

实验2：Python MapReduce

4学时 · 起点镜像 L02

今天的成果：**一次结果可手算核对的YARN词频作业**

实验任务与课堂安排

阶段	时间	成果
启动与手算	25分钟	8个词的计数
Mapper	40分钟	8条中间记录
Reducer	35分钟	4条聚合结果
YARN作业	35分钟	application与HDFS输出
修改与核对	45分钟	大小写实验、报告

词频问题的数学表达

给定单词序列，计算每个词出现的次数：

$$c(w) = \sum_{i=1}^n \mathbf{1}(x_i = w)$$

Map为每次出现生成 $(w, 1)$

同一词的中间记录归为一组

Reduce对该组的1求和

小样本让每个阶段都能手算核对。

开始前确认

```
labctl stop  
labctl start lab02  
cd ~/work/lab02  
cp /opt/course/starter/REPORT.md REPORT.md
```

直接用L02开始，本次材料与配置由L02独立提供。

一份能手算的输入

```
spark hadoop spark  
data spark data  
hadoop python
```

一共8个词。先写下每个词的次数。

期望结果

单词	次数
data	2
hadoop	2
python	1
spark	3

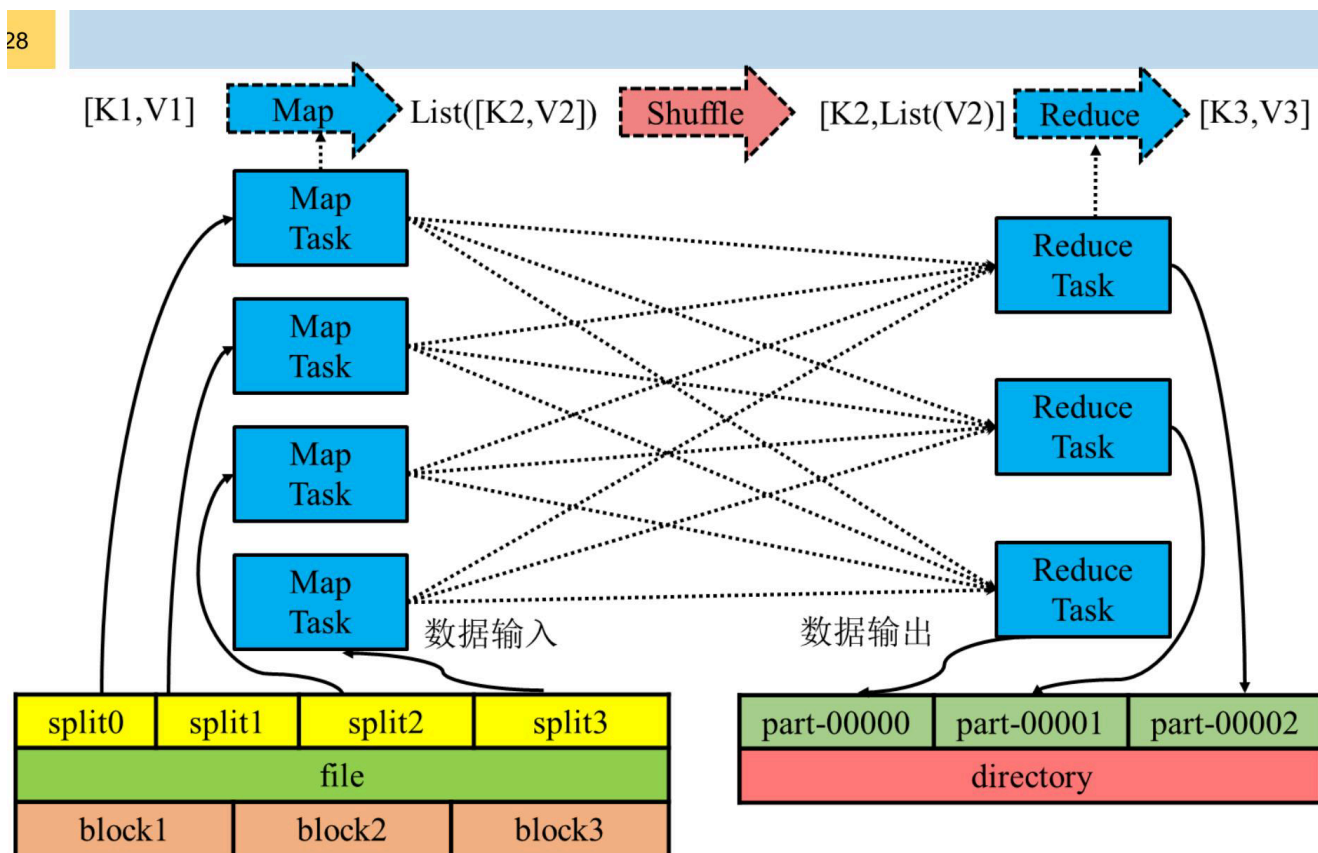
逐项核对四个词的计数，合计应为8。

Map、Shuffle与Reduce的数据流

Map输出中间键值对

Shuffle将相同键组织到对应Reducer

Reduce生成每个键的最终统计



输入行与Map输出的数量

输入第一行: `spark hadoop spark`

位置	单词	Map记录
1	spark	<code>(spark, 1)</code>
2	hadoop	<code>(hadoop, 1)</code>
3	spark	<code>(spark, 1)</code>

三行输入包含8个词, Mapper应产生8条记录。

Map阶段：每出现一次就记一笔

输入 `spark hadoop spark` :

```
spark    1  
hadoop   1  
spark    1
```

Mapper将每次出现转换成一条局部记录。

键值对与制表符接口

```
spark\t1
```

左侧键为单词，右侧值为计数

`\t` 在Python字符串中表示一个制表符

`print` 默认在末尾添加换行

Reducer通过制表符拆出键和值

可以用 `cat -T` 将真实TAB显示成 `^I`。

Python标准输入、输出与错误

```
import sys
print("example", 1, sep="\t")
print("正在处理一行", file=sys.stderr)
```

stdin接收输入记录

stdout输出后续计算要读取的数据

stderr输出调试信息与错误

数据和日志分别传递。

Python程序的输入输出

```
import sys
for line in sys.stdin:
    for word in line.split():
        # 在这里输出 word、制表符、1
        ...
```

stdout只输出数据；调试信息写stderr。

单独运行Mapper

```
python mapper.py < /opt/course/data/words.txt
```

< 把文件交给程序的标准输入

每个词输出一条，合计8行

每行严格采用 `word<TAB>1`

重定向与管道的执行顺序

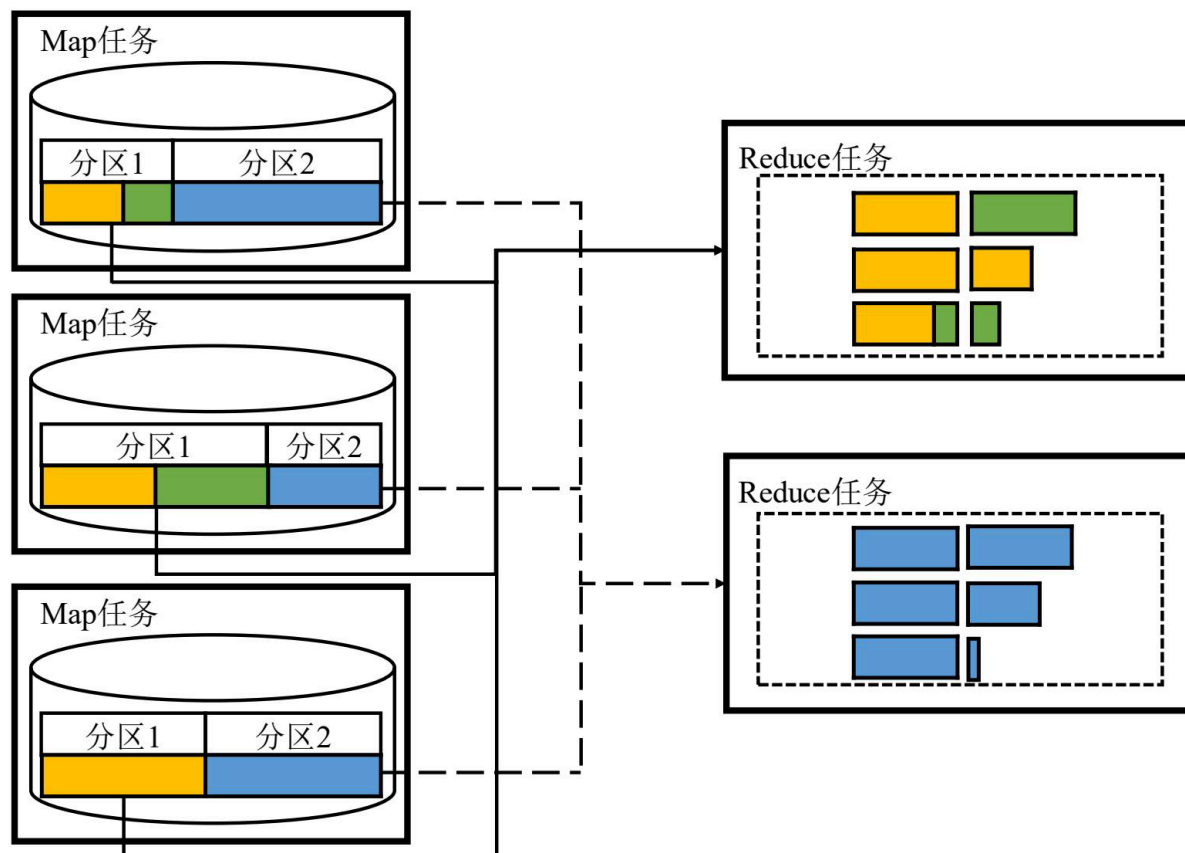
```
python mapper.py < /opt/course/data/words.txt >  
mapped.txt  
LC_ALL=C sort mapped.txt > sorted.txt  
python reducer.py < sorted.txt
```

< 接入文件, > 写出文件, | 连接两个程序。

分步保存中间文件, 可以定位最早出现错误的阶段。

Shuffle中的分区、排序与传输

分区决定哪个Reducer接收某个键
排序使相同键相邻，便于分组聚合
网络与磁盘承载中间记录的搬运



相同键需要进入同一分区

设有两个Reducer，概念上的分区规则为：

$$p(w) = \text{hash}(w) \bmod 2$$

相同单词得到相同分区编号

不同单词可以进入同一分区

一个Reducer可依次处理多个单词组

本实验使用一个Reducer，所有键进入同一分区。

按键排列中间记录

```
python mapper.py < /opt/course/data/words.txt |  
LC_ALL=C sort
```

同一个词挨在一起，Reducer才能连续累加。

真实Hadoop中的排序、分区与传输由Shuffle完成。

Reduce阶段：同一单词求和

```
data    1    ]  
data    1    ]    → data  2  
hadoop  1    ]  
hadoop  1    ]    → hadoop 2
```

维护“当前单词”和“当前累计值”。

Reducer的状态变量

变量	作用
current_word	当前正在累加的键
total	当前键已累计的次数
word、count	新读到的一条记录

同键时累加；换键时输出旧组，再建立新组。

按记录逐步推演Reducer

新记录	处理后当前键	累计值	当步输出
data,1	data	1	暂存
data,1	data	2	暂存
hadoop,1	hadoop	1	data,2
hadoop,1	hadoop	2	暂存

遇到下一个键，前一组的总数才完整确定。

输入结束与最后一组

```
python 1  
spark 1  
spark 1  
spark 1  
<输入结束>
```

结束时仍有 `spark=3` 保存在变量中。

循环结束后，若已有有效组，输出这组结果。

Reducer的两个边界

单词改变：输出前一组，开始新一组

输入结束：还要输出最后一组

空输入：不应输出 `None` 或伪造的零记录

只处理循环中间的情况，会漏掉最后一个词。

先跑通本地管道

```
python mapper.py < /opt/course/data/words.txt |  
    LC_ALL=C sort | python reducer.py  
printf ' ' | python reducer.py
```

先得到四个正确计数，再提交YARN。

局部测试的检查表

测试输入	要检查的行为
空文件	输出为空
一个词	该词计数为1
同一词重复三次	一组结果、计数3
两个词交替出现	排序后各自聚合
规定8词样例	四组结果、总计8

将输入、中间记录与最终输出分别保留。

Streaming接口与流计算的区别

名称	本课程作用
Hadoop Streaming	用标准输入输出接入Python程序
Structured Streaming	随新数据到达更新DataFrame计算

本实验提交有限文件的批处理作业。

实验5使用事件时间窗口处理持续到达的文件。

Hadoop Streaming的接口

Hadoop分配输入并启动你的程序

Python从stdin读、向stdout写

Hadoop负责Shuffle和作业调度

Reducer收到按键组织的中间记录

Hadoop Streaming采用外部程序接口，本次执行有限文件批处理。

YARN负责分配运行资源

组件	本次作用
ResourceManager	接收作业与分配资源
NodeManager	在本节点启动和管理容器
ApplicationMaster	协调本次应用的任务

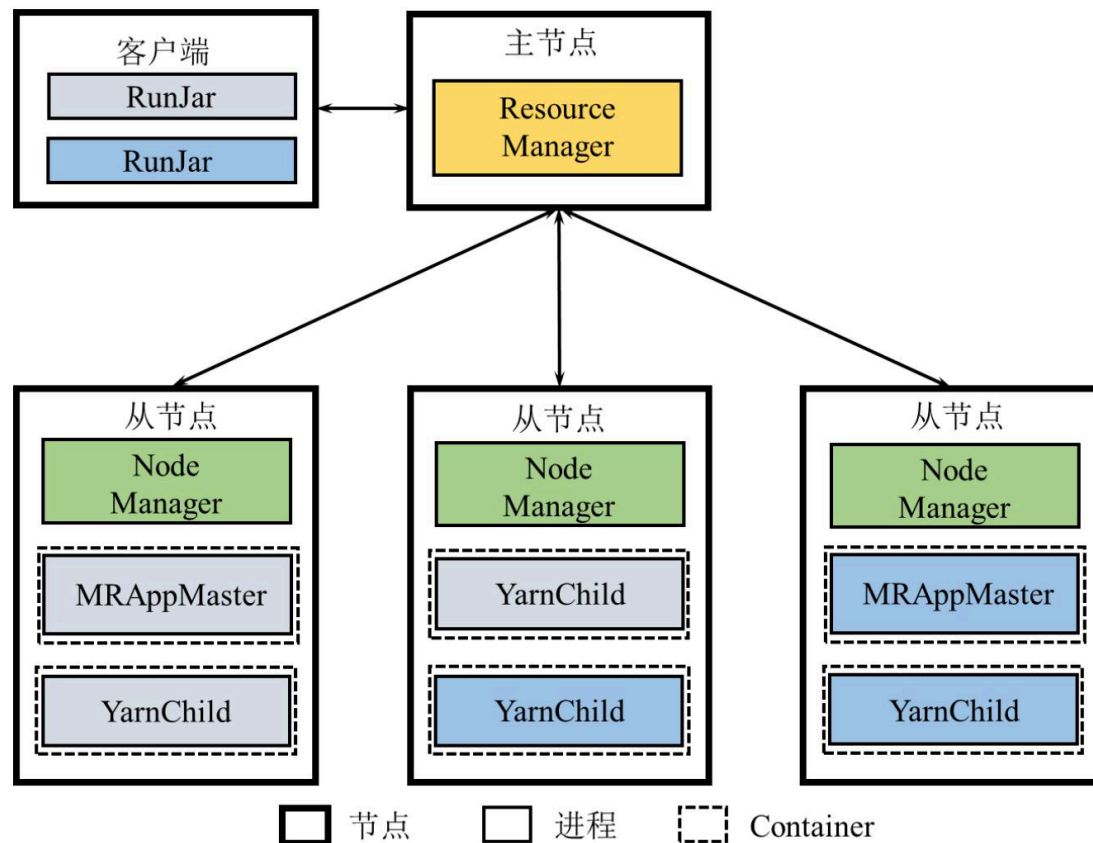
一次只运行一个课程作业。

YARN上的MapReduce作业

ResourceManager安排应用资源

NodeManager管理本机运行容器

ApplicationMaster协调Map与Reduce任务



submit.sh中的关键参数

```
-D mapreduce.job.reduces=1  
-files mapper.py,reducer.py  
-mapper 'python3 mapper.py'  
-reducer 'python3 reducer.py'
```

`-files` 将脚本分发到任务工作目录。

`-input` 与 `-output` 指定HDFS输入和输出路径。

提交命令与日志文件

```
set -o pipefail  
bash submit.sh 2>&1 | tee run.log
```

`2>&1` 把stderr合并到日志流

`tee` 同时显示并保存日志

`pipefail` 让前段失败反映到管道状态

最终结合YARN状态和数据结果核验

提交本次作业

```
bash submit.sh 2>&1 | tee run.log  
hdfs dfs -cat '/course/lab02/output/part-*'  
labctl check lab02
```

脚本会上传规定输入，分发Mapper与Reducer。

记录运行证据

YARN的application ID

最终状态： FINISHED、 SUCCEEDED

HDFS输出的四条计数

本地结果与集群结果的比较

网页通过SSH隧道访问宿主机的18088端口。

应用结束状态与最终结果

```
yarn application -status application_实际编号
```

字段	本次成功值
State	FINISHED
Final-State	SUCCEEDED

结合HDFS的 `part-*` 内容，确认应用完成且计数正确。

输出目录与part文件

```
/course/lab02/output/  
_SUCCESS  
part-00000
```

`part-*` 保存Reducer产生的数据

`_SUCCESS` 是作业成功提交输出的标记

多个Reducer可产生多个part文件

用 `hdfs dfs -cat '路径/part-*` 读取全部结果

为什么再次提交会失败

默认输出目录已经存在时，程序拒绝覆盖。

```
hdfs dfs -ls /course/lab02/output  
hdfs dfs -rm -r /course/lab02/output  
bash submit.sh
```

先保存结果；只删除本实验的output。

个人修改：大小写归一

加入 `Spark`，先预测未修改时的计数。

再让Mapper统一转成小写，比较结果。

新输入保存到个人文件

使用另一个HDFS输出路径

原规定样例仍需通过

大小写实验的预测与验证

额外输入: `Spark spark Hadoop`

规则	预测
保留大小写	Spark:1、spark:1、Hadoop:1
转为小写	spark:2、hadoop:1

先写预测，再修改Map输出前的词形转换。

额外样例使用个人文件和独立HDFS路径。

从失败应用定位Python错误

```
tail -n 30 run.log  
yarn logs -applicationId application_实际编号
```

找到Python异常类型、文件名和行号

检查Mapper或Reducer对应位置

使用同一输入在本地复现

修复后重新提交并核对结果

结果正确性与运行开销

检查问题	本次证据
算法是否正确	手算与四条词频一致
是否经过YARN	application与任务记录
文件是否写入HDFS	HDFS目录和part内容
耗时来自哪里	启动、调度、数据处理、提交

8个词的输入用于观察计算机制和执行流程。

按层定位失败

现象	优先检查
本地语法报错	Python源码与行号
Reducer拆分失败	中间记录是否为TAB分隔
集群任务失败	容器日志、文件分发、退出码
结果不对	手算样例与边界输入

报告截图核对

图1 YARN作业完成（报告4 提交YARN作业）

截图内容与图号按Word模板填写，直接插入对应位置。

提交Word报告

“源码与运行说明”： mapper.py全文、reducer.py全文

完成各步骤实际结果、规定截图和已有书面栏目

AI使用情况： 简短说明或一张截图； 未使用填“未使用”

保存后重新打开，核对文本和截图清晰可读

正式提交一份填写完成的Word报告。