HADOOP笔记

# 系统配置

## 系统信息

系统：kali linux2018.4

用户名：root/123、azi/…

切换至字符界面：sudo systemctl set-default multi-user.target

或者是：ln -sf /lib/systemd/system/multi-user-target /lib/systemd/system/default.target

切换至图形界面：sudo systemctl set-default graphical.target

或者是：ln -sf /lib/systemd/system/graphical.target /lib/systemd/system/default.target

与fedora类似，但fedora在/etc/systemd/system/目录下

IP地址：192.168.18.128

## 数据库

MySQL数据库信息：muphy/root/…

Hadoop用户组：sudo groupadd hadoop

Hadoop用户名：sudo useradd -m azi –g hadoop

Hadoop用户密码：sudo passwd azi/123

添加到管理员：usermod -a -G hadoop,adm azi

计算机名：主节点-kali、从节点-kali\_s1/kali\_s2

修改方式：vim /etc/hostname 重命名后重启

添加主机名：vim /etc/hosts

## SSH配置

修改配置文件 /etc/ssh/sshd.config

#PermitRootLogin prohibit-password改为PermitRootLogin yes#prohibit-password

PasswordAuthentication yes前面的#好去掉

重启SSH服务：service ssh restart

开机自启动：update-rc.d ssh enable

免密登录配置

生成公钥：ssh-keygen

分发公钥：ssh-copy-id kali/kali\_s1/kali\_s2

## Hadoop环境

Hadoop下载链接：

https://hadoop.apache.org/releases.html

Hadoop安装

解压hadoop-2.9.2.tar.gz到/home/azi/

Hadoop基础文档：

http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

# 关于YARN

## HADOOP结构

MapRedurce

Spart

Tez

YARN

HDFS and HBase

Application

Compute

Storage

HIVE

PIG

Operate

## YARN运行机制

Resource manager node

Application

ResourceManage

NodeManage

Container

Application

NodeManage

Container

Application

node manager node

## 生命周期模型

三种模型：

1. 一个用户作业对应一个应用，MapRedurce采用的模型
2. 作业的每个工作流或者每个用户对话对应一个应用，可重用，并且效率高，Spart采用的模型
3. 多个用户共享一个长期运行的应用，通常作为协调者的角色运行，Slider和Impala采用的模型

## YARN的优点

1. 可扩展性更高，最多10000个节点和100000个任务
2. 可用性，对jobtracker的资源管理器和application master之间惊醒了职责划分
3. 利用率，YARN中的资源是精细化管理的
4. 多租户，向MapRedurce以外的分布式应用开发了Hadoop

## YARN中的调度

### 调度选项

三种调度规则

FIFI调度器

公平调度器

容量调度器

Job1

Job2

Time

Utilization

2

1

Time

Job1

Job2

Utilization

2

1

Time

Job1

Job2

Utilization

2

1

QUEUE B

FIFO QUEUE

QUEUE A

### 队列层次结构

root

prod

dev

eng

science

### FIFO调度器配置

简单易懂，不需要任何配置

### 容量调度配置

#### 文件名

Capacity-scheduler.xml

#### 队列放置

应用放在哪个队列却决于应用本身，比如MapRedurce可以通过mapreduce.job.queuename属性来指定队列

#### 文件内容

1. <?xml version="1.0">  
    <!-- 参数是通过yarn-scheduler.capacity.<queue-path>.<sub-property> 来进行设置的 -->  
   <configuration>  
    <property>  
    <name>yarn.scheduler.capacity.root.queues</name>  
    <value>prod,dev</value>  
    </property>  
    <property>  
    <name>yarn.scheduler.capacity.root.dev.queues</name>  
    <value>eng,science</value>  
    </property>  
    <property>  
    <name>yarn.scheduler.capacity.root.prod.capacity</name>  
    <value>40</value>  
    </property>  
    <property>  
    <name>yarn.scheduler.capacity.root.dev.capacity</name>  
    <value>60</value>  
    </property>  
    <property>  
    <!-- 设置dev队列最大容量为75% -->  
    <name>yarn.scheduler.capacity.root.dev.maxinum-capacity</name>  
    <value>75</value>  
    </property>  
    <property>  
    <name>yarn.scheduler.capacity.root.dev.eng.capacity</name>  
    <value>50</value>  
    </property>  
    <property>  
    <name>yarn.scheduler.capacity.root.dev.science</name>  
    <value>50</value>  
    </property>  
   </configuration>

### 公平调度器

#### 启用公平调度器

需要在yarn-site.xml文件中的yarn.resourccenamager.scheduler.class设置为公平调度器的完全限定名：org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler

#### 文件名

fair-scheduler.xml

可以通过设置yarn.scheduler.fair.allocation.file来修改文件名

#### 队列放置

公平调度器基于规则的系统来确定应用应该放到哪个队列（配置文件中的queuePlacementPolicy属性），省略此属性时默认遵从如下规则：

<defaultPlacementPolicy>

<rule name="specified" />

<rule name="user" />

</defaultPlacementPolicy>

也可以将所有的应用放在一个队列中以公平共享资源，等同于如下规则：

<defaultPlacementPolicy>

<rule name="default" />

</defaultPlacementPolicy>

#### 文件内容

1. <?xml version="1.0">  
   <allocations>  
    <!-- 队列的调度策略设置，默认就是公平调度器 -->  
    <defaultQueueSchedulingPolicy>fair</defaultQueueSchedulingPolicy>  
    <queue name="prod">  
    <weight>40</weight>  
    <!-- 设置队列级别的调用策略为FIFO -->  
    <schedulingPolicy>fifo</schedulingPolicy>  
    </queue>  
    <queue name="dev">  
    <weight>60</weight>  
    <!-- 设置嵌套队列 -->  
    <queue name="eng" />  
    <queue name="science" />  
    </queue>  
    <defaultPlacementPolicy>  
    <!-- 放在指明的队列中，没有指明队列，不允许创建队列，采用下一条 -->  
    <rule name="specified" create="false" />  
    <!-- 放在用户的主Unix组名命名的队列中，不存在这样的队列，也不允许创建，采用下一条 -->  
    <rule name="primaryGroup" create="false" />  
    <!-- 最底的队列设置，前面的都不成立时，采用此默认队列 -->  
    <rule name="default" queue="dev.eng" />  
    </defaultPlacementPolicy>  
   </allocations>

# HADOOP分布式文件系统

## 配置文件

### Hdfs-site.xml

1. <?xml version="1.0" encoding="UTF-8"?>  
   <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  
   <configuration>  
    <property>  
    <name>dfs.replication</name>  
    <value>1</value>  
    </property>  
   </configuration>

### Yarn-site.xml

1. <?xml version="1.0"?>  
   <configuration>  
    <property>  
    <name>yarn.resourcemanager.hostname</name>  
    <value>localhost</value>  
    </property>  
    <property>  
    <name>yarn.nodemanager.aux-services</name>  
    <value>mapreduce\_shuffle</value>  
    </property>  
   </configuration>

### Core-site.xml

1. <?xml version="1.0" encoding="UTF-8"?>  
   <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  
   <configuration>  
    <property>  
    <name>fs.defaultFS</name>  
    <value>hdfs://localhost:8020</value>  
    </property>  
   </configuration>

### mapred-site.xml

1. <?xml version="1.0"?>  
   <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  
   <configuration>  
    <property>  
    <name>mapreduce.framework.name</name>  
    <value>yarn</value>  
    </property>  
   </configuration>

## 网络拓扑结构

d1 data center d2

r1 机架 r2

r3 rack

n1 d=0

n3 d=4

n2 d=2

node

n4 d=6

## 数据流

### MapReduce数据流

Input HDFS

Copy

Split 0

Map

Split 1

Map

Split 2

Map

Reduce

Part 0

Merage

Reduce

Part 1

Merage

Sort

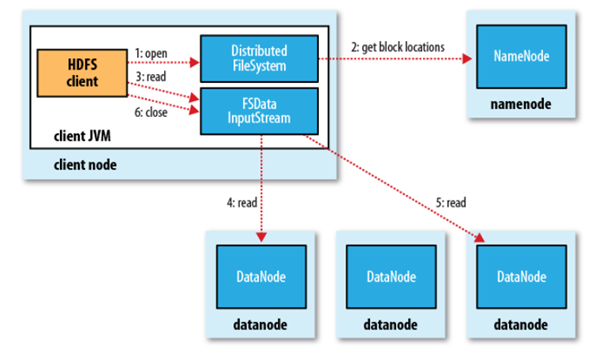
Sort

Sort

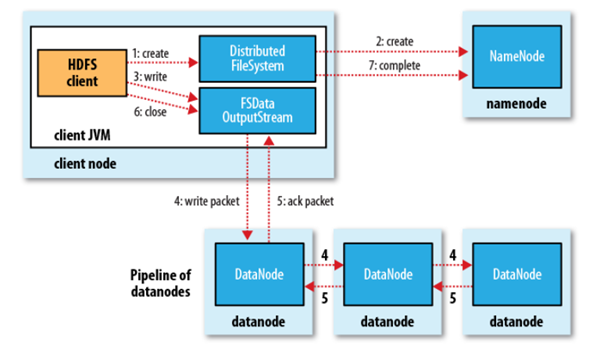
可能没有，或者一个或者多个redurce

Output HDFS

### 读取过程



### 写入过程



## 基本操作

1. 复制

% hadoop fs –copyFromLocal input/file \ hdfs://localhost/user/azi/file

1. 并行复制

% hadoop distcp –update –delete -p hdfs://namenode1/foo webhdfs://namenode2:50070/foo

1. 建文件夹

% hadoop fs –mkdir books

1. 查看文件

% hadoop fs –ls .

## HADOOP文件系统

|  |  |  |  |
| --- | --- | --- | --- |
| 文件系统 | URI方案 | Java实现  （org.apache.hadoop） | 描述 |
| Local | file | fs.LocalFileSystem | 使用客户端校验和的本地磁盘文件系统。使用RawLocalFileSystem表示无校验和的本地磁盘文件系统 |
| HDFS | hdfs | hdfs.DistributionFileSystem | Hadoop的分布式文件系统。将HDFS设计成与MpRedurce结合使用，可以实现高性能 |
| WebHDFS | Webhdfs | Hdfs.web.WebHdfsFileSystem | 基于HTTP的文件系统，提供对HDFS的认证读/写访问 |
| Secure WebHDFS | swebhdfs | Hdfs.web.SWebHdfsFileSystem | WebHDFS的HTTPS版本 |
| HAR | har | fs.HarFileSystem | 构建在Hadoop文件系统之上，对文件进行归档。Hadoop归档文件主要用来**减少**NameNode的**内存使用**。 |
| View | viewfs | viewfs.ViewFileSystem | 针对其他Hadoop文件系统的客户端挂载表，通常用于为联邦namenode创建挂载点 |
| FTP | ftp | fs.ftp.FtpFileSystem | 由FTP服务器支持的文件系统 |
| S3 | s3a | fs.s3a.S3AFileSystem | 由Amazon S3支持的文件系统，替代老版的s3n(s3 原生)实现 |
| Azure | wasb | fs.azure.NativeAzureFileSystem | 由Microsoft Azure支持的文件系统 |
| Swift | swift | fs.swift.snative.SwiftNativeFileSystem | 由OpenStack Swift支持的文件系统 |

# Hadoop的IO操作

## 压缩

减少存储文件磁盘空间，加速数据在网络和磁盘上的传输。

## Codec/CodecPool

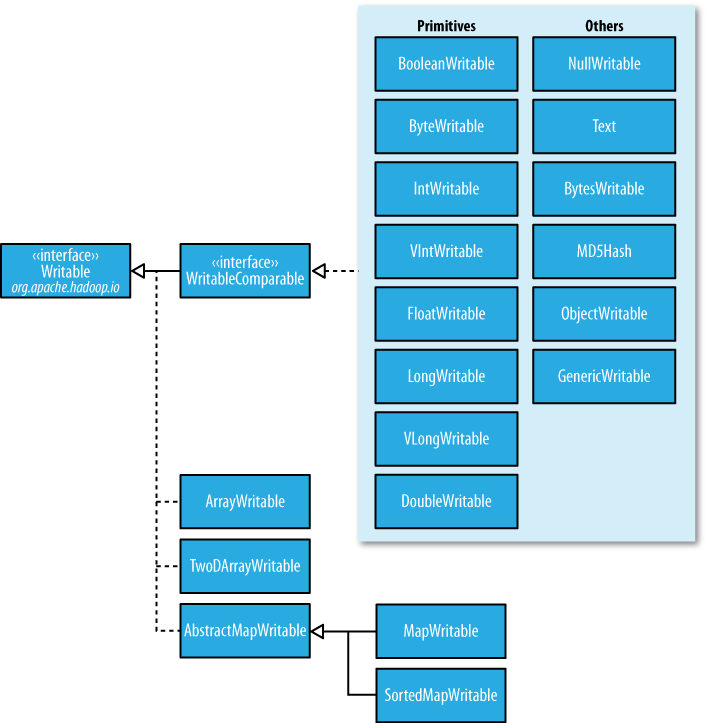
是压缩-解压算法（DEFLATE、gzip、bzip2、LZO、LZ4、Snappy）的一种实现，大文件最好使用支持切分的压缩格式（bzip2、LZO(通过所以实现)、LZ4、Snappy）

## 序列化

将结构化对象转换为字节流方便网络传输和磁盘存储，反序列化为逆过程

处理两大领域：进程间通信和永久存储

## Writable



## 文件数据结构

### 面向行的文件格式

适合同时处理一行中较多列的情况

1. SequenceFile

为二进制键值对提供了持久的数据结构，通过SequenceFile.Writer实例写，通过SequenceFile.Reader实例读

Header

Record

Sync

Record

Record

Sync

Record

Key length

Record length

Key

Compressed value

4

4

Record compression

Key length

Record length

Key

Value

4

4

No compression

SEQ

版本号

3

1

同步点

**正常顺序文件**

Header

Sync

Block

Sync

Block

Sync

Block

Compressed

keys

Compressed

key lengths

Compressed

value lengths

Compressed

values

**块压缩顺序文件**

Number of

recdrds

1-5

1. MapFile

已经排过序的SequenceFile，变种：SetFile（顺序存储键的集合）、ArrayFile（键是元素的索引，值是Writable

1. Avro

面向大规模数据处理的设计（紧凑且可分），可移植（文件中存储的对象使用模式来描述而不是JAVA代码）

### 面向列的文件格式

适合对那些只访问表中一部分列的查询

# MapReduce应用开发

## 测试驱动程序

1. 使用本地作业运行器，在本地文件系统的测试文件上运行作业
2. 使用mini集群来运行，Hadoop有一组测试类（MiniDFSCluster、MiniMRCluster和MiniYARNCluster），以程序方式创建正在运行的集群：

实例代码:

Hadoop jar $HADOOP\_HOME/share/Hadoop/mapreduce/hadoop-mapreduce-\*-tests.jar minicluster

## MapReduce的Web界面

页面网址：<http://resource-manager-host:8088/>

1. 资源管理器页面(Cluster Metrics)
2. MapReduce作业页面(Tracking UI)
3. 任务和任务尝试页面(map)

## MapReduce工作流

### Job控制

如果处理的过程比较复杂，通常是将问题分解成更多的作业而不是增加作业的复杂度。

结合Hadoop自带的ChainMapper可以在一个MapReduce作业中运行的一系列mapper，再运行一个reducer和另一个mapper链。

线性链表作业：

JobClient.runJob(conf1);

JobClient.runJob(conf2);

新API：

bool r = waitForCompletion();

### 关于Apache Oozie

Oozie是运行工作流的系统，运行开始和结束都可以通过HTTP的回调向客户端发送工作流的状态，由两部分组成：

1. 一个工作流引擎，负责不同类型的Hadoop作业存储和运行
2. 一个coordinator引擎，负责基于预定义的调度策略及数据可用性运行工作流作业

### 定义Oozie工作流workflow.xml

<workflow-app xmlns="uri:oozie:workflow:0.1" name="max-temp-workflow">  
 <!-- 开始控制节点 -->  
 <start to="max-temp-mr"/>  
 <!-- 动作控制节点 -->  
 <action name="max-temp-mr">  
 <map-reduce>  
 <job-tracker>${resoureManager}</job-tracker>  
 <name-node>${nameNode}</name-node>  
 <prepare>  
 <delete path="${nameNode}/user/${wf:user()}/output"/>  
 </prepare>  
 <configuration>  
 <property>  
 <name>mapred.mapper.new-api</name>  
 <value>true</value>  
 </property>  
 <property>  
 <name>mapred.reducer.new-api</name>  
 <value>true</value>  
 </property>  
 <property>  
 <name>mapreduce.job.map.class</name>  
 <value>MaxTempratureMapper</value>  
 </property>  
 <property>  
 <name>mapreduce.job.reduce.class</name>  
 <value>MaxTempratureReducer</value>  
 </property>  
 <property>  
 <name>mapreduce.job.output.key.class</name>  
 <value>org.apache.hadoop.io.Text</value>  
 </property>  
 <property>  
 <name>mapreduce.job.output.value.class</name>  
 <value>org.apache.hadoop.io.IntWritable</value>  
 </property>  
 <property>  
 <name>mapreduce.input.fileinputformat.inputdir</name>  
 <value>/user/${wf:user()}/input/ncdc/micro</value>  
 </property>  
 <property>  
 <name>mapreduce.input.fileinputformat.outputdir</name>  
 <value>/user/${wf:user()}/output</value>  
 </property>  
 </configuration>  
 </map-reduce>  
 <ok to="end"/>  
 <error to="fail"/>  
 </action>  
 <!-- kill控制节点 -->  
 <kill name="fail">  
 <message>MapRedurce failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>  
 </kill>  
 <!-- end控制节点 -->  
 <end name="end"/>  
</workflow-app>

### Oozie工作流转移图

ok

error

### 打包和配置Oozie工作流

Max-temp-workflow/

Lib/

Hadoop-examples.jar

workflow.xml

将这个文件夹复制到HDFS：

hadoop fs –put hadoop-examples/target/max-temp-workflow max-temp-workflow

必须在顶层

### Oozie运行工作流

1. 设置环境变量：

export OOZIE\_URL=<http://localhost:11000/oozie>

1. 运行：

oozie job –config ch06-mr-dev/src/main/resources/max-temp-workflow.properties –run

job: 0000001-140911033236841-oozie-oozi-w

max-temp-workflow.properties文件内容:

nameNode=hdfs://localhost:8032

resourceManager= localhost:8032

oozie.wf.application.path=${nameNode}/user/${user.name}/max-temp-workflow

1. 查看状态信息：

oozie job –info 0000001-140911033236841-oozie-oozi-w

1. 检查结果：

hadoop fs –cat output/part-\*

# MapReduce作业运行机制

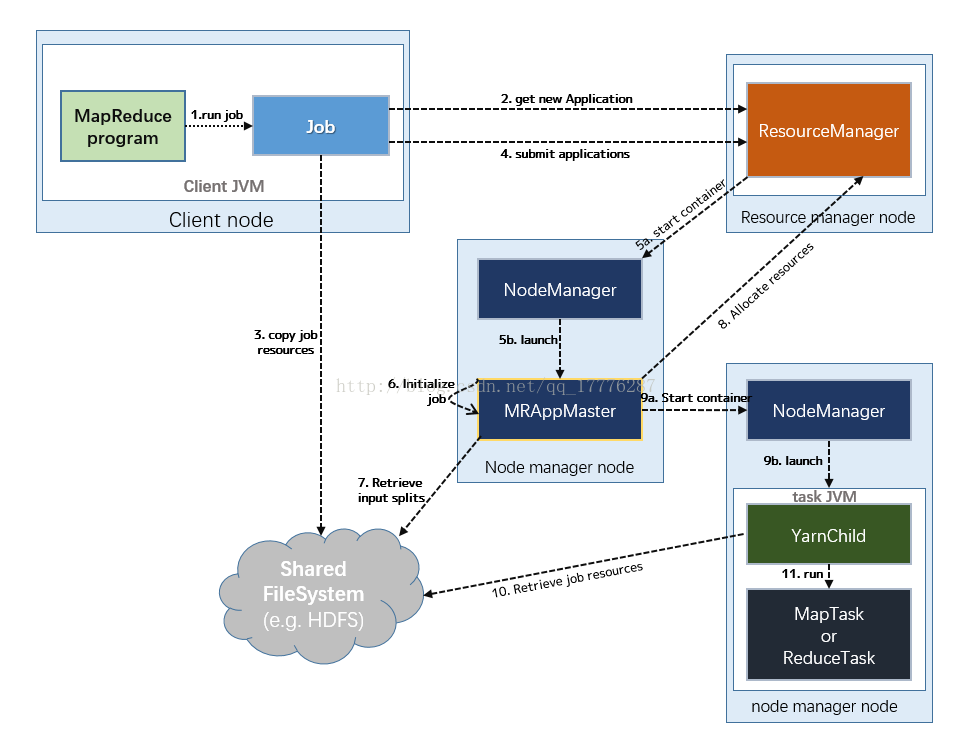
## 运行MapReduce作业

1. Job对象的submit()方法：封装处理细节
2. waitForCompletion()方法：用于提交以前没有提交过的作业，并等待作为完成。

## 5个独立实体

1. 客户端：提交MapReduce作业。
2. YARN资源管理器：负责协调集群上计算机资源的分配。
3. YARN节点管理器：负责启动和监视集群中机器上的计算容器（container）
4. MapReduce的application master：负责协调运作MapReduce作业的任务，它和MapReduce任务在容器中运行，这些容器由资源管理器分配，并由节点管理器进行管理。
5. 分布式文件系统（HDFS）：用来与其他实体间共享作业文件。

## MapReduce作业工作原理



### 作业提交

步骤1-4

请求ID，检查输出目录，计算输入分片，复制所需资源(mapreduce.client.submit.file.replication 默认值10)，通过submitApplication()方法提交作业，有异常就不提交。

### 作业初始化

步骤5-7，没有第八步的作业称为uber任务，参数可设置：

mapreduce.job.ubertask.maxmaps 默认遵10

mapreduce.job.ubertask.maxreduces 默认值1

mapreduce.job.ubertask.maxbytes 默认值1个块大小

### 任务分配

步骤8，内存和需求都可通过4个属性设置：

mapreduce.map.memory.mb 默认值1024

mapreduce.reduce.memory.mb 默认值1

mapreduce.map.cpu.vcores 默认值1024

mapreduce.reduce.cpu.vcores 默认值1

### 任务执行

步骤9-10

Streaming运行特殊的map任务和reduce任务，使用标准的输入输出流与进程通信

nodeManage

YARNChild

MapTask or ReduceTask

Run

Task JVM

Streaming process

Launch

Launch

Std in

key/value

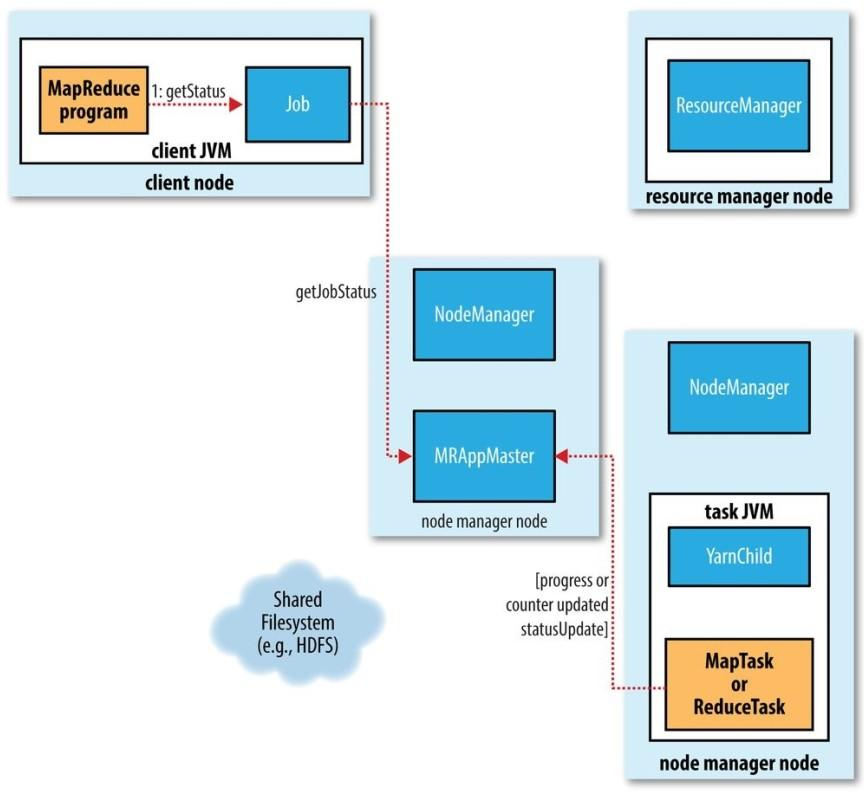
Std out

key/value

### 进度状态更新

Map或者Reduce任务运行时子进程每个3秒通过umbilical接口向父进程发送状态和计数值；

客户端没1秒(mapreduce.client.progressmonitor.pollinterval 默认值3)轮询一次application master以接收最新状态，也可以使用Job的getStatus()方法得到一个JobStatus的实例(包含所有状态)



### 作业成功

Application master轮询到成功通知，在控制台输出统计信息，通过HTTP通知客户端（如有设置mapreduce.job.end-notification.url），清理和存档到历史服务器

### 作业失败

1. 任务失败

Streaming任务失败：退出代码为非零（stream.non.zero.exit.is.failure 默认值true）

超时失败：mapreduce.task.timeout 默认值10\*60\*1000，为0失效

失败尝试：mapreduce.map/reduce.maxattempts 默认值4

失败百分比：mapreduce.map/reduce.failures.maxpercent 默认值100%

1. Application master失败

Mapreduce application master尝试次数：mapreduce.am.max-attempts 默认值2，大于2时必须修改YARN application master尝试次数

YARN application master尝试次数：yarn.resourcemanager.am.max-attempts 默认值2

Mapreduce application master重试：yarn.app.mapreduce.am.job.recovery.enable 默认值true

1. 节点管理器失败

心跳超时：yarn.resourceManager.nm.liveness-monitor.expiry-intever-ms 默认值10\*60\*1000

节点任务失败数(可能被资源管理器拉黑)：mapreduce.job.maxtaskfailure.per.tracker 默认值3

1. 资源管理器失败，严重的问题，任务无法启动

双机热备(一对资源管理器)通过故障转移控制器转换

高可用状态存储区

## Shuffle和排序

### Map端

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 值类型 | 默认值 | 说明 |
| mapreduce.task.io.sort.mb | int | 100 | 用于map输出排序的内存大小 |
| mapreduce.map.sort.spill.percent | float | 0.80 | 开始spill的缓冲池阈值 |
| mapreduce.task.io.sort.factor | int | 10 | 合并文件数最大值，与reduce共用 |
| mapreduce.map.combine.minspills | int | 3 | 运行combiner的最低spill文件数 |
| mapreduce.map.out.compress | boolean | false | 输出是否压缩 |
| mapreduce.map.out.compress | Class | DefaultCodec | 压缩算法 |
| mapreduce.shuffle.max.threads | int | 0 | 服务于reduce提取结果的线程数量 |

缓冲区大小：mapreduce.task.io.sort.mb 默认值100

阈值：mapreduce.map.sort.spill.percent 默认值 0.80

溢出写入目录：mapreduce.cluster.local.dir

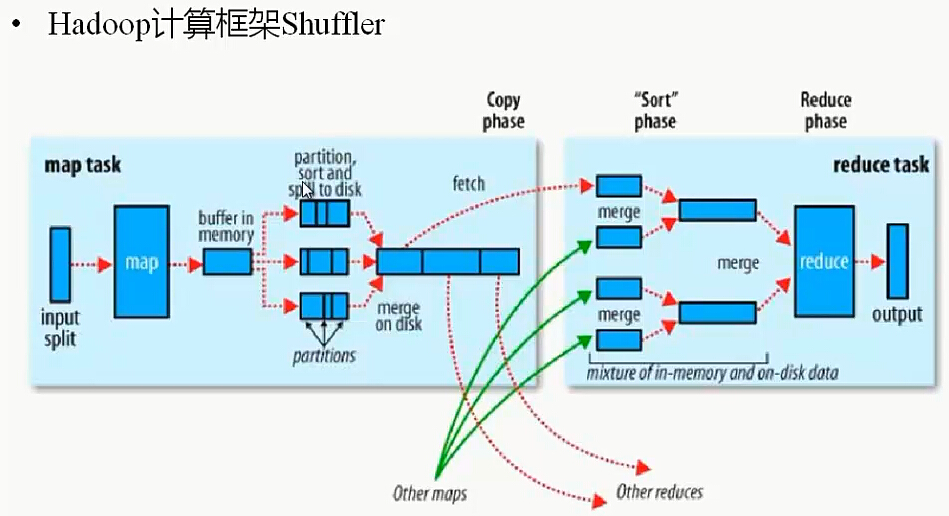
溢出文件合并流数：mapreduce.task.io.sort.factor 默认值10，很常见增加到100

Combine函数再次运行所需溢出文件数：mapreduce.map.combine.minspills 默认值3

输出压缩：mapreduce.map.output.compress 默认值false

压缩格式：mapreduce.map.output.compress.codec 默认值org.apach.hadoop.io.compress.DefaultCode

节点工作线程数：mapreduce.shuffle.max.threads 默认值0，表示设为节点处理器的两倍



### Reduce端

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 值类型 | 默认值 | 说明 |
| mapreduce.reduce.shuffle.parallelcopies | int | 5 | 提取map输出的copier线程数 |
| mapreduce.reduce.shuffle.maxfetchfailures | int | 10 | 提取map输出最大尝试次数，超出后报错 |
| mapreduce.task.io.sort.factor | int | 10 | 合并文件数最大值，与map共用 |
| mapreduce.reduce.shuffle.input.buffer.percent | float | 0.70 | copy阶段用于保存map输出的堆内存比例 |
| mapreduce.reduce.shuffle.merge.percent | float | 0.66 | 开始spill的缓冲池比例阈值 |
| mapreduce.reduce.shuffle.inmem.threshold | int | 1000 | 开始spill的map输出文件数阈值，小于等于0表示没有阈值，此时只由缓冲池比例来控制 |
| mapreduce.reduce.input.buffer.percent | float | 0.0 | reduce函数开始运行时，内存中的map输出所占的堆内存比例不得高于这个值，默认情况内存都用于reduce函数，也就是map输出都写入到磁盘 |

1. 复制阶段

复制线程数：mapreduce.reduce.shuffle.parallelcopies 默认值5

Reduce堆内存的阈值：mapreduce.reduce.shuffle.input.buffer.percent

reduce缓冲区阈值：mapreduce.reduce.shuffle.merge.percent

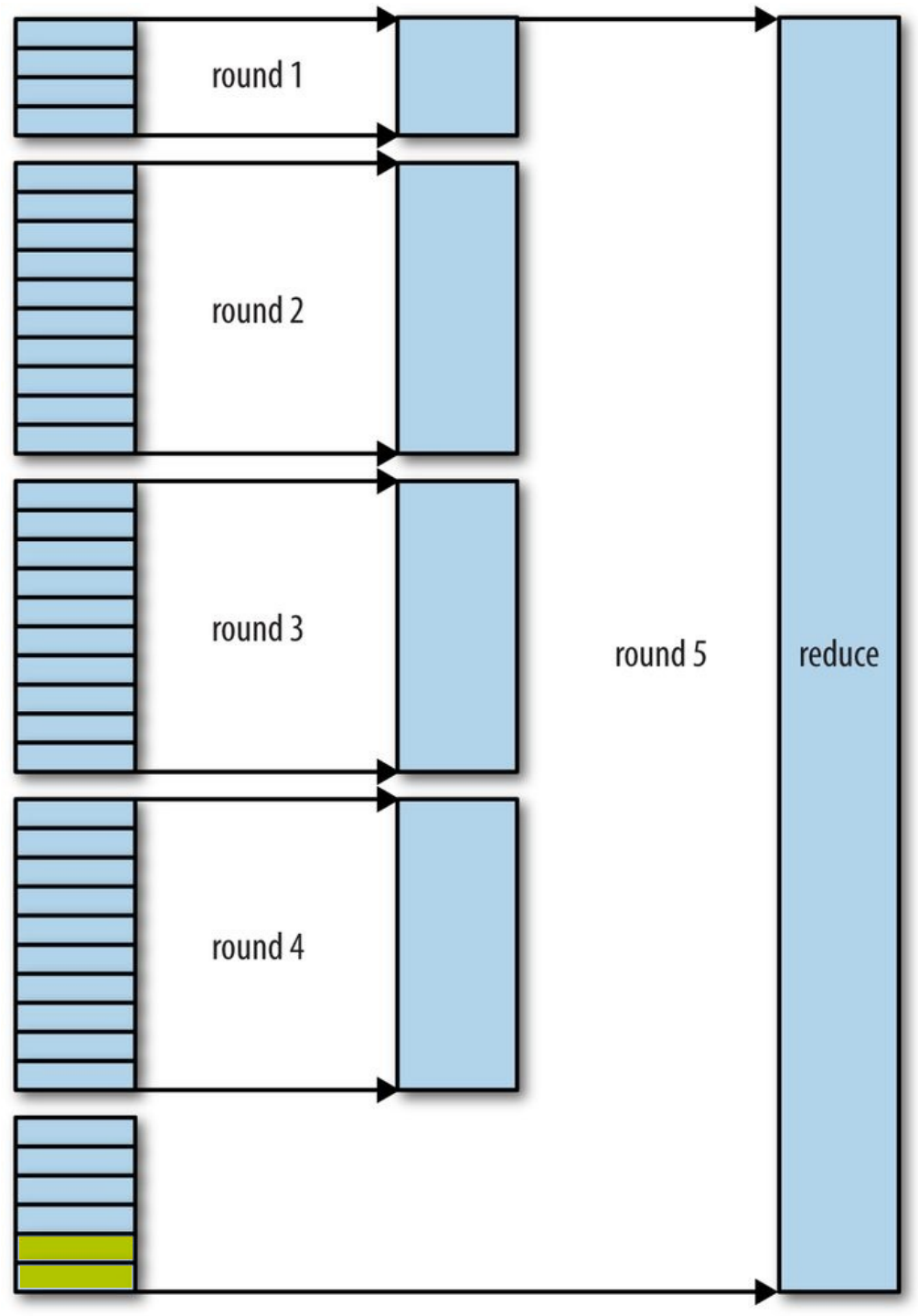
Map输出阈值：mapreduce.reduce.merge.inmem.threshold

未到达这些阈值时复制到reduce的JVM内存，否则输出到磁盘

1. 排序（合并）阶段

合并因子：mapreduce.task.io.sort.factor 默认值10

1. Reduce阶段



1. 配置调优

总的原则是给shuffle过程尽量多提供内存空间，最小化访问磁盘的次数，但也要确保map和reduce函数能得到足够的内存来运行（避免在map中堆积数据）

Map和reduce的JVM内存大小：mapred.child.java.opts

## 任务的执行

### 任务执行环境属性

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 值类型 | 说明 | 范例 |
| mapreduce.job.id | String | 作业ID | Job\_200811201130\_0004 |
| mapreduce.task.id | String | 任务ID | task\_201811201130\_0004\_m\_000003 |
| mapreduce.task.attemp.id | String | 任务尝试ID | attempt\_2018112011300004\_m\_000003\_0 |
| mapreduce.task.partition | int | 作业中任务的索引 | 3 |
| mapreduce.task.ismap | boolean | 任务是否map任务 | true |

### Streaming环境变量

用下划线代替非字母数字

### 推测执行

Hadoop会尽量检测比预期慢的任务，并启动另一个系统任务作为备份

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 值类型 | 默认值 | 说明 |
| mapreduce.map.speculative | Boolean | True | 如果任务运行变慢，该属性决定着是否要启动map任务的另外一个实例 |
| mapreduce.reduce.speculative | Boolean | True | 如果任务运行变慢，该属性决定着是否要启动reduce任务的另外一个实例 |
| Yarn.app.mapreduce.am.job.speculator.class | Class | Org.apache.hadoop.map.redduce.v2.app.speculate.DefaultSpeculator | Speculator类实现推测执行策略(只针对MapReduce2) |
| Yarn.app.mapreduce.am.job.estimator.class  （遗留任务运行时估计量） | Class | Org.apache.hadoop.map.redduce.v2.app.speculate.legacyTaskRuntimeEstimator | Speculator实例使用的TaskRuntimeEstimator的实现，提供任务运行时间的估值(只针对MapReduce2) |

### OutputCommitters

作业或者任务的输出目录：mapreduce.output.fileoutputformat.outputdir

任务工作目录：mapreduce.task.out.put.dir/FileOutputFormat.getWirkOutputPath()

setupJob

commitJob

abortJob

\_SUCCESSS

\_temporary

工作空间

Ok

Ok

setupTask

needsTaskCommit

commitTask

abortTask

True

JOB

TASK

# MapReduce类型和格式

## 常规格式

map:(K1,V1) -> list(K2,K2) 输入输出类型可以不同

combine:(K2,list(V2)) -> list(K2,V2) 输入输出类型都是map输出类型

reduce:(K2,list(V2)) -> list(K3,V3) 输入输出类型可以不同，但输入是map输出类型

partition:(K2,V2) -> integer 输入处理K2和V2，输出返回分区索引

## 接口

public interface Mapper<K1,V1,K2,V2> extends JobConfigurable, Closeable {  
 void map(K1 key, V1 value, OutputCollector<K2, V2> output, Reporter reporter) throws IOException;  
}  
public interface Reducer<K2,V2,K3,V3> extends JobConfigurable, Closeable {  
 void reduce(K2 key, Iterator<V2> values, OutputCollector<K3, V3> output, Reporter reporter) throws IOException;  
}  
public interface Partitioner<K2,V2> extends JobConfigurable {  
 void getPartition(K2 key, V2 value, int numPartitions) throws IOException;  
}

## MapReduce API设置类型

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 属性 | 属性设置方法 | K1 | V1 | K2 | V2 | K3 | V3 |
| 可以设置类型的属性 |  | \* | \* |  |  |  |  |
| mapreduce.job.inputformat.class | setInputFormatClass() |  | \* |  |  |  |  |
| mapreduce.map.output.key.class | setMapOutputKeyClass() |  |  | \* |  |  |  |
| mapreduce.map.output.value.class | setMapOutputValueClass() |  |  |  | \* |  |  |
| mapreduce.job.output.key.class | setOutputKeyClass() |  |  |  |  | \* |  |
| mapreduce.job.output.value.class | setOutputValueClass() |  |  |  |  |  | \* |
| 类型必须一致的属性 |  |  |  |  |  |  |  |
| mapreduce.job.map.class | setMapperClass() | \* | \* | \* | \* |  |  |
| mapreduce.job.combine.class | setCombinerClass() |  |  | \* | \* |  |  |
| mapreduce.job.partitioner.class | setPartitionerClass() |  |  | \* | \* |  |  |
| mapreduce.job.output.key.comparator.class | setSortComparatorClass() |  |  |  |  |  |  |
| mapreduce.job.output.group.comparator.class | setGroupingComparatorClass() |  |  | \* |  |  |  |
| mapreduce.job.reduce.class | setReducerClass() |  |  | \* | \* | \* | \* |
| mapreduce.job.outputformat.class | setOutputFormatClass() |  |  |  |  | \* | \* |

package me.muphy.MinimalMapReduceWithDefaults;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.conf.Configured;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  
import org.apache.hadoop.mapreduce.lib.partition.HashPartitioner;  
import org.apache.hadoop.util.Tool;  
import org.apache.hadoop.util.ToolRunner;  
  
public class MinimalMapReduceWithDefaultsDriver extends Configured implements Tool {  
 public int run(String[] strings) throws Exception {  
 Job job = JobBuilder.*parseInputAndOutput*(this, new Configuration(), strings);  
 if (job == null) {  
 return -1;  
 }  
 job.setInputFormatClass(TextInputFormat.class);  
 job.setMapperClass(Mapper.class);  
 job.setMapOutputKeyClass(LongWritable.class);  
 job.setMapOutputValueClass(Text.class);  
 job.setPartitionerClass(HashPartitioner.class);  
 job.setNumReduceTasks(1);  
 job.setReducerClass(Reducer.class);  
 job.setOutputKeyClass(LongWritable.class);  
 job.setOutputValueClass(Text.class);  
 job.setOutputFormatClass(TextOutputFormat.class);  
 return job.waitForCompletion(true) ? 0 : 1;  
 }  
 public static void main(String[] args) throws Exception {  
 int exitCode = ToolRunner.*run*(new MinimalMapReduceWithDefaultsDriver(), args);  
 System.*exit*(exitCode);  
 }  
}

## 默认的Streaming作业

$ hadoop jar $HADOOP\_HOME/share/Hadoop/Hadoop/tools/lib/Hadoop-streaming-\*.jar \

-input input/ncdc/sample.txt \

–output output \

-inputformat org.apache.hadoop.mapred.TextInputFormat \ #默认值

–mapper /bin/cat \

-combiner \*\*\*

-D \*\*\*

-partitioner org.apache.hadoop.mapred.lib.HashPartitioner \ #默认值

-numReduceTask 1 \ #默认值

-reducer org.apache.hadoop.mapred.lib.IdentityReducer \ #默认值

-outputformat org.apache.hadoop.mapred.TextOutputFormat \ #默认值

-io text #默认值

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 类型 | 默认值 | 描述 |
| stream.map.input.ignorekey | bool | true | 忽视键输入 |
| mapreduce.output.textoutputformat.separator |  | \t | TextOutputFormat |
| stream.map.input.field.separator | String | \t | map输入的key与value分隔符 |
| stream.map.output.field.separator | String | \t | map输出的key与value分隔符 |
| stream.num.map.output.key.fields | int | 1 | map中分隔符的位置 |
| stream.reduce.input.field.separator | String | \t | reduce输入key与value分隔符 |
| stream.reduce.output.field.separator | String | \t | reduce输出key与value分隔符 |
| stream.num.reduce.output.key.fields | int | 1 | reduce中分隔符的位置 |

MapTask

ReduceTask

Streaming proccess

Streaming process

Stream.map.input.field.separator

Stream.map.output.field.separator

Stream.reduce.output.field.separator

Stream.reduce.input.field.separator

shuffle

Std in

Std in

Std out

Std out