Spark

No. Date :

1. Spark是什么.

小快速且通用的集群计算框架.

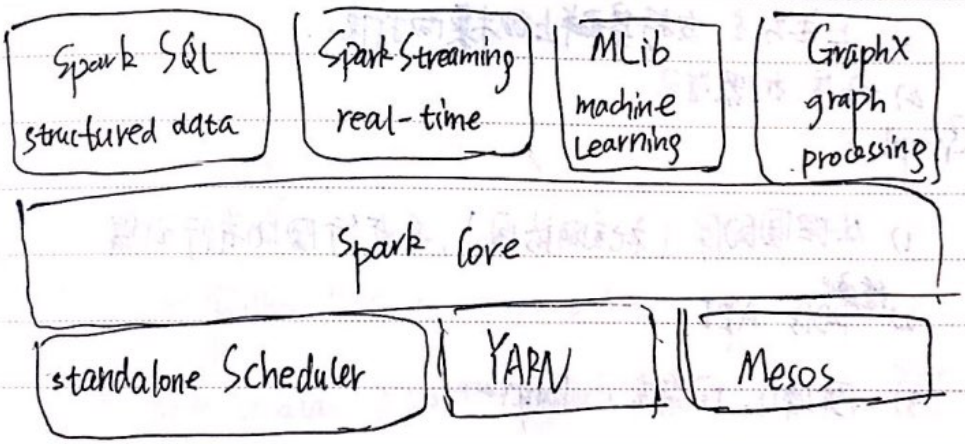
特. 快速: { 1. 扩充 M/R 计算模型.
2. 基于内存.

通用 { 1. 容纳其他分布式系统拥有的功能.
2. 批处理, 迭代计算, 交互查询, 和流处理.

优势: 降低成本.

2. Spark生态

Spark组件:



① Spark Core

1) 包含Spark基本功能, 包含任务调度, 内存管理、容错机制.

2) 内部定义了 RDDs (弹性分布式数据集).

3) API s → RDDs.

4) 为其他组件, 提供底层服务.

No. Date :

② Spark SQL

1) 兼容 Hive, MySQL

2) 场景：报表统计

③ Spark Stream

1) 实时数据流处理组件，类似 Storm.

2) API → 实时数据流.

3) 场景：从 kafka 接收数据做实时统计.

④. MLib

1) 包含通用机器学习的包.

2) 分类，聚类，回归等，模型评估，数据导入.

3) 上述方法，支持集群上的横向扩展.

4) 场景：机器学习.

⑤ GraphX

1) 处理图的库（社交网络图），并进行图的并行计算.

2) 继承 RDDs API

3) 图操作，图算法，PageRank.

4) 场景：图计算.

⑥. Cluster Managers

1) 自带集群管理：单独调度器；

2) 常见集群管理：YARN, Mesos.

No. Date :

① 紧密集成优点
1) 整体优化.
2) 便于部署
3) 共享组件.

3. Spark 与 Hadoop.

缺: hadoop: { 1. 离线处理. 2. 时效性不高. } Spark { 1. 时效性 2. 机器学习. }

比较: ① 各自生态.
② 不是各HDFS, 借助
③

二. Spark安装.

① Scala编写 → Java7+
② Python API → Python 2.6+ 或 Python 3.4+
③ Spark 1.6.2 ↔ Scala 2.10, Spark 2.0.0 - Scala 2.11

下载:
① spark.apache.org/downloads.html
② 不需要hadoop, 如果有, 可下载 hadoop-Spark 版本.
③ 解压.

No. Date :

目录：

① bin → Spark Shell

② core, streaming, python → 源码.

③ examples ⟷ 单机 Spark job.

Spark Shell

① 处理分布函集群上函数据

② 把数据加载到节点函内存中，秒级

③ 迭代式计算，实时查询，分析 在shells中完成.

④ Python shells, Scalar shells.

Python shell:

bin/pyspark.

Scala shell:

bin/spark-shell

var lines = sc.textFile("../../testfile/hellospark")

lines.count()

lines.first()

修改日志级别

conf/log4j, cp log4j.properties.

vi log4j.properties. log4j.rootCategory=WARN,console

No. Date :

三 Spark开发环境搭建.

① idea.

② 插件安装: Scalar, 重启idea.

③ 新建 → Scalar → SBT打包, jdk1.8.0_40, SBT 0.13.8, Scalar 2.10.5.

④ 常见问题:

版本兼容 → Spark 1.6.2

ssh配置.

① ssh-keygen.

② cat rsa-pub.key > authorized_keys

③ chmod 600 authorized-keys.

Spark程序:

① Scalar Class → word Count → Object.

② 代码. 1) conf=, sc= 2) input=, 当 lines, count= 4) output

③ project Structure

Aatifcats → jar → From modules with depadens

Module (依赖). → Main Class → copy empty class → ok

Build Artifacts → build

④ 启动集群 JPS 1) 启动Master ./sbin/start-master.sh 2) 启动worker ./bin/spark-class 3) 提交作业 ./bin/spark-submit

No. Date :

Spark web

localhost:8080.

Spark master at spark://主机名:7077

./bin/spark-class org.apache.spark.deploy.worker.Worker 主机名:7077

./bin/spark-submit --master 主机名:7077 --class WordCount.jar
↓
主class

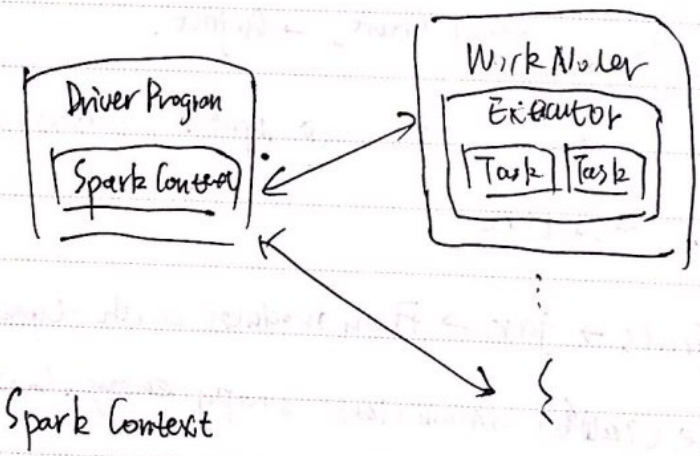
localhost:4040/jobs/ 任务进度条.

四. RDDs.

1. 介绍.

1) Driver Program:

包含 main(), RDDs 定义. 操作.



2) Spark Context

- 访问 Spark
- 一个和 spark 集群的连接.
- 在 shell 中, SC 已经创建
- RDDs 分发数据和计算的基础

No. Date :

RDDs:

一个RDD是一个不可改变的分布式集合对象。

Spark中，所有计算 → RDDs创建，转换，操作完成。

一个RDD内部由许多partitions（分片）组成。

分片：

partition在不同机器。

spark并行处理单元，spark顺序的并行且处理分片。

RDDs的创建：

把一个已存在的集合传给SparkContext的parallelize()方法。

```
var rdd = sc.parallelize(Array(1,2,2,4),4)
```

第一个参数：待并行处理的集合，第二个参数：分区个数。

Scalar变量声明：

- val，或var
- val变量值不可修改，一旦分配不能重新
- var分配后，可按相同类型的值。

Scalar匿名函数和类型推断：

- lines.filter(line => line.contains("world"))

  String推断。

No. Date :

2. RDDs 的 — Transformations

map > val lines = sc.parallelize(Array("hello", "spark", "hello world", "!"))

↓ 已存在的分布式数据集

> lines.foreach(println)

> val line2 = lines.map(word => (word, 1))

> line2.foreach(println).

filter > val line3 = lines.filter(word => word.contains("hello"))

> line3.foreach(println)

flatMap > val inputs = sc.textFile("*.text")

> inputs.foreach(println)

> val lines = inputs.flatMap(line => line.split(" "))

> lines.foreach(print).

3. 集合：并集，交集

> val rdd1 = sc.parallelize(自数组)

> val rdd_distinct = rdd1.distinct()

> 打印

> val rdd_union = rdd1.union(rdd2)

> val rdd_sub = rdd1.subtract(rdd2)

No. Date :

4. RDDs - Action

> reduce()

> val rdd = sc.parallelize(Array(1,2,2,3))

> rdd.collect()

> rdd.reduce((x,y)=> x+y)

> rdd.take(3)

> rdd.top(2)

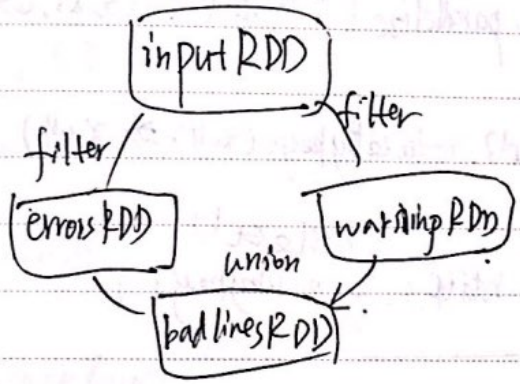
> rdd.foreach(println)

5. RDDs 特性

① 血统关系图

1) Spark 维护着 RDD 之间的依赖关系和创建关系，叫做血统关系图。

2) Spark 使用它来计算 RDD 的需求和恢复丢失的数据。



② 延迟计算

1) 第一次使用 action。

2) 减少数据传输

3) metadata 表为 transformation 操作之前的。

4) 加载数据，必要时。

No. Date :

⑤. RDDs 持久化.

rdd.persist() 重复利用

rdd.unpersist() 解除.

Memory, seq, Disk.

空间. CPU, 磁盘, 内存.

6. keyValue对 RDDs.

```
> val rdd=sc.textFile("x.txt")
> val rdd_map = rdd.map(line => (line.split(" ")(0), line))
                                          key-value
> val rdd3 = sc.parallelize((数组(<1,2>,<3,4>,<3,6>)
> val rdd4 = rdd3.reduceByKey((x,y) => x+y) ;-<1,2><3,10>
                   .collect()
> val rdd5 = rdd3.groupbykey()
```

```
> var rdd6 = rdd3.keys
> var rdd7 = rdd3.values
> var rdd8 = rdd3.sortBykey()
```

No. Date :

1. combineByKey(): 聚合函数.

返回类型可以与输入类型不一致.

key → 没过: mergeValue()

→ 未见过: createCombiner() → partition中之后在 mergeCombiners()中处理.

求平均值.

> val scores = sc.parallelize(Array(("Jack", 80), ("Jack", 90), ("Jack", 85), ("Mike", 80), ("Mike", 85), ("Mike", 100)))

> val scores2 = scores.combineByKey(score => (1, score), (c1: (Int, Double), newScore) => (c1._1 + 1, c1._2 + newScore), (c1: (Int, Double), c2: (Int, Double)) => (c1._1 + c2._1, c1._2 + c2._2))

(Jack, (3, 255.0))

(Mike, (3, 267.0))

> var average = scores2.map{ case (name, (num, score)) => (name, score/num) }

(Mik, 85)

(Jack, 87.5)