

学 院： 信息与软件工程学院

姓 名：李亮 齐秀秀 石硕光 梁容

学 号： 15组

指导教师： 林迪

目录

[HadoopMapReduce及Spark词频统计实验报告 3](#_Toc24715)

[1 实验目的 3](#_Toc24716)

[2 环境部署 3](#_Toc24717)

[2.1 安装部署Hadoop 完全分布式Hadoop集群搭建 3](#_Toc24718)

[2.1.1 Hadoop部署准备 3](#_Toc24719)

[2.1.2 建立hadoop运行账号 4](#_Toc24720)

[2.1.3 配置ssh免密码连入验证 5](#_Toc24721)

[2.1.4 JDK、Hadoop安装并配置环境变量 6](#_Toc24722)

[2.1.5 格式化namenode并启动集群 8](#_Toc24723)

[2.1.6 浏览器可视化管理Hadoop集群 9](#_Toc24724)

[2.2 配置HDFS与MapReduce 10](#_Toc24725)

[2.2.1 HDFS与MapReduce的设计思路 10](#_Toc24726)

[2.2.2 HDFS分布式配置 11](#_Toc24727)

[2.3 安装部署Spark-2.2.0 11](#_Toc24728)

[2.3.1 Spark安装之前的准备 11](#_Toc24729)

[2.3.2 配置环境变量及Spark环境 12](#_Toc24730)

[2.3.3 修改spark-env.sh文件 12](#_Toc24731)

[2.3.4 同步Slave1和Slave2的配置 13](#_Toc24732)

[2.3.5 启动Spark集群 13](#_Toc24733)

[3 实验步骤 14](#_Toc24734)

[3.1 案例描述 14](#_Toc24735)

[3.2 案例分析 15](#_Toc24736)

[3.2.1 MapReduce思想 15](#_Toc24737)

[3.2.2 Hadoop MapReduce 15](#_Toc24738)

[3.2.3 Spark 18](#_Toc24739)

[3.3 具体步骤 19](#_Toc24740)

[3.3.1 测试Hadoop集群 20](#_Toc24741)

[3.3.2 运行Hadoop MapReduce 20](#_Toc24742)

[3.3.3 运行Spark-shell 21](#_Toc24743)

[4 实验结果 21](#_Toc24744)

[4.1 Mapreduce输出结果 22](#_Toc24745)

[4.1.1 时间消耗及文件大小 22](#_Toc24746)

[4.1.2 资源分配 25](#_Toc24747)

[4.1.3 MapReduce过程 25](#_Toc24748)

[4.2 Spark输出结果 26](#_Toc24749)

[4.2.1 Spark运行总耗时 26](#_Toc24750)

[4.2.2 Spark参数调优 27](#_Toc24751)

[4.3 从软件体系架构角度，解释分析实验结果 30](#_Toc24752)

[4.3.1 HadoopMapReduce与Spark优缺点比较 30](#_Toc24753)

[4.3.2 实验结果分析 31](#_Toc24754)

[5 实验问题 34](#_Toc24755)

# **HadoopMapReduce**及**Spark**词频统计实验报告

# 实验目的

1. Ubuntu16.04 安装 Hadoop/HDFS/MapReduce/Spark，熟悉分布式集成开发环境及并行计算框架，了解大数据整个软件体系框架；
2. 理解 MapReduce 思想，分别以 1K、1M、10M、100M、300M 英文文本文件为 WordCount 词频统计数据集，使用 Hadoop MapReduce、Spark 框架进行

WordCount 词频统计实验，并比较 WordCount 词频统计在 MapReduce 及

Spark 执行效率异同；

1. 综合运用所学知识，从软件体系架构角度，解释分析实验结果，编写相关实验报告；
2. 搭建环境：

三台Ubuntu16.04版本的系统主机，进行完全分布式hadoop集群搭建内存：16G 显存：8G

1. 小组人员任务分配：

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 | 任务 |
| 李亮 | 201821090121 | 环境搭建，WordCount 实验 |
| 齐秀秀 | 201852090618 | 实验报告，WordCount 实验 |
| 石硕光 | 201852090615 | PPT，WordCount 实验 |
| 梁容 | 201852090623 | 资料收集，WordCount 实验 |

# 环境部署

## 安装部署Hadoop 完全分布式Hadoop集群搭建

### Hadoop部署准备

Hadoop是一个能够对大量数据进行分布式处理的软件框架，实现了Google的 MapReduce编程模型和框架，能够把应用程序分割成许多的小的工作单元，并把这些单元放到任何集群节点上执行。在MapReduce中，一个准备提交执行的应用程序称为“作业（job）”，而从一个作业划分出得、运行于各个计算节点的工作单元称为“任务（task）”。此外，Hadoop提供的分布式文件系统（HDFS）主要负责各个节点的数据存储，并实现了高吞吐率的数据读写。

Hadoop使用（Master/Slave）主从架构进行分布式储存和分布式计算。Master 负责分配和管理任务，Slave负责实际执行任务。

在分布式存储和分布式计算方面，Hadoop都是用从/从(Master/Slave)架构。在一个配置完整的集群上，想让Hadoop这头大象奔跑起来，需要在集群中运行一系列

后台(deamon)程序。不同的后台程序扮演不用的角色，这些角色由NameNode、

DataNode、SecondaryNameNode、JobTracker、TaskTracker组成。其中NameNode、 Secondary NameNode、JobTracker运行在Master节点上，而在每个Slave节点上，部署一个DataNode和TaskTracker，以便这个Slave服务器运行的数据处理程序能尽可能直接处理本机的数据。对Master节点需要特别说明的是，在小集群中，

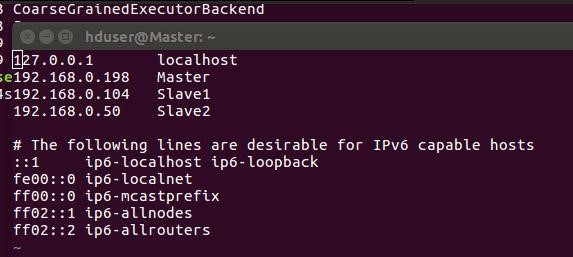
SecondaryNameNode可以属于某个从节点；在大型集群中，NameNode和JobTracker 被分别部署在两台服务器上。

准备三台Ubuntu客户机，并使用vim /etc/hostname命令分别对主机名进行重命名，来区分一个主节点和两个从节点：其中一台主机Master，两台节点Slave1、

Slave2。修改etc/hosts文件，使三台客机之间能够ping通。



将三台客机的ip地址和主机名加在里面，依次在三台客机的命令行中输入sudo vim /etc/hosts，修改内容如图所示：



完成修改后，用ping的命令测试三台客机是否可以相互ping通。

### 建立hadoop运行账号

建立hadoop运行账号，就是建立一个group组，然后在三台客户机上重新建立

新的用户，将这三个用户都加入到这个group中。以下操作三台客机都要进行相同操作：

首先添加一个叫hadoop用户组进来：

sudo groupadd hadoop

添加名叫hduser的用户，并添加到hadoop组中：

sudo user add -s /bin/bash -d /home/hduser -m hduser -g hadoop

重复输入密码，并赋予hduser用户admin权限：

sudo password hduser，sudo adduser hduser sudo

切换到刚刚新建的用户进行之后的操作：

su hduser

### 配置ssh免密码连入验证

开始配置ssh之前，先确保三台机器都装了ssh。输入dpkg --list|grep ssh

查看安装的ssh。若没有安装则使用sudo apt-get install ssh安装ssh。

首先在Master机上输入以下命令，生成Master机的一对公钥和私钥--id\_rsa 和 id\_rsa.pub： ssh-kengen -t rsa -P ‘’ -f ~/.ssh/id\_rsa

将公钥加入到已认证的key中：

cat ~/.ssh/id\_dsa.pub >> ~/.ssh/authorized\_keys

然后输入sshlocalhost 登录本机命令，第一次提示输入密码，输入exit退出，

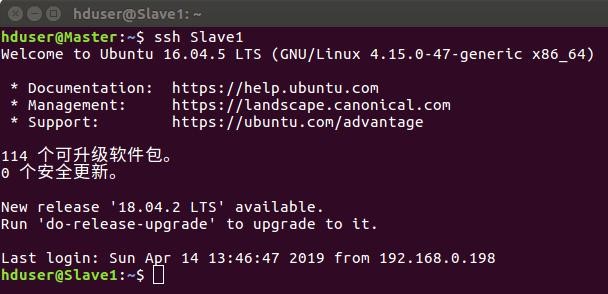
再次输入ssh localhost不用输入密码就可以登录本机成功，则本机ssh免密码登录已经成功。

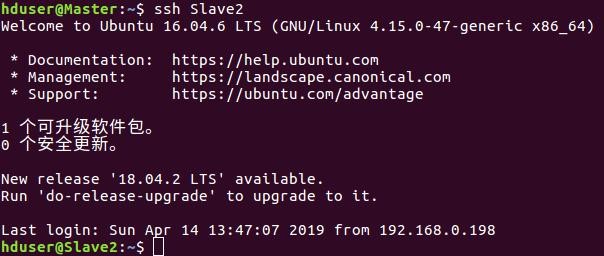
配置Slave1和Slave2节点的ssh免密码登录，目的是让Master主机可以不用密码登录到Slave1和Slave2主机。将Master主机上的is\_dsa.pub复制到Slave1 主机上，命名为Slave1\_dsa.pub。Slave2主机进行同样的操作：

scp hduser@master:~/.ssh/is\_dsa.pub ~/.ssh/Slave1\_dsa.pub

将从master得到的密钥加入到认证，Slave2主机进行同样的操作：

命令如下：cat ~/.ssh/Slave1\_dsa.pub >> ~/.ssh/aurhorized\_keys 然后验证是否已经可以进行ssh免密码登录。在Master机上进行验证同样第一次需要密码，之后exit退出，再ssh Slave1 就不需要密码登录成功，说明ssh免密码登录配置成功。





### JDK、Hadoop安装并配置环境变量

在hduser用户的根目录下创建文件夹local，jdk和hadoop都安装在这个文件

夹下。配置环境，修改 /etc/hadoop目录下的几个文件及 /etc/profile 三台机都要进行5个文件修改，文件名如下：

~/etc/hadoop/hadoop-env.sh

~/etc/hadoop/core-site.xml

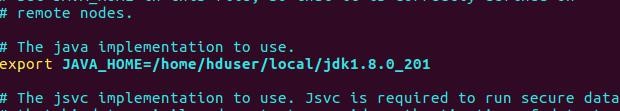
~/etc/hadoop/hdfs-site.xml

~/etc/hadoop/mapred-site.xml

~/etc/hadoop/slaves

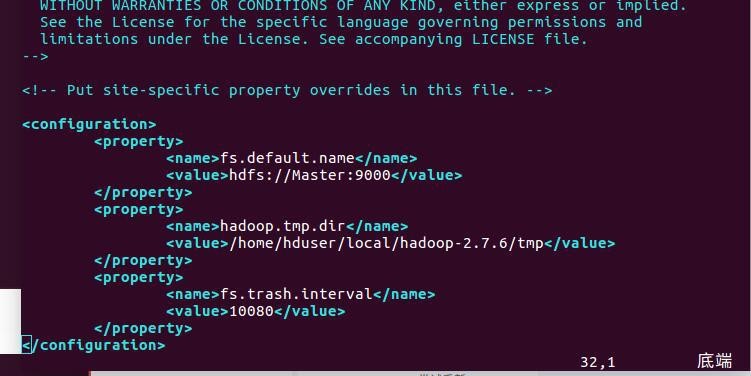
/etc/profile

首先是hadoop-env.sh ,添加java安装的地址，保存退出即可：



然后core-site.cml文件修改：

第一个fs.default.name设置Master机为namenode；第二个hadoop.tmp.dir配置Hadoop的一个临时目录，用来存放每次运行的作业jpb的信息。

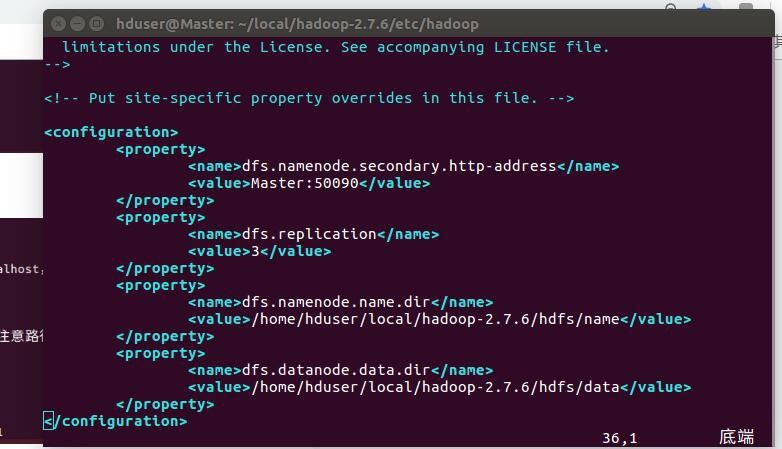


接下来hdfs-site.xml的修改：

dfs.name.dir是namenode存储永久性的元数据的目录列表。这个目录会创建在

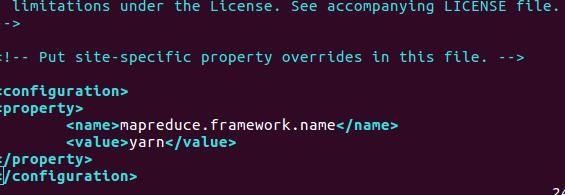
master机上； dfs.data.dir是datanode存放数据块的目录列表，这个目录在node1和node2

机都会创建； dfs.replication 设置文件副本数，这里两个datanode，所以设置副本数为2。

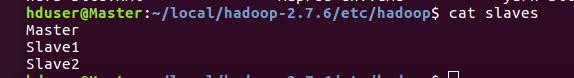


mapred-site.xml的修改：

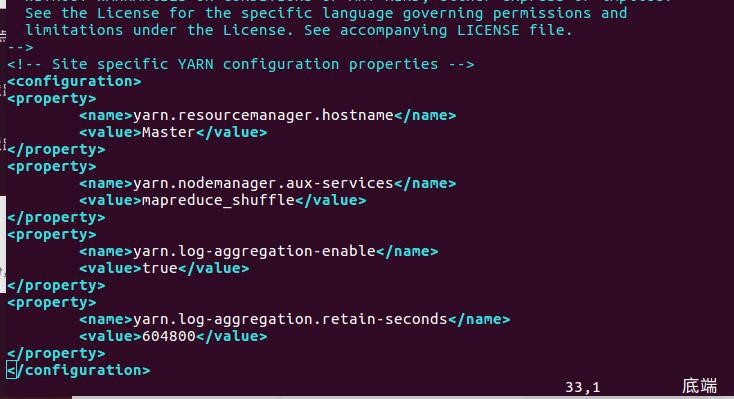
本文件设置的是运行jobtracker的服务器主机名和端口，也就是作业将在 Master主机的9001端口执行。



修改slaves文件，加入两台从主机的主机名Slave1和Slave2：



yarn-site.xml的修改：



最后修改profile文件 ，如下进入profile,将这几个路径添加到末尾：



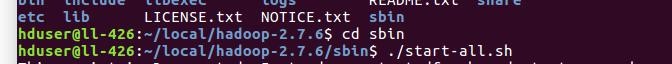
修改完让它生效source/etc/prifile，检查下是否可以看到hadoop版本信息。

### 格式化namenode并启动集群

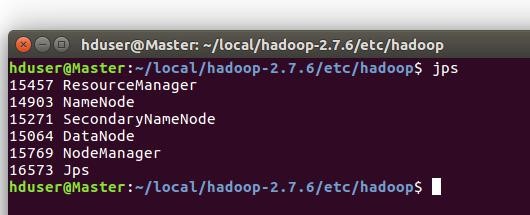
只需要在 master主机上进行格式化，successful表示格式化成功：

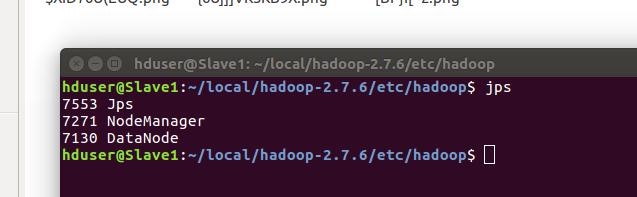


启动集群：



启动完毕，检查下启动情况：Master主机看到四个开启的进程，Slave1和Slave2 看到三个开启的进程表示启动成功。

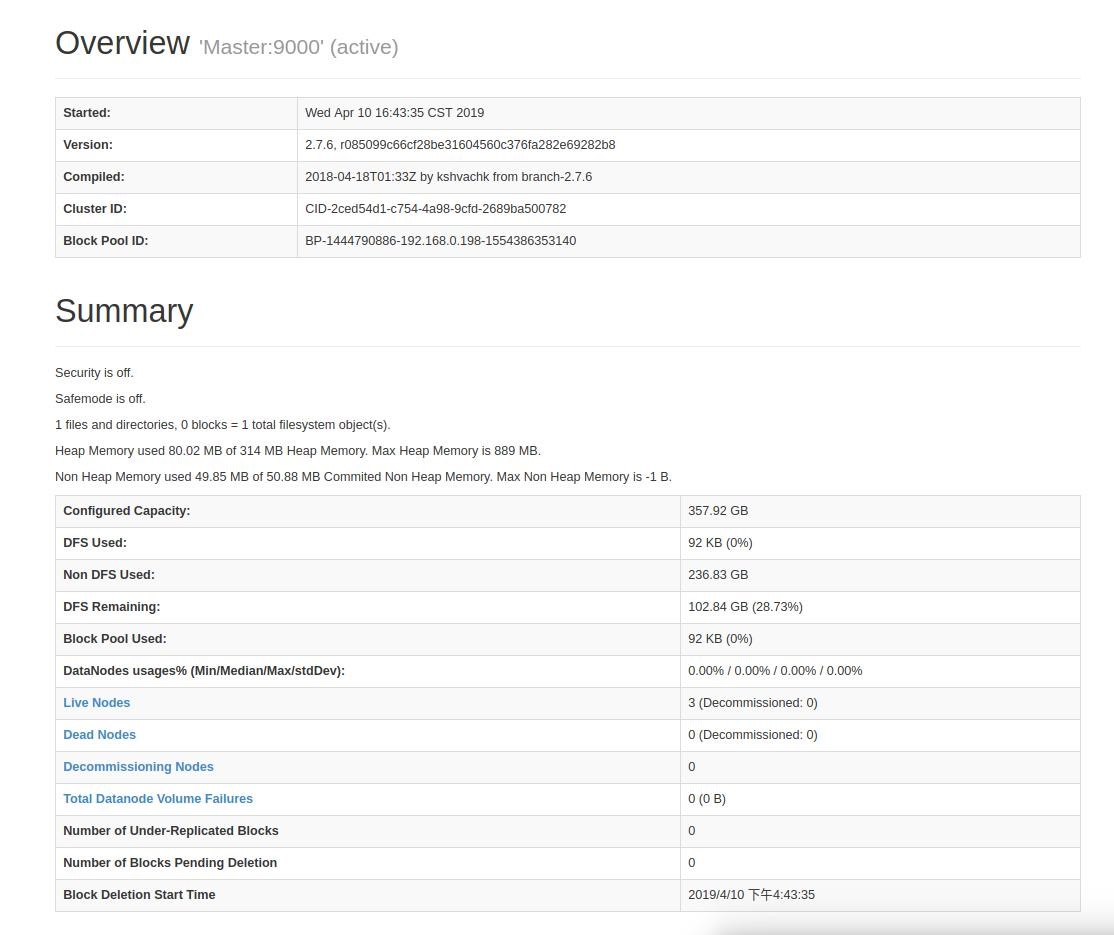
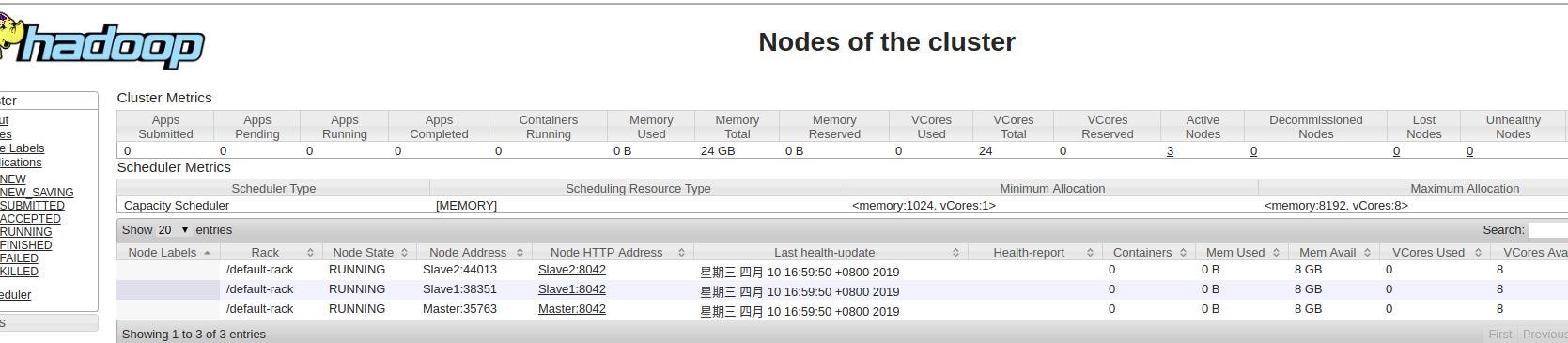




### 浏览器可视化管理Hadoop集群

在主节点Master的浏览器中分别输入192.168.0.198:8088和

192.168.0.198:50090，IP地址为主节点客机Master的IP地址。如图所示：



两个界面都能正常显示节点信息，说明Hadoop的分布式集群环境已经基本安装好了。

## 配置HDFS与MapReduce

### HDFS与MapReduce的设计思路

HDFS即Hadoop分布式文件系统，它擅长存储大文件，流式读取，运行于一般性的商业硬件上。HDFS不适合存储大量的小文件，namenode会在内存中存储元数据，通常情况下每个文件、目录和块都将占用150个字节；也不适合任意并发写的场景，

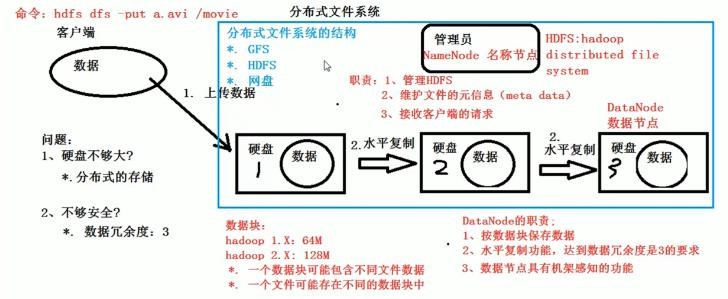
HDFS的写文件操作是append的模式。

除此之外，HDFS特别的地方是，加强了存储文件的安全性和完整性。namenode 存在副本节点（standbynode），当主节点出现问题，比如所运行的机器出现宕机，将由standby节点来取代并获得datanode的管理权限；而且每一段数据将备份在几个不同的datanode上面（根据集群的datanode数量来决定备份个数）。因此，对

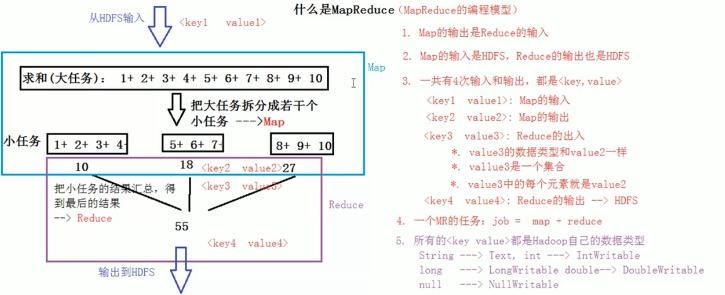
应的将会有一个存储映射表，来记录每一段数据分别存储在哪些节点中，便于 namenode进行管理和查询。

|  |  |  |
| --- | --- | --- |
|  | **HDFS** | **MapReduce** |
| 结构 | 数据存取 - HDFS架构：  Master / Slave(主从结构) - 节点可以理解为物理机器。主节点，只有一个: Namenode；从节点，有很多个: Datanodes。将文件进行切块处理，再通过文件信息服务器Namenode存放切块的文件信息存放地址，实际存放数据的服务器Datanode存放切块后的数据。  系统默认：每个片块大小为64M，以保证寻址速度；数据会写入3个Datanode中，以保证更高的容错性。  HDFS还设计了Secondary Namenode来更新  Namenode，以避免日志文件过大。  HDFS Client帮助Namenode对写入读取数据进行预处理，进行文件的分块与发送读取操作。  Namenode负责为数据任务寻址。 | 数据运算 - MapReduce架构（主从结  构）：  主节点，只有一个: JobTracker；从节点，有很多个:TaskTrackers。通过JobClient 生成任务运行文件，并在JobTracker进行调度指派TaskTracker完成任务。  JobTracker分为把任务文件进行分解  并派送到TaskTracker的程序JobinProgress 和执行调度器的TaskScheduler。  JobinProgress把作业分解成Map计算和Reduce计算并放置到TaskTracker服务器中。 |
| 组件介绍 | HDFSClient：进行文件的分块与文件的发送读取。  Namespaceimage：记录每个文件的存在位置信息。  Edit log：记录每个文件的位置移动信息。  Namenode(Master)：管理着每个文件中各个块所在的数据节点的位置信息。  Secondary Namenode：更新并备份Namenode。  Datanode(Slave)：记录着服务器内所储存的数据块的列表。 | JobClient：用于把用户的作业任务生成 Job的运行包，并存放到HDFS中。  JobinProgress：把Job运行包分解成 MapTask和ReduceTask并存放于 TaskTracker中。  JobTracker(Master)：进行调度管理 TaskTracker执行任务。  TaskTracker(Slave)：执行分配下来的 Map计算或Reduce计算任务。 |
|  | Secondary Namenode：更新并备份Namenode |  |

**HDFS** 工作原理：



**MapReduce** 工作原理：



### HDFS分布式配置

使用MapReduce时，与HDFS建立连接，及使用HDFS中的文件，还需要做一些

配置。

首先进入Hadoop的安装目录：cd /usr/local/hadoop/hadoop

在HDFS中创建用户目录：./bin/hdfs dfs -mkdir -p /user/hadoop

创建input目录，并将./etc/hadoop中的xml文件复制到分布式文件系统中：

./bin/hdfs dfs -mkdir input

./bin/hdfs dfs -put ./etc/Hadoop/\*.xml input

复制完成后，可以使用下面命令查看文件列表：

./bin/hdfs dfs -ls input

## 安装部署Spark-2.2.0

Spark是一种用于Hadoop数据的快速通用计算引擎。Spark提供一种支持广泛应用的简单而易懂的编程模型，包括ETL（ Extract-Transform-Load）、机器学习、流处理以及图计算。

### Spark安装之前的准备

下载Spark2.2.0，文件的解压并改名，

tar -zxvf spark-2.2.0-bin-hadoop2.7.tgz rm -rf spark-2.2.0-bin-hadoop2.7.tgz mv spark-2.2.0-bin-hadoop2.7 spark-2.2.0

### 配置环境变量及Spark环境

配置环境变量，执行命令vi /etc/profile，在最尾巴加入：



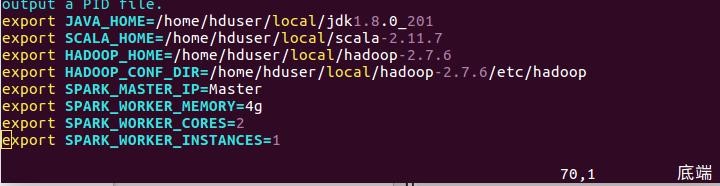
配置Spark环境，打开spark-2.2.0文件夹cd spark-2.2.0。此处需要配置的

文件为两个 spark-env.sh和slaves。

首先我们把缓存的文件spark-env.sh.template改为spark识别的文件 spark-env.sh cp conf/spark-env.sh.template conf /spark-env.sh

### 修改spark-env.sh文件

执行命令vi conf/spark-env.sh，在最尾巴加入：



变量说明：

JAVA\_HOME：Java安装目录

SCALA\_HOME：Scala安装目录

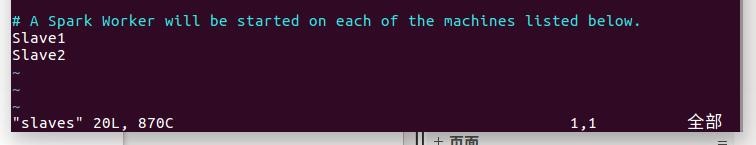
HADOOP\_HOME：hadoop安装目录

HADOOP\_CONF\_DIR：hadoop集群的配置文件的目录

SPARK\_MASTER\_IP：spark集群的Master节点的ip地址

SPARK\_WORKER\_MEMORY：每个worker节点能够最大分配给exectors的内存大小

SPARK\_WORKER\_CORES：每个worker节点所占有的CPU核数目 SPARK\_WORKER\_INSTANCES：每台机器上开启的worker节点的数目修改slaves文件vi conf/slaves，并将最后修改为：



### 同步Slave1和Slave2的配置

使用rsync命令：

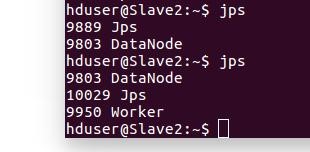
rsync-av/usr/local/spark-2.2.0/SparkWorker1:/usr/local/spark-2.2.0/ rsync-av/usr/local/spark-2.2.0/SparkWorker2:/usr/local/spark-2.2.0/

### 启动Spark集群

启动Spark前需要使用hadoop的HDFS文件系统，所以启动hadoop的HDFS文

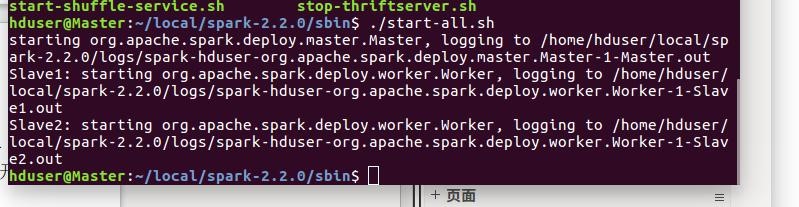
件系统start-dfs.sh。

启动之后使用jps命令可以查看到Master已经启动了namenode，Slave1和 Slave2都启动了datanode，说明hadoop的HDFS文件系统已经启动了。



启动Spark，因为hadoop/sbin以及spark/sbin均配置到了系统的环境中，它

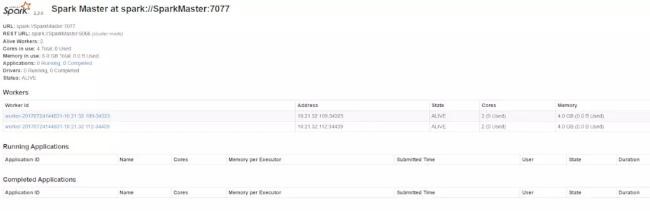
们同一个文件夹下存在同样的start-all.sh文件。最好是打开spark-2.2.0，在文件夹下面打开该文件./sbin/start-all.sh。



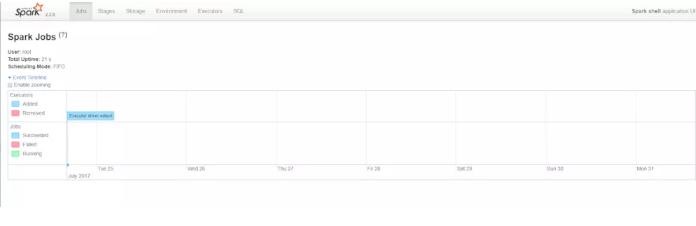
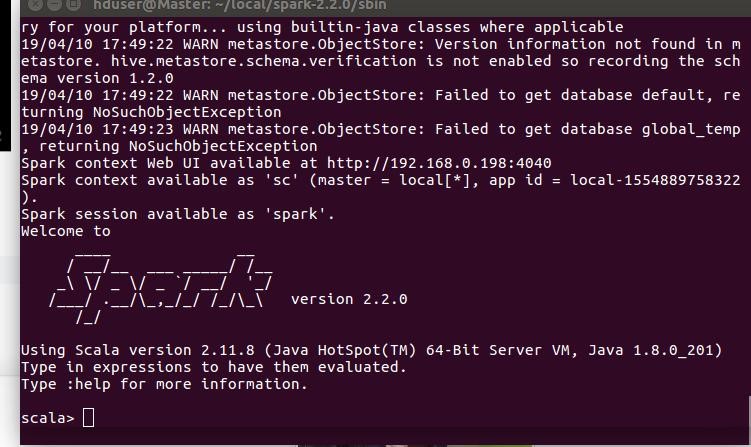
成功打开之后使用jps在Master、Slave1和Slave2节点上分别可以看到新开

启的Master和Slave进程。

成功打开Spark集群之后可以进入Spark的WebUI界面，可以通过 Master\_IP:8080访问，可见有两个正在运行的Slave节点。



使用spark-shell便可打开Spark的shell。同时，因为shell在运行，我们也可以通过Spark的Master\_IP:4040访问WebUI查看当前执行的任务：



# 实验步骤

## 案例描述

单词计数 (WordCount) 是最简单也是最能体现 MapReduce 思想的程序之一。单词计数主要完成功能是：统计各个数据集中每个单词出现的次数。

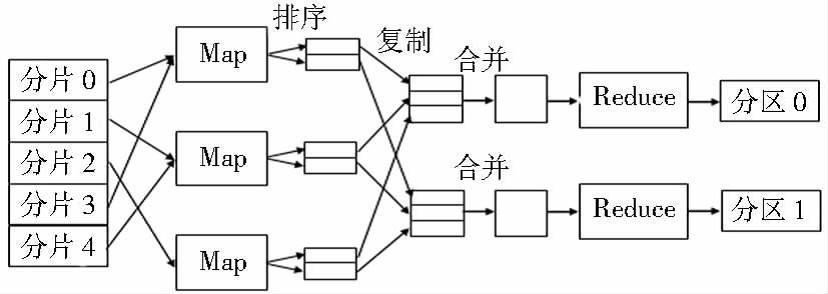
本次实验的实验数据样本为五个1K、1M、10M、100M、300M英文文本文件，使用 Hadoop MapReduce、Spark框架分别运行文本分析程序，基于结果一致的前提下进行统计出运行时间、运行时 CPU 占有率、运行时内存占有率、容错性、通用型等执行效率对比。

## 案例分析

### MapReduce思想

MapReduce是一个分布式运算程序的编程框架。核心功能是用户编写的业务逻辑代码和自带默认组件整合成一个完整的分布式运算程序。一个大数据若可以分为具有同样计算过程的数据块，并且这些数据块之间不存在数据依赖关系，则提高处理速度的最好办法就是采用“分而治之”的策略进行并行化计算。

MapReduce采用了这种“分而治之”的设计思想，即一个Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的Reduce（归约）函数，用来保证所有映射的键值对中的每一个共享相同的键组。对相互间不具有或者有较少数据依赖关系的大数据，用一定的数据划分方法对数据分片，然后将每个数据分片交由一个节点去处理，最后汇总处理结果。一般地，一个典型的 MapReduce 程序的执行流程如下图所示：



**MapReduce** 程序执行流程图

MapReduce 执行过程主要包括：

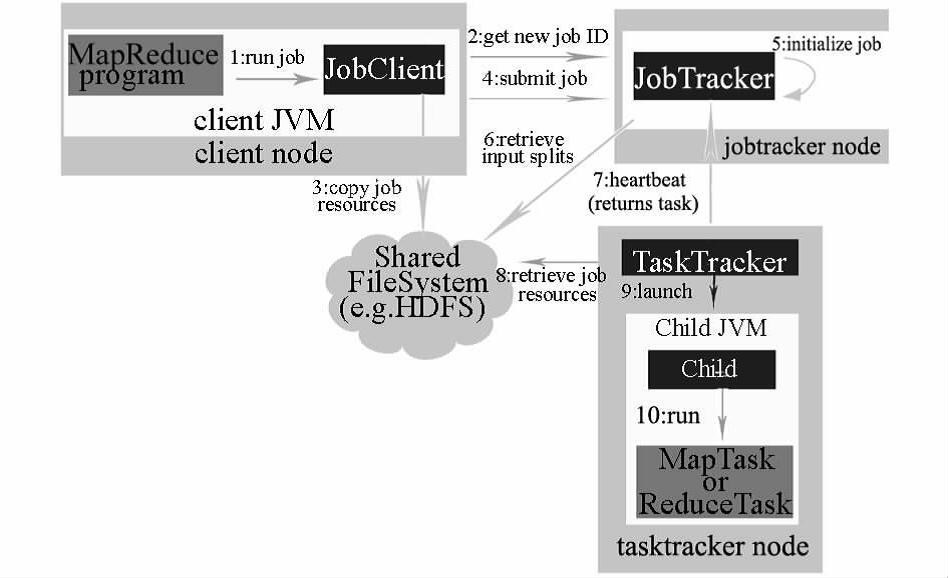
1. 将输入的海量数据切片分给不同的机器处理；
2. 执行 Map 任务的 Worker 将输入数据解析成 key/valuepair，用户定义的 Map 函数把输入的 key/value pair 转成中间形式的 key/value pair；
3. 按照 key 值对中间形式的 key/value 进行排序、聚合；
4. 把不同的 key 值和相应的 value 集分配给不同的机器，完成 Reduce 运算；
5. 输出 Reduce 结果。

任务成功完成后，MapReduce 的输出存放在 R 个输出文件中，一般情况下，这 R 个输出文件不需要合并成一个文件，而是作为另外一个 MapReduce 的输入，或者在另一个可处理多个分割文件的分布式应用中使用。

### Hadoop MapReduce

Hadoop 的设计思路来源于 Google 的 GFS 和 MapReduce。它是一个开源软件框架，通过在集群计算机中使用简单的编程模型，可编写和运行分布式应用程序处理大规模数据。

第一代 Hadoop MapReduce 是一个在计算机集群上分布式处理海量数据集的软件框架，包括一个 JobTracker 和一定数量的 TaskTracker.



**HadoopMapReduce** 系统架构图

在最上层有 4 个独立的实体，即客户端、JobTracker、TaskTracker 和分布式文件系统。客户端提交 MapReduce 作业；JobTracker 协调作业的运行；TaskTracker 运行作业划分后的任务。Hadoop 运行 MapReduce 作业的步骤主要包括提交作业、

初始化作业、分配任务、执行任务、更新进度和状态、完成作业等 6 个步骤。

MapReduce 组成部分：

ResourceManager（RM）包含两个主要的组件：定时调用器(Scheduler)以及应用管理器(ApplicationManager)

1. 调度器（Scheduler）：根据容量，队列等限制条件，将系统中的资源分配给各个正在运行的应用。这里的调度器是一个“纯调度器”，因为它不再负责监控或者跟踪应用的执行状态等，此外，他也不负责重新启动因应用执行失败或者硬件故障而产生的失败任务。调度器仅根据各个应用的资源需求进行调度，这是通过抽

象概念“资源容器”完成的，资源容器（Resource Container）将内存，CPU，磁盘，网络等资源封装在一起，从而限定每个任务使用的资源量。总而言之，定时调度器负责向应用程序分配资源，它不做监控以及应用程序的状态跟踪，并且它不保证会重启由于应用程序本身或硬件出错而执行失败的应用程序。

1. 应用管理器（ApplicationsManager，ASM）：ASM 主要负责接收作业，协商获取第一个容器用于执行 AM 和提供重启失败 AM container 的服务。

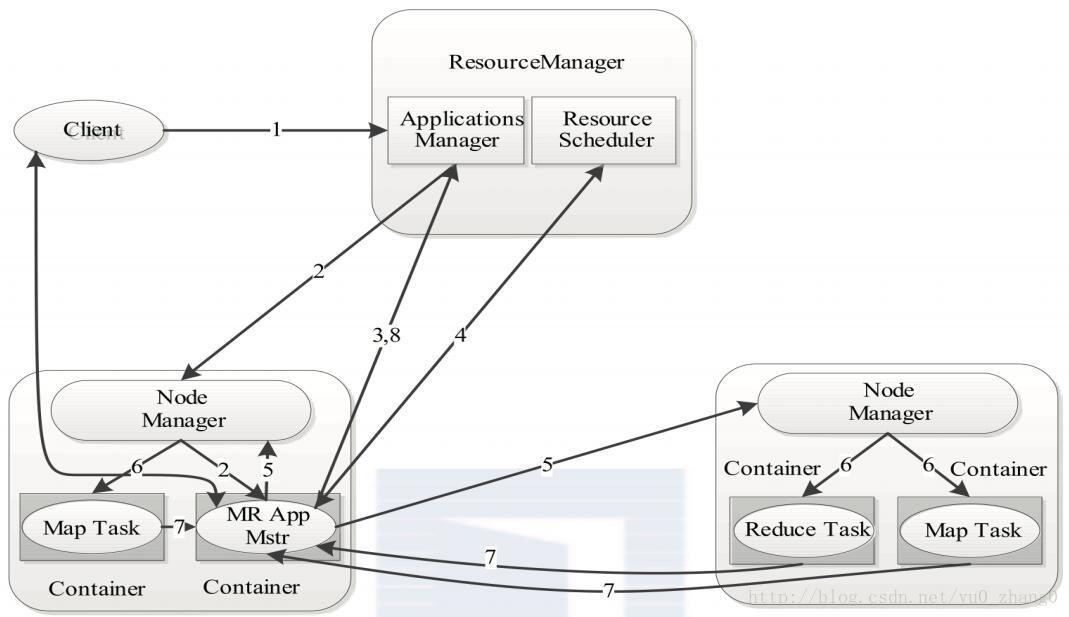
NodeManager：NM 是每个节点上的框架代理，主要负责启动应用所需的容器，

监控资源（内存，CPU，磁盘，网络等）的使用情况并将之汇报给调度器（Scheduler）。

ApplicationMaster：每个应用程序的 ApplicationMaster 负责从 Scheduler 申请资源，以及跟踪这些资源的使用情况以及任务进度的监控。

Container：是 YARN 中资源的抽象，它将内存、CPU、磁盘、网络等资源封装在一起。当 AM 向 RM 申请资源时，RM 为 AM 返回的资源便是用 Container 表示的。

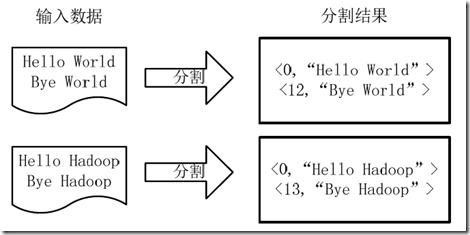
以下是 mapreduce 的结构图：



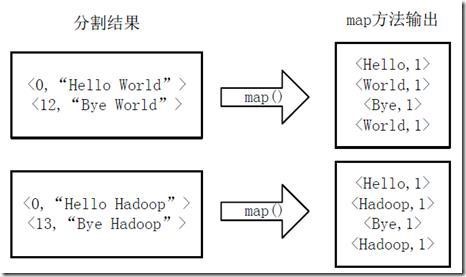
map阶段：

1. 读取HDFS文件，将文件拆分成 splits，并解析每一行成<行偏移量，行内容>

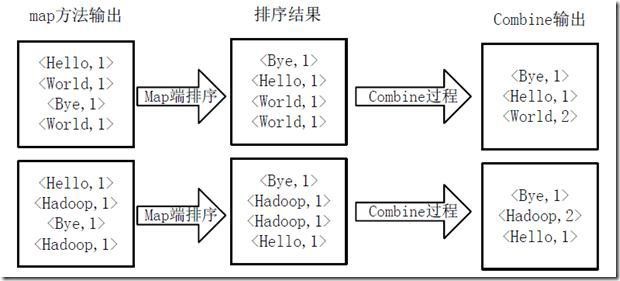
（）形式。



1. 调用map()方法，将分割好的<key,value>对交给用户定义的 map 方法进行处理，生成新的<key,value>对。



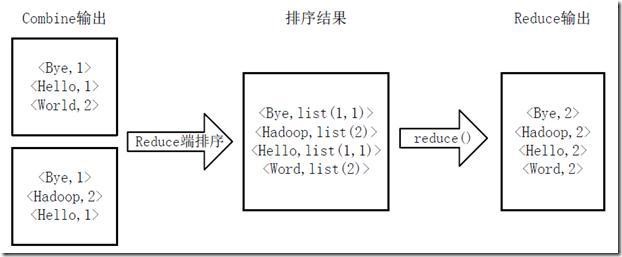
1. 得到 map 方法输出的<key,value>对后，Mapper 会将它们按照 key 值进行排序，并执行 Combine 过程，将 key 相同的 value 值累加，得到 Mapper 的最终输出结果。Map 端排序及 Combine 过程如下图所示：



reduce阶段：

Reducer 先对从 Mapper 接收的数据进行排序，再交由用户自定义的 reduce 方法进行处理，得到新的<key,value>对，并作为 WordCount 的输出结果

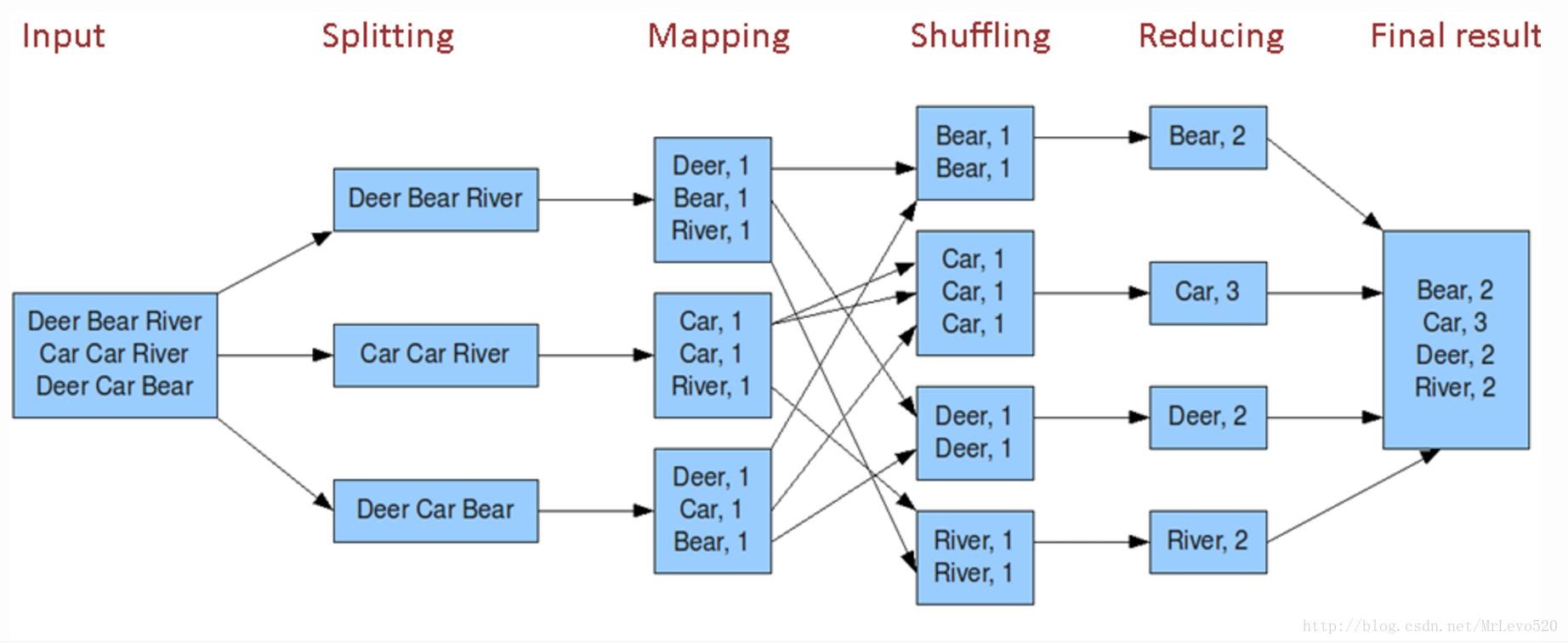
1. reduce拉取数据，也就是Shuffle阶段。
2. reduce的合拼与排序，合拼是指将key对应的value数组进行合拼。
3. reduce输出结果数据。



例如如果要正则过滤出想要的内容，Hadoop 首先把 1PB的数据文件导入到

HDFS中, 然后编程人员定义好 map和reduce, 也就是把文件的行定义为key,每行的内容定义为value , 然后进行正则匹配,匹配成功则把结果 通过reduce聚合起来返回.

Hadoop 就会把这个程序分布到N 个结点去并行的操作. 那么原本可能需要计算好几天,在有了足够多的结点之后就可以把时间缩小到几小时之内,流程图如下：

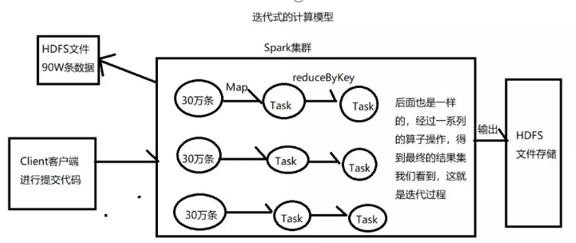


### Spark

Spark 是一个基于内存计算的开源的集群计算系统，目的是让数据分析更加快速。Spark 启用了内存分布数据集，除了能够提供交互式查询外，它还可以优化迭代工作负载。Spark 提供了基于内存的计算集群，在分析数据时将数据导入内存以实现快速查询，“速度比”基于磁盘的系统，如比 Hadoop 快很多。Spark 最初是为了处理迭代算法，如机器学习、图挖掘算法等，以及交互式数据挖掘算法而开发的。在这两种场景下，Spark 的运行速度可以达到 Hadoop 的几百倍。

Spark 与 HadoopMapReduce 的最大区别是它把所有数据保存在内存中，Hadoop MapReduce 需要从外部存储介质里把数据读入到内存，Spark 不需要这一步骤。它的实现原理与 Hadoop MapReduce 没有太大区别。

Spark迭代计算过程：



Spark的计算模型时迭代式的，这个迭代过程是通过算子操作完成的。即Spark中有很多算子操作，可以实现迭代计算的逻辑，然后通过迭代计算，得到最终结果集并存储。假设HDFS上有一个数据文件data.txt文件，需要对其进行WordCount统计计算。

1. val lines = sc.textFile("hdfs://") ，主要功能是加载HDFS中的数据文件进入Spark本地或是集群计算，这里我们使用的是SparkContext的textFile算子，加载后的数据将以每行记录组成元素，元素类型为String。这时假设数据被分为3 个RDD进入Spark集群的不同节点1、2、3。RDD由Hadoop中HDFS数据文件转变为 MappedRDD,这也是创建RDD的一种方式。
2. valwords=lines.flatMap{line=>line.split("")} ，主要是对MappedRDD 中的每个元素(也就是每一行)进行操作。这里使用transformation中的flatMap算子，作用是可以将一个map数据集转变成为flap数据集，即数据扁平化处理。这里也就是将输入文件的每一行数据，按空格("")进行拆分，得到单词数组，再将数组进行扁平化后形成单词字符串，在flatMapRDD中。RDD由MappedRDD转变为 flatMappedRDD,这又是创建RDD的一种方式，由RDD创建RDD。
3. val pairs = words.map { word => (word, 1) }，主要是将第二步的单词数组 flatMapRDD中的数据进行标记，即每个行的格式由单个单词转变成的形式。这里使用的是map算子，Java编写中略有不同，是mapToPair算子。RDD由flatMappedRDD 转变为MapPartitoinsRDD。
4. valwordCounts=pairs.reduceByKey{\_+\_}，主要是将第三步产生的pairs 元素的不同RDD中相同key值拉到一起进行value的归并操作。RDD由flatMappedRDD 转变为MapPartitoinsRDD的时候Hello对应的value变为了2，这是reduceByKey 算子的特性，会在总的归并之前先对每个RDD进行相同可以对应的value执行merge 操作(也就是归并相同可以的value)，这也是在代码性能优化中优先reduceByKey

的原因，而不选择GroupByKey。这时RDD由MapPartitoinsRDD转变为shuffleRDD 又转变成MappedRDD。

1. wordCounts.foreach(wordCount => println(""))，对结果数据进行action操作遍历输出到客户端控制台。这里使用foreach算子，该算子属于action类型，每个job任务必须要有一种叫做action类型的算子才能触发应用程序的执行。

## 具体步骤

### 测试Hadoop集群

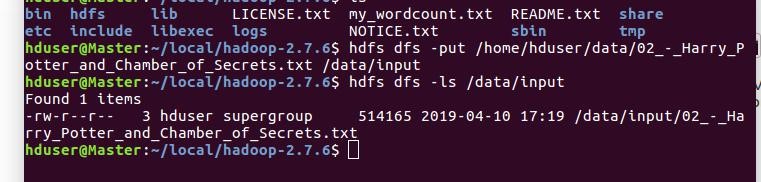
向hadoop集群提交第一MapReduce任务（wordcount）。在主节点Master上，

进入hadoop目录（~/local/hadoop-2.7.6），依次执行下列命令：



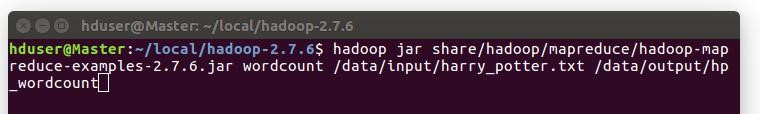


结果如图所示：



### 运行Hadoop MapReduce

向Hadoop提交WordCount任务，命令如下：

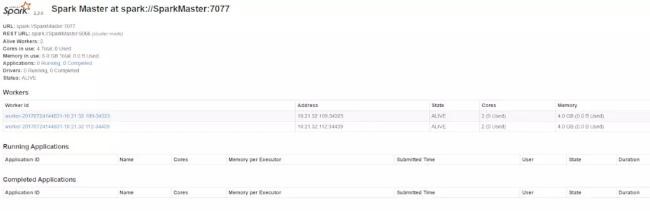


MapReduce输出结果：



### 运行Spark-shell

启动Spark集群，进入Spark的WebUI界面，可以通过Master\_IP:8080访问，可见有两个正在运行的Slave节点。



当启动spark-shell启动以后 , 监控页面会监控到spark-shell，提交

WordCount任务。



spark-shell 是以集群的方式启动的 , 读取分布式文件系统的文件。

spark中的RDD上的方法都称之为算子。分为两类：转换transformation lazy 执行。 当遇到action算子的时候，才开始真正的运行。转换类的算子，都会生成新的rdd。

行动类的算子 action及词频统计结果:



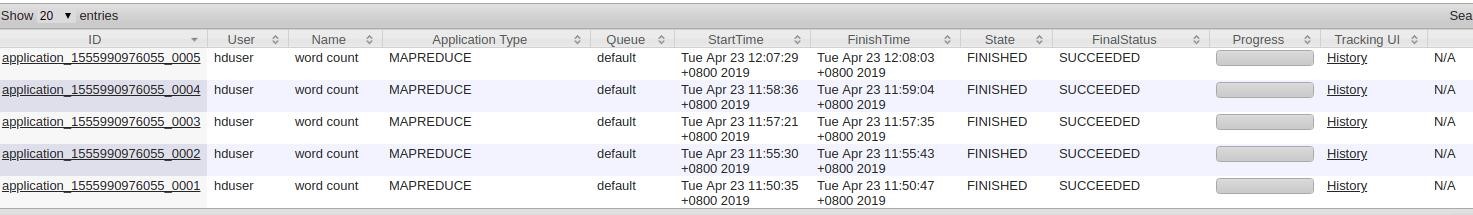
# 实验结果

## Mapreduce输出结果

datanode UI 界面：

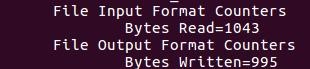
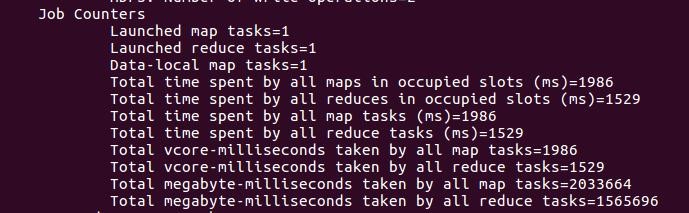


MapReduce UI界面:5个WordCount词频统计任务成功：

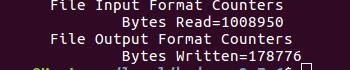
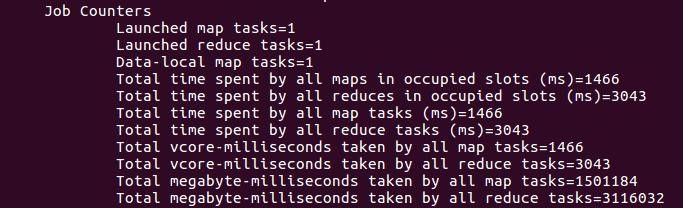


### 时间消耗及文件大小

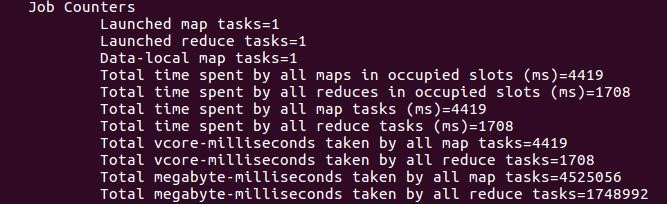
1. 1K.txt 文本执行 Map、Reduce 时间及文件大小输入输出的图：



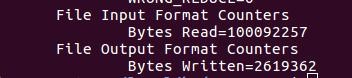
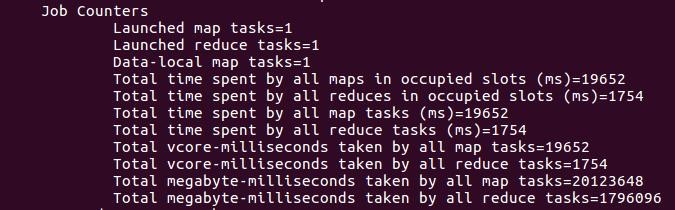
1. 1M.txt执行 Map、Reduce 时间及文件大小输入输出的图：



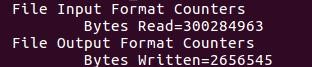
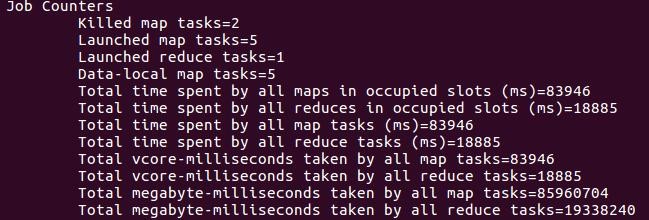
1. 10M.txt执行 Map、Reduce 时间及文件大小输入输出的图：



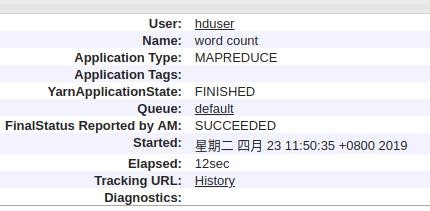
1. 100M.txt执行 Map、Reduce 时间及文件大小输入输出的图：



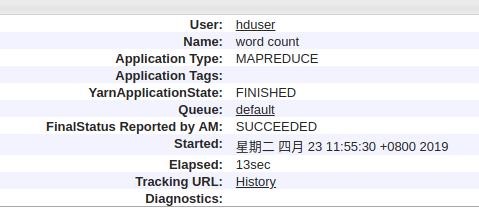
1. 300M.txt执行 Map、Reduce 时间及文件大小输入输出的图：



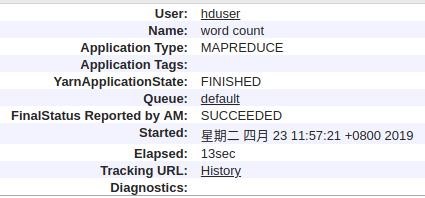
mapreduce UI界面总耗时： 1) 1K.txt 文本执行 MapReduce 总时耗：



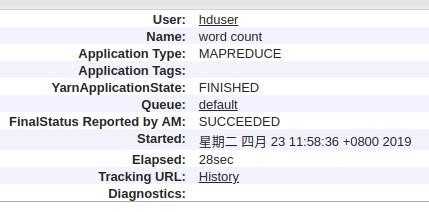
1. 1M.txt 文本执行 MapReduce 总时耗：



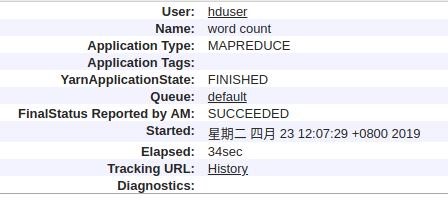
1. 10M.txt 文本执行 MapReduce 总时耗：



1. 100M.txt 文本执行 MapReduce 总时耗：

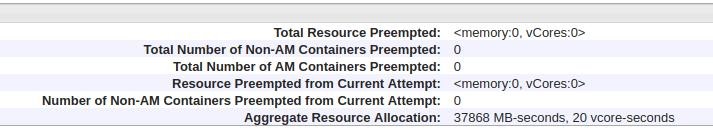


1. 300M.txt 文本执行 MapReduce 总时耗：

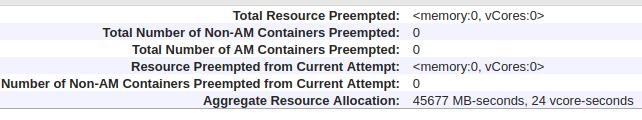


### 资源分配

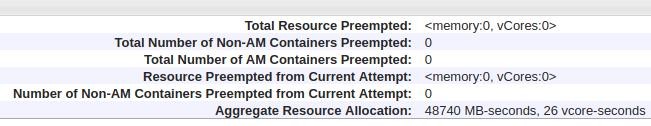
1. 1K.txt 文本执行 MapReduce 资源分配：



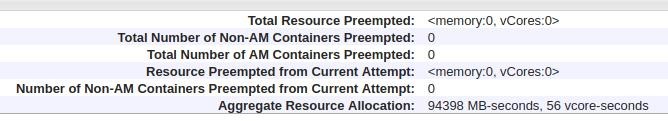
1. 1M.txt 文本执行 MapReduce 资源分配：



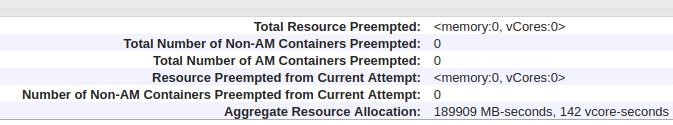
1. 10M.txt 文本执行 MapReduce 资源分配：



1. 100M.txt 文本执行 MapReduce 资源分配：

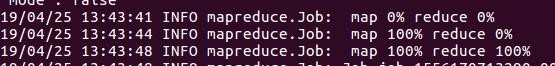


1. 300M.txt 文本执行 MapReduce 资源分配：



### MapReduce过程

1. 1Ktxt 文本执行 MapReduce：



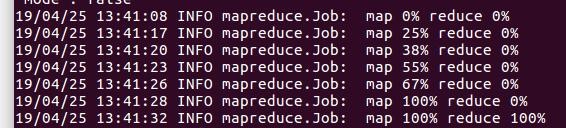
1. 1M.txt 文本执行 MapReduce：



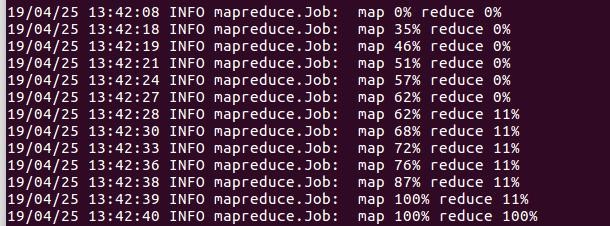
1. 10M.txt 文本执行 MapReduce：



1. 100M.txt 文本执行 MapReduce：



1. 300M.txt 文本执行 MapReduce：



## Spark输出结果

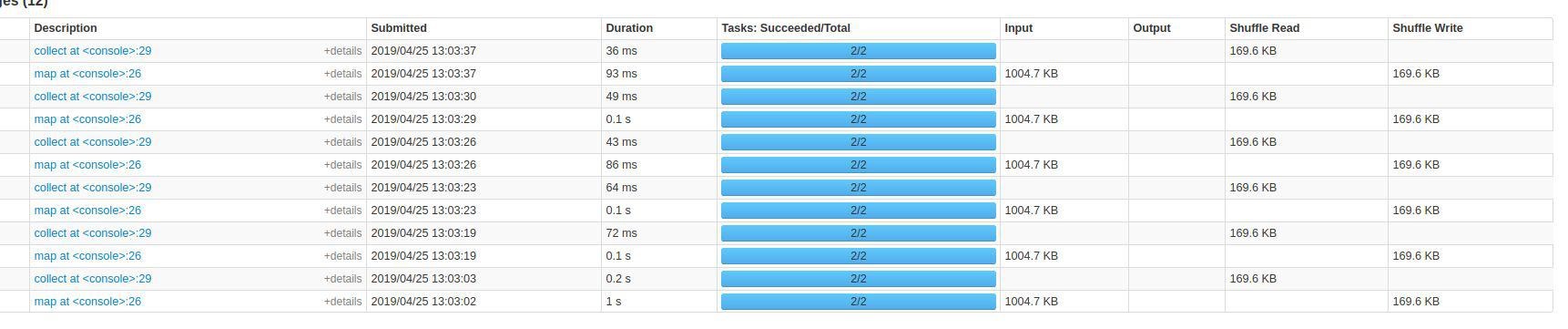
### Spark运行总耗时

通过在 1K、1M、10M、100M、300M 数据集上进行 5 次实验表明，第一次实验准备时间会长于之后的 5 次实验,这是由于实验的第一次都要算加载到内存的时间，所以后面几次的 map 时间比第一次都快。

1. 1K.txt 文本 Spark 执行 Map、Reduce 运行总时耗：



1. 1M.txt 文本 Spark 执行 Map、Reduce 运行总时耗：



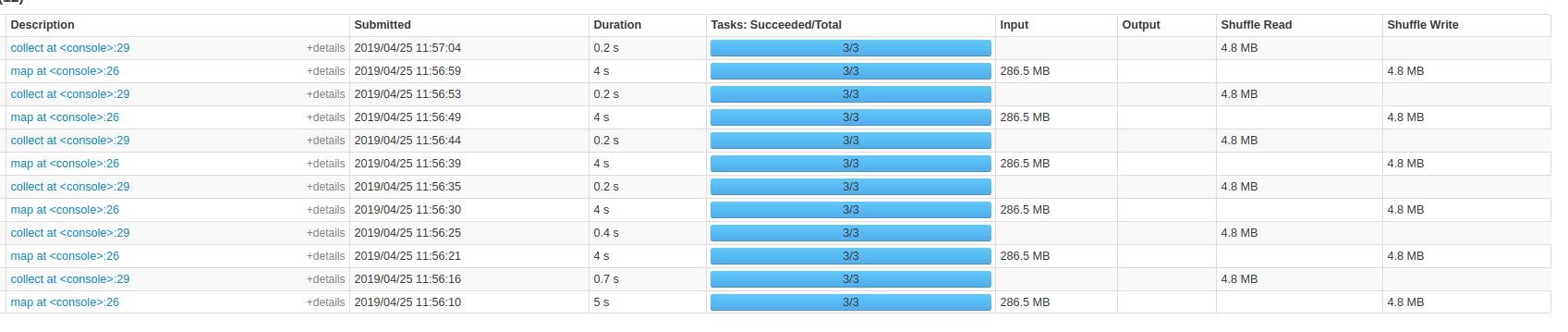
1. 10M.txt 文本 Spark 执行 Map、Reduce 运行总时耗：



1. 100M.txt 文本 Spark 执行 Map、Reduce 运行总时耗：



1. 300M.txt 文本 Spark 执行 Map、Reduce 运行总时耗：



### Spark参数调优

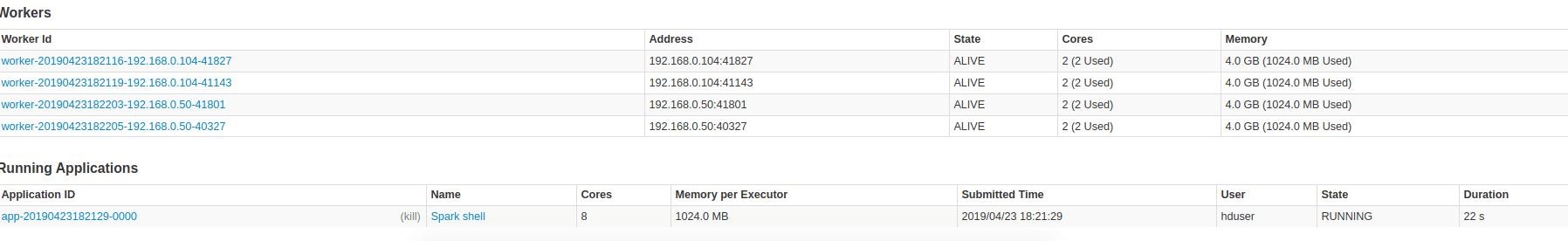
Spark 参数调优实验在 100M 文本数据集下进行，调优参数为：num-executors、 executor-memory、executor-cores。

num-executors调参： num-executors：该参数用于设置 Spark 作业总共要用多少个 Executor 进程来执行。

1. num-executors:2

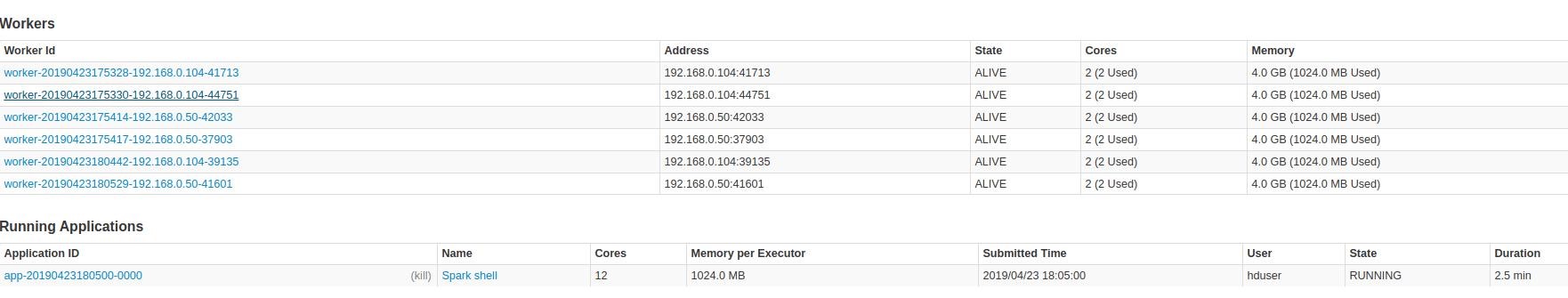


1. num-executors:4

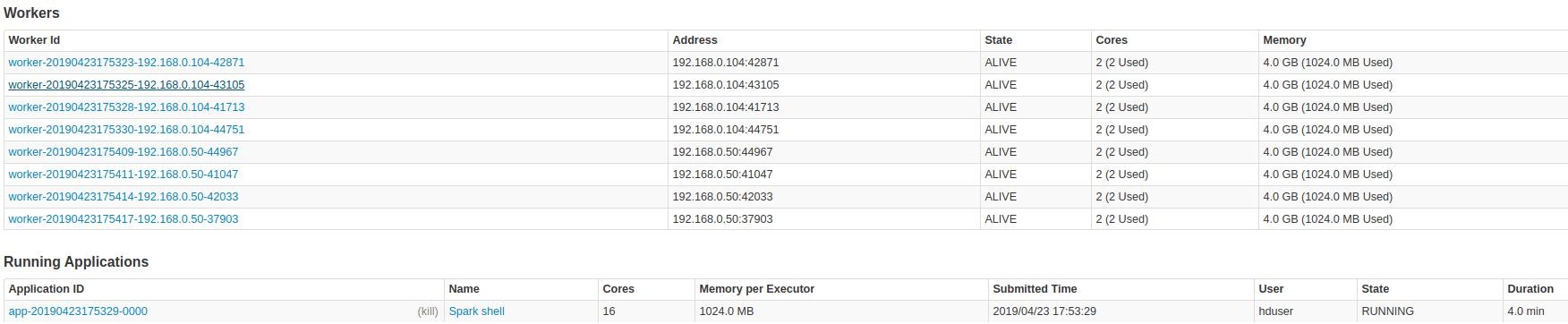


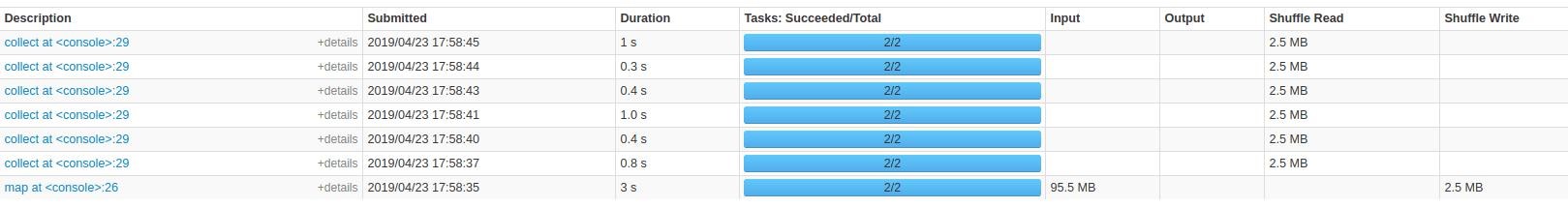


1. num-executors:6



1. num-executors:8





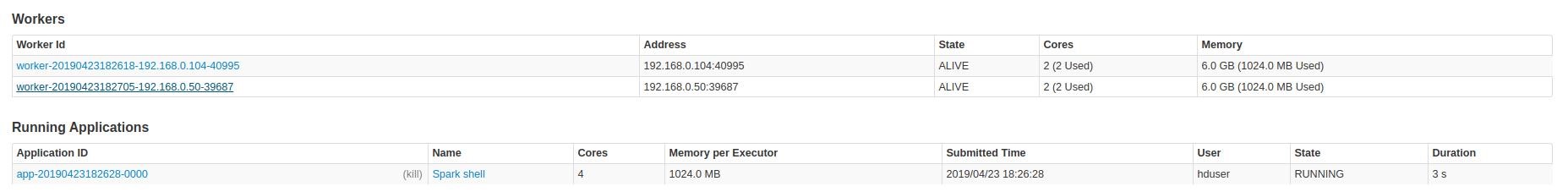
**executor-memory** 调参：

executor-memory：该参数用于设置每个 Executor 进程的内存。Executor 内存的大小，很多时候直接决定了 Spark 作业的性能

1. executor-memory：4G

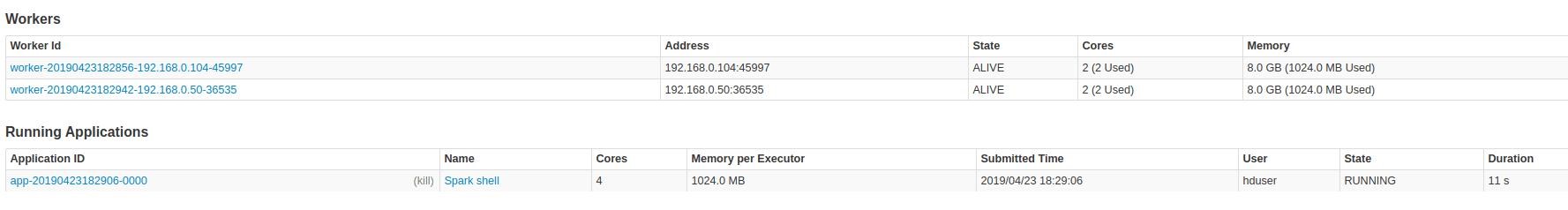


1. executor-memory：6G





executor-memory：8G



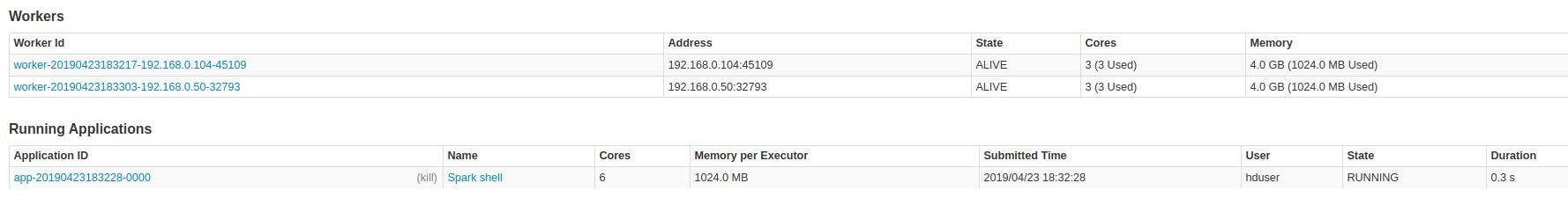
executor-cores调参：

executor-cores:该参数用于设置每个 Executor 进程的 CPU core 数量。这个参数决定了每个 Executor 进程并行执行 task 线程的能力。因为每个 CPU core 同一时间只能执行一个 task 线程，因此每个 Executor 进程的 CPU core 数量越多，越能够快速地执行完分配给自己的所有 task 线程。

1. executor-cores：2

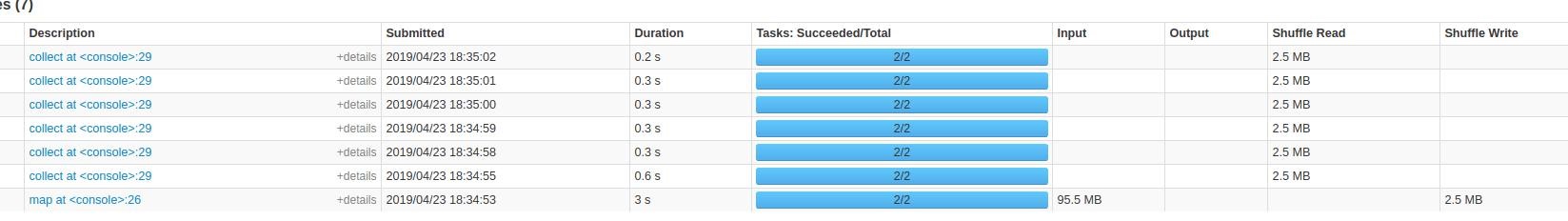
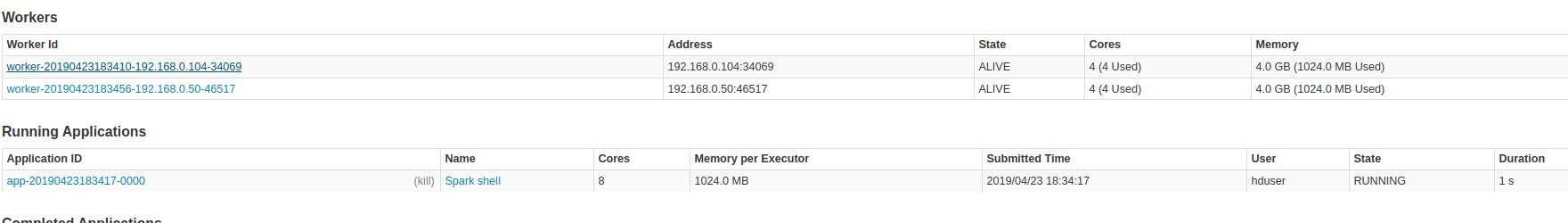


1. executor-cores：3





1. executor-cores：4



## 从软件体系架构角度，解释分析实验结果

### HadoopMapReduce与Spark优缺点比较

|  |  |  |
| --- | --- | --- |
|  | Hadoop MapReduce | Spark |
| 构建平台 | 需要首先架构基于 Hadoop 的集群系统，通过 HDFS 完成运算的数据存储工作 | 可以的单独运行，也可以与 Hadoop 框架完整结合 |
| 功能特点 | 计算能力非常强，适合超大数据集的应用程序，但是由于系统开销等原因，处理小规模数据的速度不一定比串行程序快，并且本身集群的稳定性不高 | 在保证容错的前提下，用内存来承载工作集，内存的存取速度快于磁盘多个数量级，从而可以极大提升性能 |
| 优势与不足 | 高可靠性：因为 Hadoop 假设计算元素和存储会出现故障，因为它维 | 1.只提供两个操作，Map和Reduce，表达力欠缺。 |
|  | 护多个工作数据副本，在出现故障时可以对失败的节点重新分布处理。  高扩展性：在集群间分配任务数据，可方便的扩展数以千计的节点。  高效性：在 MapReduce 的思想下，Hadoop 是并行工作的，以加快任务处理速度。  高容错性：自动保存多份副本数据，并且能够自动将失败的任务重新分配。 | 2.一个Job只有Map和Reduce两个阶段，复杂的计算需要大量的Job完成，Job之间的依赖关系是由开发者自己管理的。  3.中间结果也放在HDFS文件系统中  4.ReduceTask需要等待所有  MapTask都完成后才可以开始延时高，只适用Batch数据处理，对于交互式数据处理，实时数据处理的支持不够 |

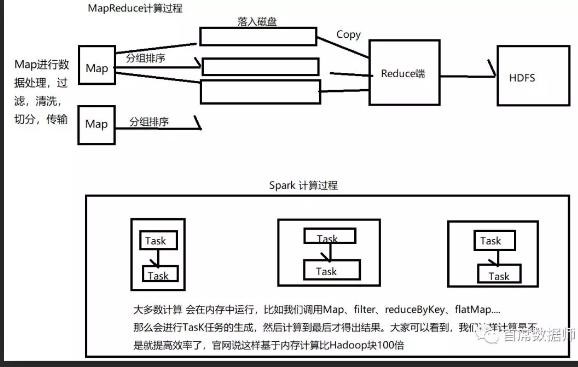
Spark 与 hadoop 比较:

· Hadoop 有两个核心模块，分布式存储模块 HDFS 和分布式计算模块 Mapreduce；

· spark 本身并没有提供分布式文件系统，因此 spark 的分析大多依赖于 Hadoop 的分布式文件系统 HDFS；

· Hadoop 的 Mapreduce 与 spark 都可以进行数据计算，而相比于 Mapreduce，spark 的速度更快并且提供的功能更加丰富；

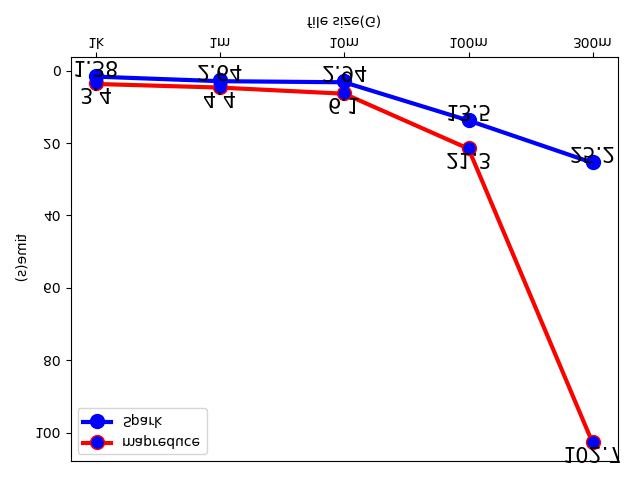
Spark 与 hadoop 关系计算图如下：



### 实验结果分析

**1)** 分析不同大小的文本文件所消耗的时间

在1k，1M，10M，100M，300M文件下，MapReduce和Spark的处理速度比较：



由上节实验结果可得出，HadoopMapReduce 的运行时间最长，原因是 Hadoop 生态环境包含内容过多，所以每次任务启动时首先需要加载所需资源包，然后缓慢地发起任务，所以导致整体计算时间长、性能差。

Spark 框架在 WordCount 实验中消耗的时长稍少。需要时长从高到低分别是

Hadoop MapReduce、Spark。

从实验的计算时间上来看，Spark的计算速度是要高于hadoop的，这是因为

Spark 把中间数据放到内存中，迭代运算效率高。MapReduce 在shuffle的时候，计算结果需要保存到磁盘上，这样势必会影响整体速度，而Spark利用RDD技术，计算在内存中进行.且Spark 支持 DAG 图的分布式并行计算的编程框架，减少了迭代过程中数据的落地，提高了处理效率。（延迟加载）

1. 分析不同大小的文本文件 **CPU** 使用率

由上节实验结果可得出，HadoopMapReduce需要占用的 CPU 资源在 300M 文本处理时基本到达较大饱和度。

Spark 的 CPU 使用率没有完全伴随着文本文件增大而大幅上涨。

1. 分析不同大小的文本文件内存使用率

由上节实验结果可得出，随着文本数据的增大，Spark 随着数据量增大而内存大幅增加，当数据量达到本次测试最大的300M 文本时，Spark 框架对内存的消耗是最小的，Hadoop MapReduce 需要占用较多的内存。

1. 分析容错性

Spark 容错性高。Spark 引进了弹性分布式数据集 RDD (Resilient

DistributedDataset) 的抽象，它是分布在一组节点中的只读对象集合，这些集合是弹性的，如果数据集一部分丢失，则可以根据“血统”（即允许基于数据衍生过程）对它们进行重建。另外在RDD 计算时可以通过 CheckPoint 来实现容错。

1. 分析通用型

Spark 更加通用。mapreduce 只提供了 Map 和 Reduce 两种操作，Spark 提供的数据集操作类型有很多，大致分为：Transformations 和 Actions 两大类。

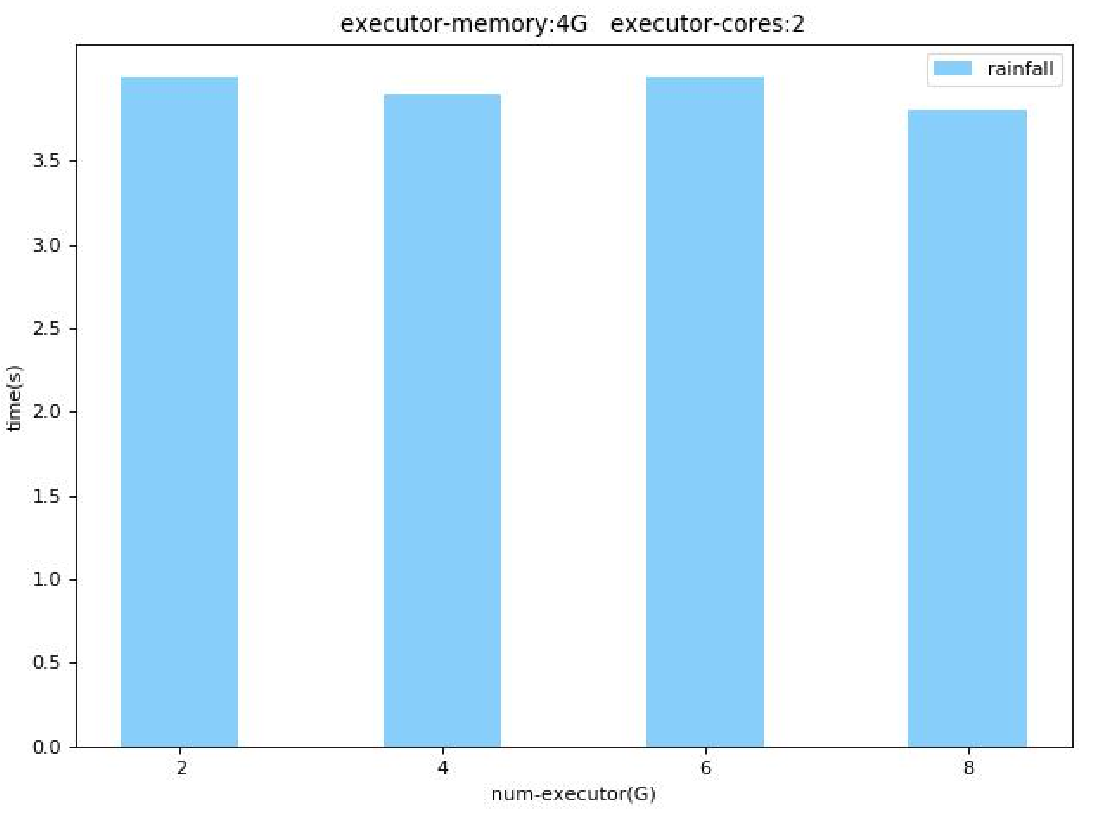
Transformations包括 Map、Filter、FlatMap、Sample、GroupByKey、ReduceByKey、

Union、Join、Cogroup、MapValues、Sort 等多种操作类型，同时还提供 Count,

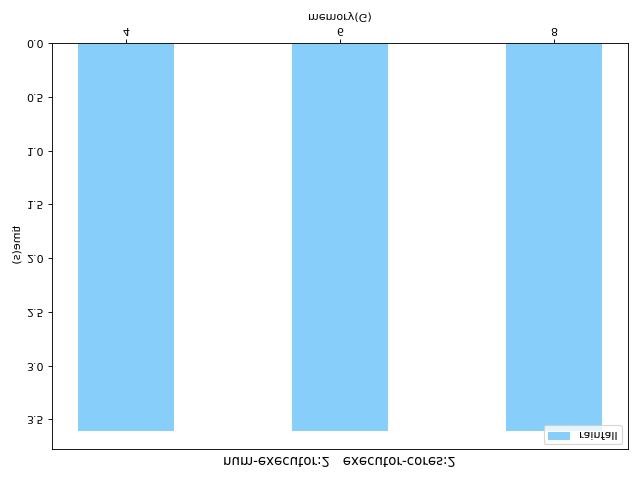
Actions 包括 Collect、Reduce、Lookup 和 Save 等操作。从上面的测试结果我们得出，如果应用程序需要处理的数据量非常大，并且客户希望计算出的数据可以被存储和二次计算或数据挖掘，那 Hadoop MapReduce 较好，因为整个 Hadoop 生态圈庞大，支持性很好。Spark 由于架构层面设计不同，所以对于 CPU、内存的使用率一直保持较低状态，它未来可以用于海量数据分析用途。

1. 分析 **Spark** 参数调优在不同参数下，spark的性能比较,对100M的文本进行处理。

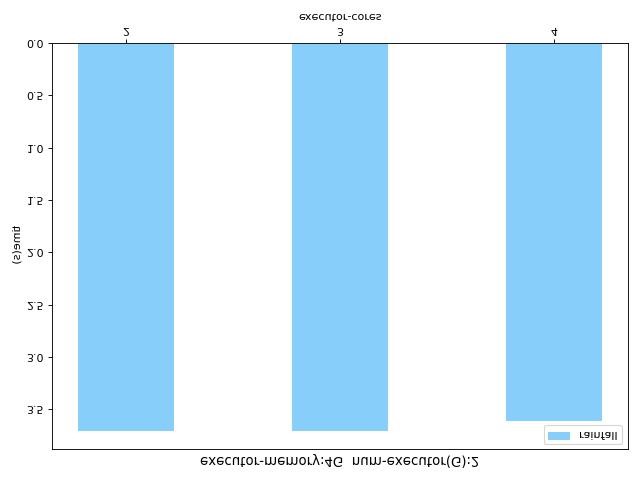
随着executor的增多,程序的处理能力是上升的,计算的时间是减小的：



因为使用的是 100M 的文本,所以在增加主机的内存的时候,对词频的处理速度的变化并不明显：



随着 executor-cores 的增加,用于设置每个 Executor 进程的 CPU core 数量增加,但效果并不明显,主要原因还是因为文本太小,CPU core 数量增加的优势体现不出来：



# 实验问题

1.JAVA\_HOME is not set and cound not be found 原因及解决方法：只修改了主节点Master的JAVA\_HOME路径，忘记修改从节点

的路径（在hadoop-env.sh上修改）。

2.词频统计任务在running job阶段阻塞原因及解决方法：未修改主节点以及从节点hostname，而/etc/hosts文件下的 hostname与IP映射却是Master,Slave1,Slave2，导致触发 UnknownHostException: ll-426。

3.Master节点中Namenode未正常启动或Slave节点的datanode未启动原因及解决方法：多次初始化namenode 导致ID不对应。关闭hadoop，删除data,name,logs,tmp目录下所有文件，然后 hadoop namenode -format 重启初始化，最后开启hadoop。

4.Master的Nodemanager和dataNode未启动

原因及解决方法：把hadoop配置文件中的slaves文件中添加Master 在

/etc/hosts文件中 也要添加IP->Master的映射，并且在yarn-site.xml中

yarn.resourcemanager.hostname和yarn.nodemanager.aux-services也要配置好 Spark Web UI上不显示Running Applications和Completed Applications 百度了一波 MASTER=spark://master:7077 ./spark-shell。

5.Hadoop为了防止数据丢失，启动了“安全模式”的设置，每次启动hadoop后一段时间内集群处于安全模式，该模式下集群会检查各节点文件块的记录，如果数据块中满足replication设置值的数据块的数量在总数据块数量中所占比例没有超过一定值，那么集群将持续处于安全模式，在该模式下，文件系统中的内容不允许修改也不允许删除，直到安全模式结束。安全模式主要是为了系统启动的时候检查各个DataNode上数据块的有效性，同时根据策略必要的复制或者删除部分数据块。运行期通过命令也可以进入 安全模式。在实践过程中，系统启动的时候去修改和删除文件也会有安全模式不允许修改的出错提示，只需要等待一会儿即可。

原因及解决方法：直接在bash输入指令脱离安全模式可以在安全模式下输入指

令即可退出安全模式：

hadoop dfsadmin -safemode leave

6.做调参的实验中 关闭spark的时候 worker出现异常没有被关闭。

原因及解决方法：使用kill -9 ID 强行杀死woker进程

7.数据是否清洗

原因及解决方法：在进行任务前已经预处理过数据，通过去停词stopword把不常用的字符和符号以及单词取出掉了。