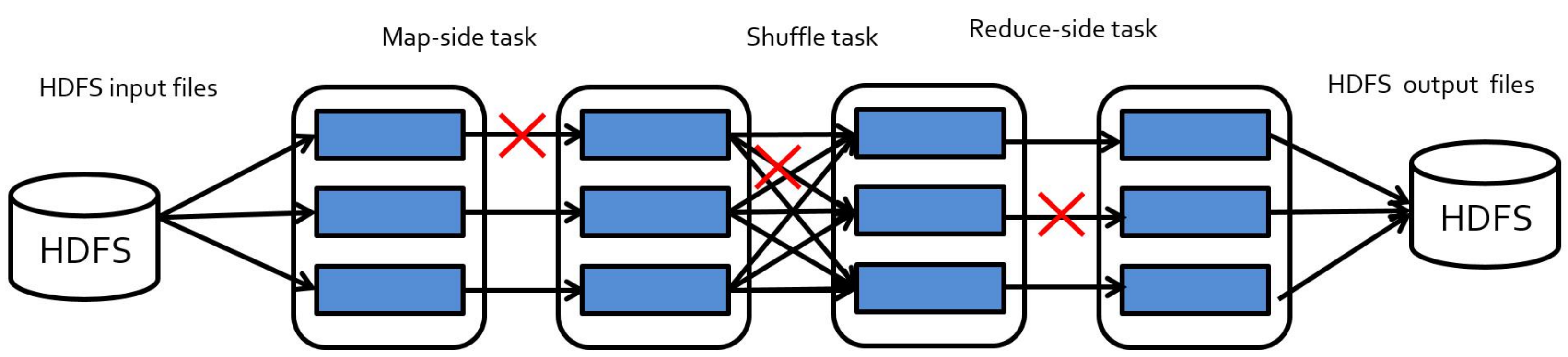
4.Spark的程序执行过程

细粒度容错

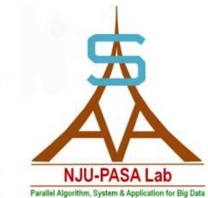


4.Spark的程序执行过程

细粒度容错

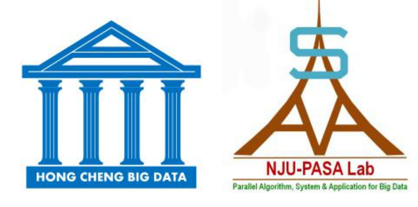


Spark 系统简介



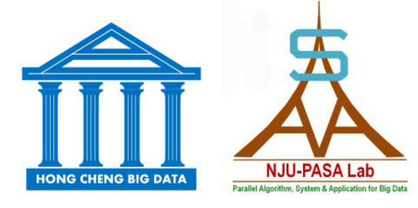
1.Scala编程语言简介
2.为什么会有Spark?
3.Spark的基本构架和组件
4.Spark的程序执行过程
5.Spark的技术特点
6.Spark编程模型与编程接口
7.Spark的安装运行模式

5.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。
- Lineage：为了保证RDD中数据的鲁棒性，Spark系统通过世系关系(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。

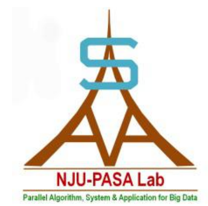
5.Spark的技术特点



- 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、Mesos、YARN等多种部署方式，可以部署在多种底层平台上

综上所述，Spark是一种基于内存的迭代式分布式计算框架，适合于完成多种计算模式的大数据处理任务

Spark 系统简介



1.Scala编程语言简介

2.为什么会有Spark?

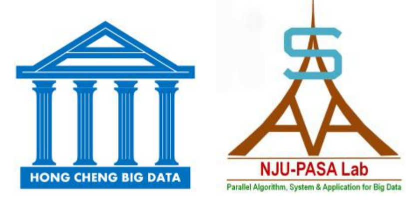
3.Spark的基本构架和组件

4.Spark的程序执行过程

5.Spark的技术特点

6.Spark编程模型与编程接口

6.Spark编程模型与编程接口



- Spark为了解决以往分布式计算框架存在的一些问题(重复计算、资源共享、系统组合)，提出了一个分布式数据集的抽象数据模型：

RDD(Resilient Distributed Datasets)弹性分布式数据集

  – RDD是一种分布式的内存抽象，允许在大型集群上执行基于内存的计算（In-Memory Computing），同时还保持了MapReduce等数据流模型的容错特性。
  – RDD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。
  – 简单来说，RDD是MapReduce模型的一种简单的扩展和延伸。

6.Spark编程模型与编程接口

Spark的基本编程方法与示例

//在一个存储于HDFS的Log文件中，计算出现ERROR的行数，本程序使用Scala语言编写，这个语言也是Spark开发和编程的推荐语言。

def main(args: Array[String]) { //定义一个main函数

val conf = new SparkConf().setAppName("Spark Pi") //定义一个sparkConf，提供Spark运行的各种参数，如程序名称、用户名称等

val sc = new SparkContext(conf) //创建Spark的运行环境，并将Spark运行的参数传入Spark的运行环境中

val fileRDD=sc.textFile("hdfs:///root/Log") //调用Spark的读文件函数，从HDFS中读取Log文件，输出一个RDD类型的实例：fileRDD。具体类型：RDD[String]

val filterRDD=fileRDD.filter(line=>line.contains("ERROR")) //调用RDD的filter函数，过滤fileRDD中的每一行，如果该行中含有ERROR，保留；否则，删除。生成另一个RDD类型的实例：filterRDD。具体类型:RDD[String]

注：line=>line.contains("ERROR")表示对每一个line应用contains()函数

val result = filterRDD.count() //统计filterRDD中总共有多少行，result为Int类型

sc.stop() //关闭Spark

}

6.Spark编程模型与编程接口

RDD的创建

- val file=sc.textFile("hdfs:///root/Log") 这句代码创建了一个RDD，那么RDD是怎么创建的？又有那些注意事项？
- 从形式上看，RDD是一个分区的只读记录的集合。因此，RDD只能通过两种方式创建：

  1、在驱动程序中并行化一个已经存在的集合，或者引用一个外部存储系统的数据集，例如共享的文件系、HDFS、HBase或其他Hadoop数据格式的数据源。例如：

  ```
  val rdd = sc.parallelize(1 to 100, 2)
  //生成一个1到100的数组，并行化成RDD
  ```

  2、其他RDD的数据上的确定性操作来创建(即Transformation)。例如：

  ```
  val filterRDD=file.filter(line=>line.contains("ERROR"))
  //通过file的filter操作生成一个新的filterRDD
  ```

6.Spark编程模型与编程接口

RDD的操作

- RDD支持两种类型的操作：

➢ 转换(transformation)：这是一种惰性操作，即使用这种方法时，只是定义了一个新的RDD，而并不马上计算新的RDD内部的值。

例：val filterRDD=fileRDD.filter(line=>line.contains(“ERROR”))

上述这个操作对于Spark来说仅仅记录从file这个RDD通过filter操作变换到filterRDD这个RDD的变换，并不计算filterRDD的结果。

➢ 动作(action)：立即计算这个RDD的值，并返回结果给程序，或者将结果写入到外存储中。

例：val result = filterRDD.count()

上述操作计算最终的result结果是多少，包括前边transformation时的变换。

6.Spark程序模型

HONG CHENG BIG DATA

RDD的transformation示例

例： val filterRDD=fileRDD.filter(line=>line.contains(“ERROR”))

设fileRDD中包含以下7行数据：

”ERROR1 ERROR2 ERROR3 TRUE4 ERROR5 ERROR6 TRUE7”

File：RDD[String]

ERROR1
ERROR2
ERROR3
TRUE4

ERROR5
ERROR6
TRUE7

只记录不计算

filterRDD：RDD[String]

ERROR1
ERROR2
ERROR3

ERROR5
ERROR6

6.Spark程序模型

HONG CHENG BIG DATA

RDD的action图示

例： val result = filterRDD.count()

File：RDD[String]

ERROR 1
ERROR 2
ERROR 3
ERROR 4 TRUE5
ERROR 5
ERROR 6
TRUE7

只记录不计算

filterRDD：RDD[String]

ERROR 1
ERROR 2
ERROR 3
ERROR 5
ERROR 6

计算

Result:Int

5

6.Spark程序模型

Spark 支持的一些常用 transformation操作

Transformation	Meaning
map(func)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(func)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
mapPartitions(func)	Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
mapPartitionsWithIndex(func)	Similar to mapPartitions, but also provides *func* with an integer value representing the index of the partition, so *func* must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T.
sample(withReplacement, fraction, seed)	Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator seed.
union(otherDataset)	Return a new dataset that contains the union of the elements in the source dataset and the argument.

Transformation	Meaning
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numTasks*]))	Return a new dataset that contains the distinct elements of the source dataset.
groupByKey([*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.
reduceByKey(*func*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.
aggregateByKey(*zeroValue*)(*seqOp*, *combOp*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.
sortByKey([*ascending*], [*numTasks*])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument.

Transformation	Meaning
join(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through leftOuterJoin,rightOuterJoin, and fullOuterJoin.
cogroup(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples. This operation is also called groupWith.
cartesian(*otherDataset*)	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).
pipe(*command*, *[envVars]*)	Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script. RDD elements are written to the process's stdin and lines output to its stdout are returned as an RDD of strings.
coalesce(*numPartitions*)	Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.
repartition(*numPartitions*)	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.
repartitionAndSortWithinPartitions(*partitioner*)	Repartition the RDD according to the given partitioner and, within each resulting partition, sort records by their keys. This is more efficient than calling `repartition` and then sorting within each partition because it can push the sorting down into the shuffle machinery.

6.Spark程序模型

Spark支持的一些常用action操作

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset.
takeSample(*withReplacement*,*num*, [*seed*])	Return an array with a random sample of *num* elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
takeOrdered(*n*, *[ordering]*)	Return the first *n* elements of the RDD using either their natural order or a custom comparator.

6.Spark程序模型

Action	Meaning
saveAsTextFile(*path*)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.
saveAsSequenceFile(*path*) (Java and Scala)	Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDs of key-value pairs that either implement Hadoop's Writable interface. In Scala, it is also available on types that are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
saveAsObjectFile(*path*) (Java and Scala)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using SparkContext.objectFile().
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
foreach(*func*)	Run a function *func* on each element of the dataset. This is usually done for side effects such as updating an accumulator variable (see below) or interacting with external storage systems.

6.Spark程序模型

RDD的容错实现

- 在RDD中，存在两种容错的方式:

  1.Lineage(世系系统、依赖系统)

  RDD提供一种基于粗粒度变换的接口，这使得RDD可以通过记录RDD之间的变换，而不需要存储实际的数据，就可以完成数据的恢复，使得Spark具有高效的容错性

  2.CheckPoint(检查点)

  对于很长的lineage的RDD来说，通过lineage来恢复耗时较长。因此，在对包含宽依赖的长世系的RDD设置检查点操作非常有必要

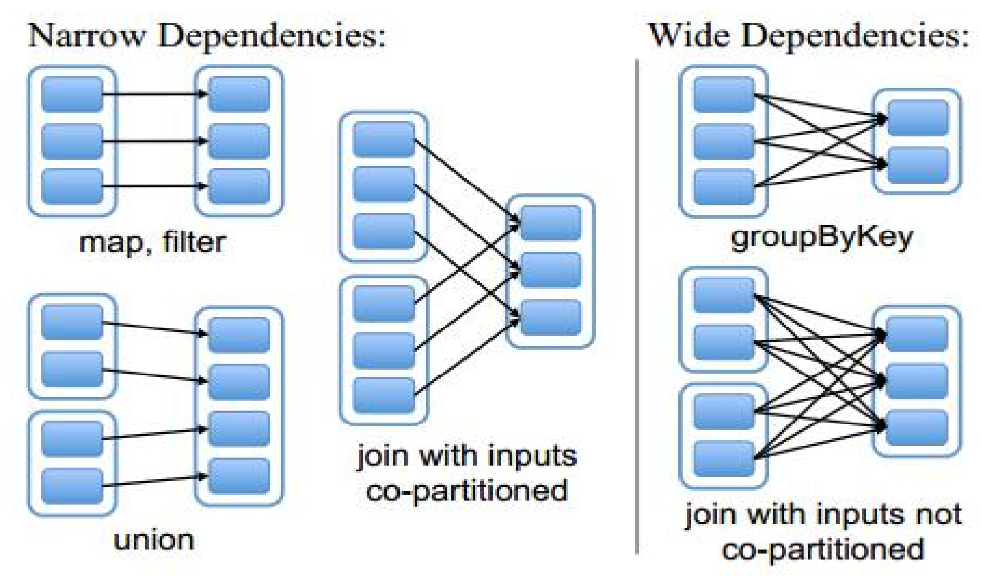
  由于RDD的只读特性使得Spark比常用的共享内存更容易完成checkpoint

HONG CHENG BIG DATA
NJU-PASA Lab
Parallel Algorithm, System & Application for Big Data

6.Spark程序模型

RDD之间的依赖关系

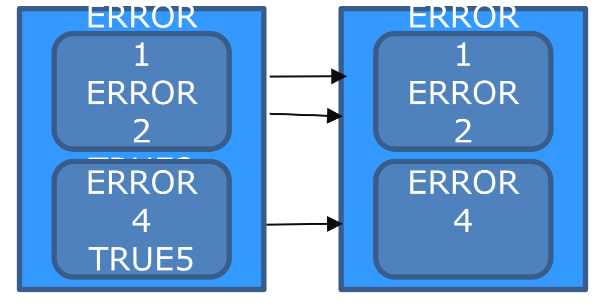
- 在Spark中存在两种类型的依赖：
  - **窄依赖**：父RDD中的一个Partition最多被子RDD中的一个Partition所依赖。
  - **宽依赖**：父RDD中的一个Partition被子RDD中的多个Partition所依赖。



例子:

```
val filterRDD=fileRDD.filter
        (line=>line.contains("ERROR"))
```

是一种窄依赖



6.Spark程序模型

RDD持久化

Spark提供了三种对持久化RDD的存储策略：

- 未序列化的Java对象，存于内存中

  性能表现最优，可以直接访问在JAVA虚拟机内存里的RDD对象

- 序列化的数据，存于内存中
  - 取消JVM中的RDD对象，将对象的状态信息转换为可存储形式，减小RDD的存储开销，但使用时需要反序列化恢复。
  - 在内存空间有限的情况下，这种方式可以让用户更有效的使用内存，但是这么做的代价是降低了性能。

- 磁盘存储
  - 适用于RDD太大难以在内存中存储的情形，但每次重新计算该RDD都会带来巨大的额外开销。

HONG CHENG BIG DATA
NJU-PASA Lab
Parallel Algorithm, System & Application for Big Data

完整的存储级别介绍

Storage Level	Meaning
MEMORY_ONLY	将RDD作为非序列化的Java对象存储在jvm中。如果RDD不适合存在内存中，一些分区将不会被缓存，从而在每次需要这些分区时都需重新计算它们。这是系统默认的存储级别。
MEMORY_AND_DISK	将RDD作为非序列化的Java对象存储在jvm中。如果RDD不适合存在内存中，将这些不适合存在内存中的分区存储在磁盘中，每次需要时读出它们。
MEMORY_ONLY_SER	将RDD作为序列化的Java对象存储（每个分区一个byte数组）。这种方式比非序列化方式更节省空间，特别是用到快速的序列化工具时，但是会更耗费cpu资源。
MEMORY_AND_DISK_SER	和MEMORY_ONLY_SER类似，但不是在每次需要时重复计算这些不适合存储到内存中的分区，而是将这些分区存储到磁盘中。
DISK_ONLY	仅仅将RDD分区存储到磁盘中
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	和上面的存储级别类似，但是复制每个分区到集群的两个节点上面
OFF_HEAP (experimental)	以序列化的格式存储RDD到Tachyon中。相对于MEMORY_ONLY_SER，OFF_HEAP减少了垃圾回收的花费，允许更小的执行者共享内存池。这使其在拥有大量内存的环境下或者多并发应用程序的环境中具有更强的吸引力。

6.Spark程序模型

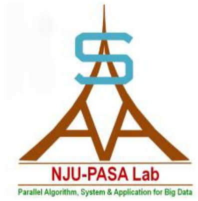
RDD内部设计

每个RDD都包含:

- 一组RDD分区（partition），即数据集的原子组成部分
- 对父RDD的一组依赖，这些依赖描述了RDD的Lineage
- 一个函数，即在父RDD上执行何种计算
- 元数据，描述分区模式和数据存放的位置

RDD内部接口:

操作	含义
partitions()	返回一组Partition对象
preferredLocations(p)	根据数据存放的位置， 返回分区p在哪些节点访问更快
dependencies()	返回一组依赖
iterator(p, parentIters)	按照父分区的迭代器，逐个计算分区p的元素
partitioner()	返回RDD是否hash/range分区的元数据信息



Spark编程接口

- Spark用Scala语言实现了RDD的API
- Scala是一种基于JVM的静态类型、函数式、面向对象的语言Scala具有简洁（特别适合交互式使用）、有效（因为是静态类型）等优点
- Spark支持三种语言的API:
  - Scala
  - Python
  - Java

Spark系统及其编程技术

- Spark系统简介
- Spark编程示例
- Spark生态系统中其它功能组件简介
- Spark 2.x新特性介绍

Spark 编程示例

WordCount MapReduce代码：

Map类代码

```
//定义Map类实现字符串分解
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  //实现map()函数
  public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException  {
  //将字符串拆解成单词
    StringTokenizer itr = new StringTokenizer(value.toString());
     while (itr.hasMoreTokens())  {
         word.set(itr.nextToken());  //将分解后的一个单词写入word类
         context.write(word, one);      //收集<key, value>
     }
  }
}
```

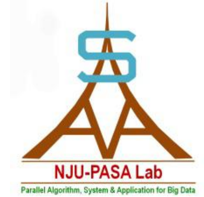
Spark 编程示例

WorCount MapReduce代码:

Reduce类代码

```
//定义Reduce类规约同一key的value
public static class IntSumReducer extends
Reducer<Text,IntWritable,Text,IntWritable>
 {
    private IntWritable result = new IntWritable();
    //实现reduce()函数
    public void reduce(Text key, Iterable<IntWritable> values, Context context )
                        throws IOException, InterruptedException
    {
        int sum = 0;
        //遍历迭代values，得到同一key的所有value
        for (IntWritable val : values) {  sum += val.get(); }
        result.set(sum);
        //产生输出对<key, value>
        context.write(key, result);
    }}
```

Spark 编程示例

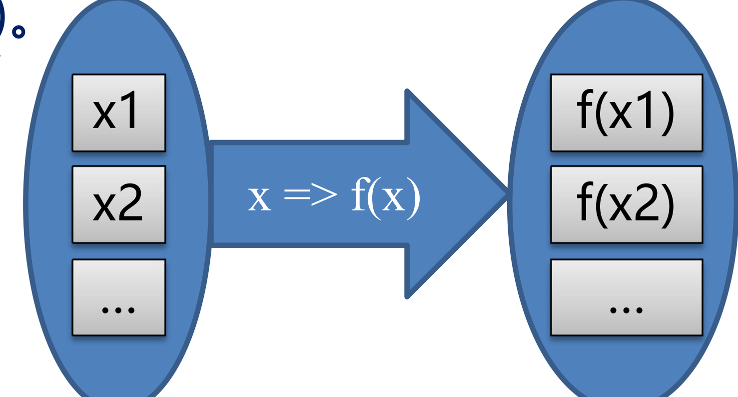


WordCount Spark Scala代码：

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(“ ”))          //分词
                 .map(word => (word, 1))                     //对应mapper的工作
                 .reduceByKey(_ + _)
                              //相同key的不同value之间进行”+”运算
counts.saveAsTextFile("hdfs://...")
```

这里，map操作表示对列表中的每个元素应用一个函数，在scala中，函数可以写成”x => f(x)”的形式，也可以更简洁地写成f(_)。例如line => line.split(“ ”)可以简写为_.split(“ ”)。

flatMap是在map之后增加了一个“扁平化”的操作，将map之后可能形成的形如List(List(1,2), List(3,4))的数据集“扁平”为List(1,2,3,4)。



Spark 编程示例

WordCount Spark Java代码:

```
JavaRDD<String> file = spark.textFile("hdfs://...");
JavaRDD<String> words = file.flatMap
        (   new FlatMapFunction<String, String>()
            {    public Iterable<String> call(String s) { return Arrays.asList(s.split(" ")); }
            }
        );//对应flatMap(line => line.split(“ ”)) 操作
JavaPairRDD<String, Integer> pairs = words.mapToPair
        (   new PairFunction<String, String, Integer>()
            {    public Tuple2<String, Integer> call(String s)
                 {   return new Tuple2<String, Integer>(s, 1);   }
            }
        );//对应map(word => (word, 1))
JavaPairRDD<String, Integer> counts = pairs.reduceByKey
        (   new Function2<Integer, Integer>()
            {    public Integer call(Integer a, Integer b) { return a + b; }
            }
        );//对应reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...");
```

Spark系统及其编程技术

- Spark系统简介
- Spark编程示例
- Spark生态系统中其它功能组件简介
- Spark 2.x新特性介绍

Spark生态系统中其它功能组件简介

主要体系结构和组件

Spark SQL	Spark Streaming	MLlib	GraphX

Spark		
MapReduce	RDD	FP

本地运行模式	独立运行模式	Mesos	YARN

HDFS　Hypertable，Hbase等

- Spark SQL、Spark Streaming、Mllib、GraphX是Spark提供的一系列高层工具，它们将在后面章节被详细介绍。同级的还有Bagel、shark等工具。
- FP：即函数式编程（function programing ）
- Mesos、YARN：即apache Mesos和YARN（Hadoop NextGen）两套资源管理框架，可以作为Spark的运行模式。同级的还有Amazon EC2等。

Spark生态系统中其它功能组件简介

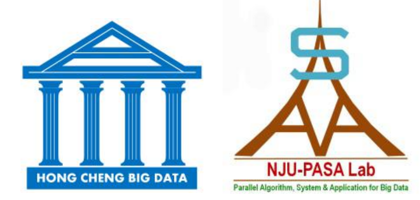
1.Spark SQL

2.Spark Streaming

3.GraphX

4.MLlib

1. Spark SQL



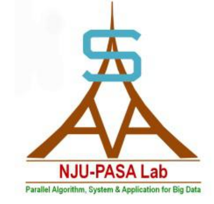
Spark SQL 是一个用来处理结构化数据的分布式SQL查询引擎，具有以下几个特点：

- 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。

2. Spark Streaming

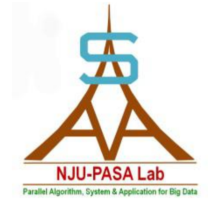
Spark Streaming 是一个对实时数据进行高吞吐量、容错处理的流式数据处理系统。它可以对多种数据源（Kafka、 Flume、Twitter、TCP 套接字等）进行各种复杂处理（map、reduce、join、window等），处理结果可以输出到文件系统、数据库或实时显示。

2.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。

3. GraphX



- GraphX是Spark系统中对图进行表示和并行处理的组件，它把图抽象为：给每个顶点和边附着了属性的有向多重图（directed multigraph）。
- GraphX提供了一系列基本图操作（比如subgraph、joinVertices、aggregateMessages等）和优化了的Pregel API变种，并且各种图算法还在不断丰富中。

基本信息接口
- numEdges
- numVertices
- degrees(in/out)

转换操作
- mapVertices
- mapEdges
- mapTriplets

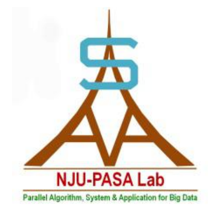
结构操作
- reverse
- subgraph
- mask
- groupEdges

关联操作
- joinVertices
- outerJoinVertices

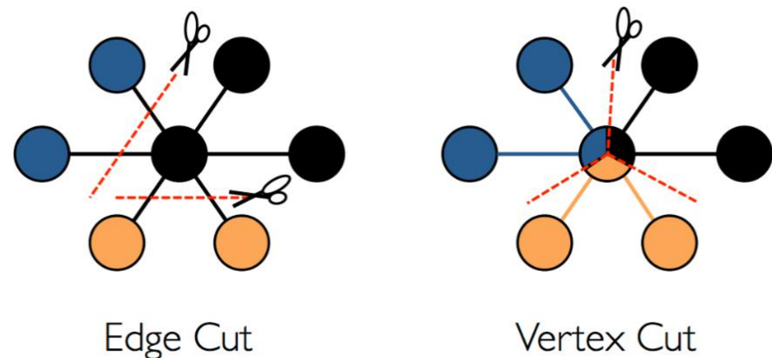
聚合操作
- mapReduceTriplets
- collectNeighbors

缓存操作
- persist
- cache
- unpersistVertices

3.GraphX



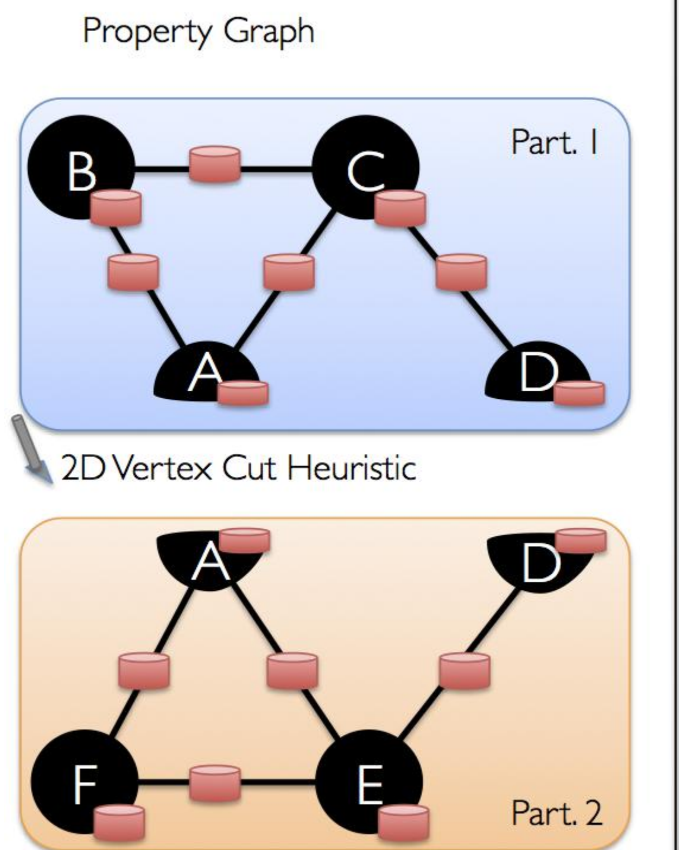
- GraphX使用高效的点分割存储模式。

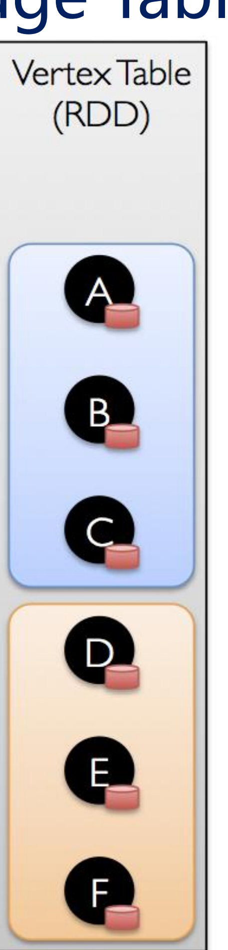


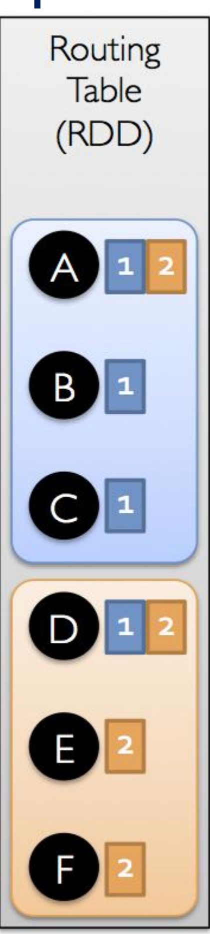
- 图的每一条边只会存储在一台机器上，但顶点可能存储在多台机器上（如下一页图所示）。当点被分割到不同机器上时，是相同的镜像，但是有一个点作为主点(master),其他的点作为虚点(ghost)，当点B的数据发生变化时,先更新点B的master的数据，然后将所有更新好的数据发送到B的ghost所在的所有机器，更新B的ghost。

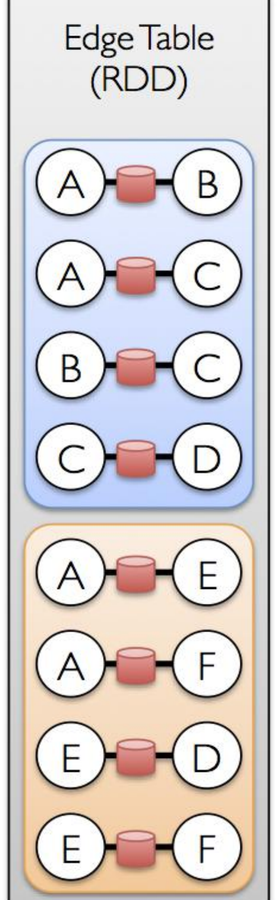
3.GraphX

- **Routing Table**：vertex Table中的一个partition对应着Routing Table中的一个partition，Routing Table指示了一个vertex会涉及到哪些Edge Table partition。









4.MLlib

- MLlib是Spark的分布式机器学习算法库，包含了很多常用机器学习算法和工具类

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

小象科技
让你的数据产生价值

Spark系统及其编程技术

- Spark系统简介
- Spark编程示例
- Spark生态系统中其它功能组件简介
- Spark 2.x新特性介绍
- Spark 安装配置速览

Spark2.x新特性介绍

1. SparkSession

   为用户提供了统一HiveContext和SQLContext的入口

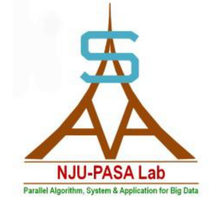
2. Tungsten引擎优化

   通过建立新的内存管理机制，提升Spark SQL性能

3. 统一DataFrame和DataSet

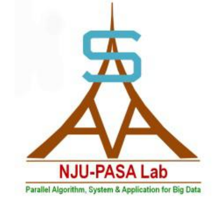
   DataFrame=DataSet[Row]

DataFrame和DataSet



- 这三者都是不可变的分布式数据集合。
- 但是不同的地方在于，DataFrame更像是一张数据库里的表，有自己的Schema（就是列名）。
- DataSet的功能更为强大。它是一种“强类型”（strongly-typed）JVM对象。Spark 2.0之后DataFrame就是DataSet[Row]的别名了。Row是一种无类型的JVM对象。

DataFrame和DataSet

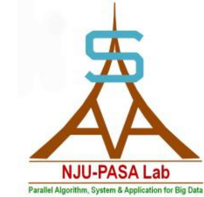


例：假设有一个Product类，case class Person(name:String, age:Int)
RDD[Person]中的元素类型在执行的时候都是会类型擦除的，记录是以Person为单位的。
DataFrame由于在创建的时候有Schema，所以对于列的访问可以通过Schema执行。
DataSet给用户的视角和DataFrame一样，但是它的强大之处在于创建的时候都不需要Schema，成员变量名集合就是Schema。

Person
Person
Person
Person

name	age
String	Int
String	Int
String	Int
String	Int

DataFrame和DataSet



对于同样的一个功能，分别用三种数据结构来实现。

实现的功能：得到所有年龄大于20岁的人的信息。

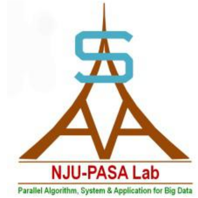
RDD的示例：

```
case class Person(name: String, age: Long)
val rdd: RDD[Person] = Seq(Person("Andy", 32)， Person("Bob", 18))
val ret = rdd.filter(p=>p.age > 20);
```

DataFrame示例：

```
val fields = Seq("name", "age")
  .map(fieldName => StructField(fieldName, StringType, nullable = true))
val schema = StructType(fields)
val rowRDD = rdd.map(attributes => Row(attributes(0), attributes(1)))
val peopleDF = spark.createDataFrame(rowRDD, schema)
peopleDF.createOrReplaceTempView("people")
//第一种方式
val results1 = spark.sql("SELECT name FROM people where age>20")
//第二种方式
val results2 = peopled.filter($"age" >20)
```

DataFrame和DataSet



DataSet**示例**:

使用方面来说DataSet和DataFrame没有区别，区别只是构造，生成一个DataSet只需要一句

```
val peopleDS = rdd.toDS()
```

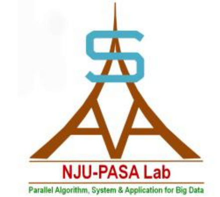
当然，如果已经有了DataSet对象，转化成DataFrame对象只需要下面这句

```
val peopleDF = peopleDS.toDF()
```

思考题:

1. DataSet是如何解析到Schema信息的?
2. 如果是Java语言，能不能实现该功能?

DataFrame和DataSet



DataFrame和DataSet的算子的实现实际上依然是用RDD完成的，那么其优势在哪里呢?

1. 类型安全

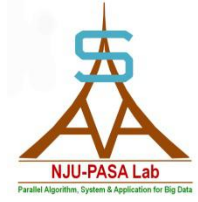
如果用RDD，那么元素成员变量的操作所涉及到的类型安全，只有到运行时才能检查出来。

如果使用DataFrame和DataSet，类型安全的检查可以提前到编译期。

DataSet甚至可以在编译期检查出分析相关的类型错误。

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

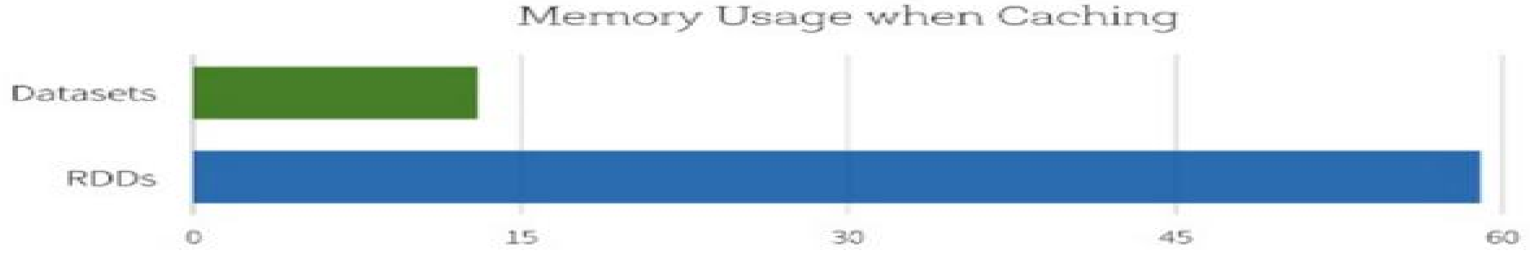
DataFrame和DataSet



2. **对结构化数据和半结构化数据更高层的抽象**
DataFrame和DataSet中的每一列不仅可以是基础类型，还可以是数组、散列表、和其他结构。所以其比数据库中的类型更为丰富。

3. **对于结构化数据的API**
习惯了函数式编程，也不妨试试SQL的语法。DataSet和DataFrame包含类似于SQL语法的算子，使用方法和rdd其他操作一样。

4. **性能优势**
DataSet的元素的数据结构比RDD更为紧凑，占用空间更小，数据扫描更快。
使用RDD编程的时候，一般的程序员使用的数据结构都很低效，JVM中一个对象占用的空间比该对象包含的数据要大很多。DataSet中的数据结构更类似于原生的byte数组，对不同成员变量的访问就是对数组不同偏移量的访问，所以可以想象其效率有多高。



Spark系统及其编程技术

- Spark系统简介
- Spark编程示例
- Spark生态系统中其它功能组件简介
- Spark 2.x新特性介绍
- Spark 安装配置速览

Spark系统运行的软件环境

- **操作系统**

  Spark是运行在Java虚拟机上的，因此在Windows、Linux和MacOS上都能够安装Spark。但由于Spark中的一些工具和脚本（如启动脚本）是针对Linux环境编写的，因此建议在Linux操作系统上安装和运行Spark。

- **SSH（Secure Shell）**

  主要用于在集群环境下远程管理Spark节点以及Spark节点间的安全共享访问。

- **Java**

  主要用于运行Spark以及使用Spark提供的Java API进行开发，如Java1.6.0

Spark系统运行的软件环境

- **Scala**

  除了Java API以外，Spark还向程序员提供了Scala API，如要用Scala开发Spark应用，则需要安装Scala，如Scala-2.10.4

- **Python**

  类似的，Spark也提供了Python API，如要用Python开发Spark应用，则需要安装Python，如Python-2.6.5

- **HDFS**

  Spark是一个分布式计算引擎，其输入输出数据可以保存在分布式文件系统中。这里推荐使用Hadoop中的HDFS，如Hadoop-2.2.0

安装Spark Standalone模式的基本步骤

基本安装步骤

- 软件环境准备
- 下载编译好的Spark包
- 修改Spark配置文件
- 启动Spark
- 运行测试程序
- 查看集群状态

Spark Standalone模式的安装过程

1. 软件环境的准备

Linux操作系统、SSH、Java、HDFS这几样软件的安装在之前的课程中已经介绍过，这里不再详述。

安装Scala的步骤（也可参考http://www.scala-lang.org/ ）

- 下载对应版本的Scala包，解压
- 添加环境变量$SCALA_HOME，修改环境变量$PATH

安装Python的步骤（也可参考https://www.python.org/ ）

- 下载对应版本的Python包，解压
- [user@master python]$ ./configure & make & make install

这里以Java-1.6.0，Hadoop-2.2.0，Scala-2.10.4，Python-2.6.5为例

Spark Standalone模式的安装过程

2. 下载编译好的Spark包

下载地址： https://spark.apache.org/downloads.html

- 选择需要下载的Spark版本
- 选择预编译的Hadoop版本
- 下载相应的Spark包

Download Spark

The latest release of Spark is Spark 1.3.1, released on April 17, 2015 (release notes) (git tag)

1. Chose a Spark release: 1.0.2 (Aug 05 2014)
2. Chose a package type: Pre-built for Hadoop 2.2
3. Chose a download type: Select Apache Mirror
4. Download Spark: spark-1.0.2-bin-hadoop2.tgz
5. Verify this release using the 1.0.2 signatures and checksums.

Note: Scala 2.11 users should download the Spark source package and build with Scala 2.11 support.

解压： [user@master spark_installs]$ tar -zxv

3. 修改Spark配置文件

Spark的配置文件存放在Spark安装目录下的conf目录中，需要将conf/xxx.template复制为conf/xxx，然后编辑conf/xxx进行配置：

- conf/spark-env.sh：主要完成Spark环境变量设置
- conf/slaves：主要完成Worker节点的IP设置

Spark Standalone模式的安装过程

3. 修改Spark配置文件

在conf/slaves中填写需要运行Worker的节点，如：

```
slave01
slave02
```

在conf/spark-env.sh中配置Spark相关的环境变量，可参照 https://spark.apache.org/docs/1.0.2/spark-standalone.html给出的说明，在 conf/spark-env.sh.template中也有对应配置项的注释。例如，修改Master的IP地址以及默认的端口和WebUI端口：

```
export SPARK_MASTER_IP=master
export SPARK_MASTER_PORT=60123
export SPARK_MASTER_WEBUI_PORT=60133
```

然后，将Master上整个Spark目录复制到每一个Worker节点上

```
[user@master ~]$ scp -r ~/spark_installs/spark-1.0.2 [worker]:~/spark_installs/
```

Spark Standalone模式的安装过程

4. 启动Spark

Spark提供了一系列用于启动/停止的脚本：

- sbin/start-master.sh： 启动Master
- sbin/start-slaves.sh： 启动所有的Worker
- sbin/start-all.sh： 启动Master和所有的Worker
- sbin/stop-master.sh： 停止Master
- sbin/stop-slaves.sh： 停止所有的Worker
- sbin/stop-all.sh： 停止Master和所有的Worker

执行启动脚本后，可以使用JPS命令查看进程信息：

```
[user@master spark-1.0.2]$ jps
```

若Spark正常启动，那么在Master节点会有一个Master进程，在每个Worker节点会有Worker进程

Spark Standalone模式的安装过程

5. 运行测试程序

启动Spark后，可以向Spark集群提交一个测试程序：

```
[user@master spark-1.0.2]$ ./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi --master spark://master:60123 \
  --executor-memory 20G --total-executor-cores 100 \
  lib/spark-examples-1.0.2-hadoop2.2.0.jar 1000
```

6. 查看集群状态

在Spark集群运行期间，可以用浏览器查看集群状态

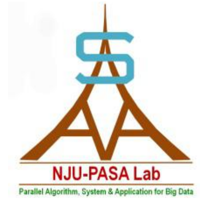
- 使用浏览器打开http://master-ip:webui-port/
- 可以看到集群信息（如右图）以及所有的Workers和Applications的信息

Spark Master at spark://master:60123

URL: spark://master:60123
Workers: 12
Cores: 192 Total, 0 Used
Memory: 728.7 GB Total, 0.0 B Used
Applications: 0 Running, 18 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

参考文献

- Spark官方网站 http://spark.apache.org/
- Spark主要开发者Matei Zaharia的博士论文：
  Zaharia M. An architecture for fast and general data processing on large clusters[R]. Technical Report No. UCB/EECS-2014-12, 3 Feb 2014. http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-12.html, 2014.
- 《2014年大数据技术与产业白皮书》
- Mesos官方网站 http://mesos.apache.org/
- Docker官方网站 http://www.docker.com/
- Hadoop官方网站中关于YARN的介绍
  http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
- 博客《利用Docker构建开发环境》 http://tech.uc.cn/?p=2726
- 博客《统一资源管理与调度平台（系统）介绍》
  http://dongxicheng.org/mapreduce-nextgen/mesos_vs_yarn/



Thank You！